



TERADATA ASTER DISCOVERY PORTFOLIO

SRI RAGHAVAN
Senior Global Product Marketing Manager
Teradata

September 2014



Table of Contents

Introduction	5
Teradata Aster Discovery Portfolio.....	5
Data Preparation and Transformation Functions.....	8
Antiselect	8
Apache Log Parser	8
E-Mail Parser.....	9
Identity Matching	10
IpGeo Mapping	12
JSON Parser.....	13
Multicase	14
Murmurhash	15
Outlier Filter.....	16
Pack and Unpack.....	17
Pivot and Unpivot	19
Sampling	21
Sessionization	22
XML Parser	24
Graph Functions.....	26
All Pairs Shortest Path.....	26
Betweenness.....	29
Closeness	32
Eigenvector Centrality	34
K-Degree Centrality.....	36
Local Clustering Coefficient	38
Loopy Belief Propagation.....	40
Pagerank	42
Path and Pattern Discovery Functions.....	44
Attribution	44

Basket Generator	45
Collaborative Filter	46
Frequent Paths (aka Sequential Paths).....	48
nTree	49
Path Generator, Path Starter, and Path Summarizer	50
Teradata Aster nPath	54
WSRecommender	55
Statistics and Machine Learning Functions	58
Approximate Distinct Count	58
Average (Simple, Moving, Weighted)	59
Canopy	59
Confusion Matrix	60
Correlation	61
Distribution Matching.....	62
F-Measure.....	64
Generalized Linear Model (GLM).....	65
Histogram	66
K-Means (Clustering and Plotting).....	67
K-Nearest Neighbor Classification Algorithm	67
Least Absolute Shrinkage and Selection Operator (LASSO).....	69
Linear Regression.....	70
Logistic Regression.....	71
Minhash	72
Naïve Bayes Classifier	73
Principal Component Analysis (PCA)	75
Percentile (Actual and Approximate)	76
Random Forest.....	77
Single Decision Tree.....	79
Support Vector Machines (SVM)	81
Text Analytics and Sentiment Extraction Functions.....	83
Chinese Text (Word) Segmentation.....	83
Latent Dirichlet Allocation (LDA)	83
Levenshtein Distance	85

Naïve BayesText Classifier	86
Named Entity Recognition (NER)	87
nGram	89
Sentiment Extraction	90
Attensity® Text and Sentiment Analytics Functions.....	92
Text Categorization/Classification	93
Text Chunker	95
Text Parser	96
Time Series Analysis Functions	98
Wavelet Transformations (Discrete, 2D, and Inverse)	98
Dynamic Time Warping.....	100
Symbolic Aggregation Approximation	101
Visualization Functions	104
Collaborative Filter Visualization	104
Teradata Aster nPath Visualization	105
Writing custom SQL-MR functions in Aster	107
Teradata Aster and R Language Integration	108
Conclusion.....	109

Introduction

Enterprises today see exploding volumes of multi-structured data and are uniquely positioned to discover novel insights from these data that deliver a strong competitive edge. Consequently, the focus has been on homing in on the best solution that not only manages voluminous data but also one that delivers critical insights rapidly. Such a solution would be a massively parallel deployment that traditionally has required extremely specialized programming skills. Technologies such as MapReduce provide a framework to help address the challenges of parallel programming though MapReduce is a complex programming and management framework itself. The challenges of complex programming, limited flexibility and reusability, and difficult integration create significant obstacles to development, testing, and deploying rich big data analytics and discovery solutions at scale. Until now, data scientists, statisticians, and business analysts have seen high cost overruns and undue latencies in meeting a growing demand for big data analytics and discovery.

The Teradata Aster Discovery Platform has been developed with these challenges in mind, and its aim is to make it easier for all people to discover crucial business insights from all data types with rapid iterations. The Discovery Platform is designed for all types of enterprise users to derive novel, high-value business insights through synergistic multi-genre analytics that invoke various state-of-the-art tools (Teradata SQL-MapReduce® (SQL-MR), Text, SQL-Graph® (SQL-GR), and Statistical functions). It also features the industry's first and expanding portfolio of more than 100 pre-built SQL, Graph, and MapReduce functions. It comes with out-of-the box functionality for data acquisition, preparation, analysis, and visualization, and it enables users to incorporate all four types of functions within a single SQL statement. It can be easily used by any SQL-savvy analyst, as well as being powerful and flexible enough for the most technically sophisticated data scientists. The Teradata Aster Discovery Platform is the market leading discovery platform that provides a complete solution for visual, interactive, and powerful analytic applications that require minimal time and effort to derive value from. It comprises the Teradata Aster Database and the Teradata Aster Discovery Portfolio.

The focus of this white paper is on providing a description of the SQL-MR and SQL-GR functions and to create the context in which they are widely implemented today across many Teradata customers. The white paper is written with the business executives, lines of business managers, and business analysts in mind.

Teradata Aster Discovery Portfolio

The Teradata Aster Discovery Portfolio is a complete solution that powers interactive applications to drive insights that strongly impact strategic and tactical decisions. It helps C-level executives, business managers, IT managers, data scientists, statisticians, and business analysts to discover and operationalize breakthrough business insights to:

- Implement process refinements and/or new services and other activities that increase customer satisfaction.
- Inculcate a data-driven ethos in which hard evidence is used to make key strategic decisions (e.g., pricing).
- Minimize risk and capitalize on emerging market trends by making decisions proactively.

The Aster Discovery Portfolio (see Figure1) includes a suite of ready-to-use SQL-MapReduce and SQL-Graph (SQL-MR and SQL-GR) functions that can be invoked from a SQL interface without having to do any programming. There are four modules of SQL-MR functions (100+ in total) for each step of the discovery process: **Data Acquisition, Data Preparation, Analytics, and Visualization**. The four processes are called from within a single SQL statement thus eliminating the need to juggle these with disparate tools and skillsets, and siloed metadata. This accelerates analytic innovation with rapid iterations and delivers new insights in minutes as opposed to days, weeks, or even months with traditional approaches.

TERADATA ASTER DISCOVERY PORTFOLIO

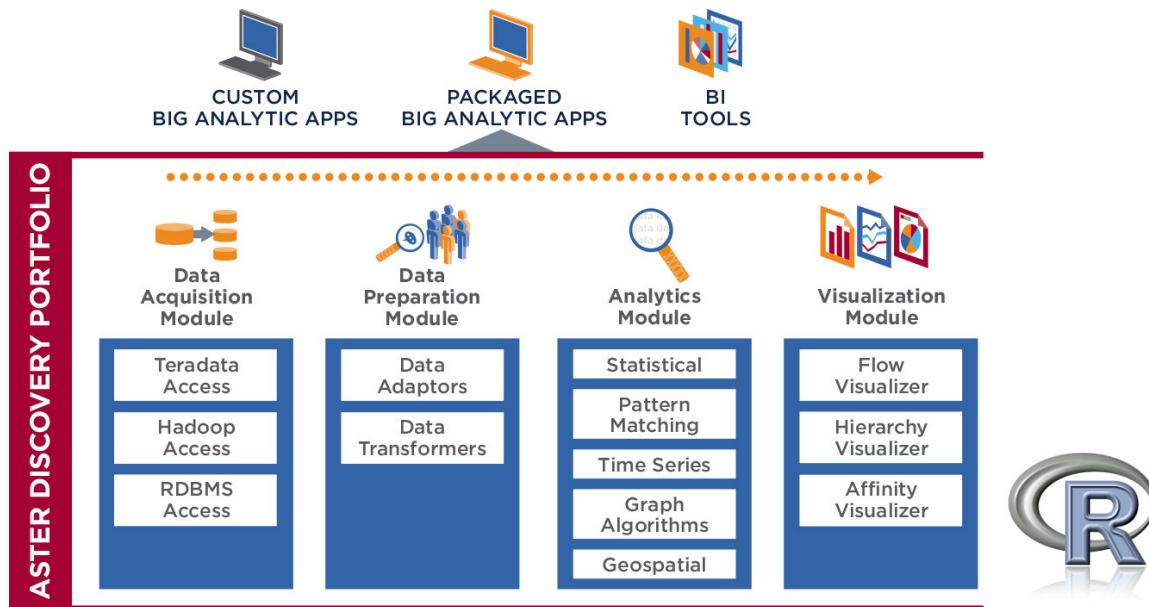


Figure 1: Teradata Aster Discovery Portfolio and R Integration

Simplify and accelerate development and deployment: The Teradata Aster Discovery Portfolio provides a set of reusable building blocks that can be leveraged by various analytic applications. Rather than spending time and effort on writing and testing custom code for common analytic functionality, analysts can easily use the pre-built functions through the SQL-MR framework so that they can focus on the higher-level logic that is unique to their applications.

Deliver fast and scalable analytic insights: The analytic functions included in the Aster Discovery Portfolio have been designed to be highly performant and scalable. They take full advantage of the Teradata Aster Database system, by leveraging its embedded MapReduce engine, SQL, SQL-MR, and SQL-GR frameworks. The inherent simplicity of the architecture makes it easier to understand and implement. This means that analysis can be done without having to sample data sets but instead on entire data sets.

Enable richer analytics: The Teradata Aster Discovery Portfolio (See Figure 2) includes 100+ advanced analytic functions that address use cases in telecommunications, healthcare, pharmaceuticals, finance and insurance, manufacturing, and in many more verticals. The use cases include fraud and risk analysis, churn, manufacturing optimization, golden path analysis, marketing attribution, sentiment extraction, and more. Analysts of all types can match their expertise to the flexibility of these functions in a manner that allows them to iteratively build on critical insights. The goal of the Discovery Portfolio is to enable multi-genre analytics (e.g., Teradata Aster nPath™ and Naïve Bayesian functions can be applied together to determine churn likelihood) as well as to introduce a rapid iterations approach to decision making that is founded on hypothesis testing.

TERADATA ASTER DISCOVERY PORTFOLIO

Sample SQL-MR Functions:

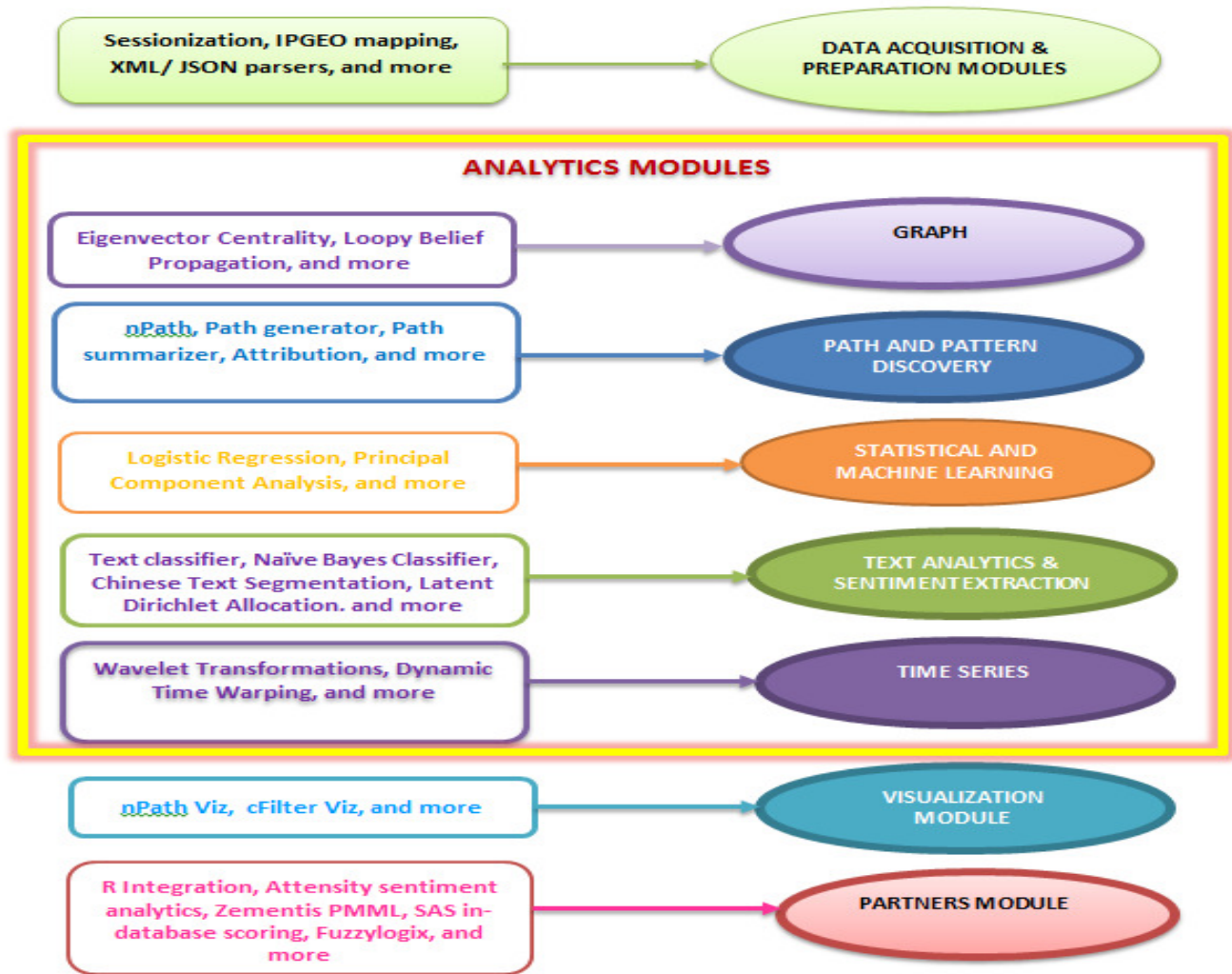


Figure 2: Teradata Aster Discovery Portfolio SQL-MR Function Groups

HIGHLIGHTS

- Massively Parallel Processing (MPP) engine with the Aster Database, MapReduce, SQL-MapReduce and SQL-Graph frameworks for advanced, iterative, and rapid discovery.
- Includes functions for path and pattern analysis, statistics, relational, graph, text analysis, and data transformations.
- Easily incorporated into custom analytics through SQL-MapReduce and SQL-Graph.
- Optimized for high performance and scalability with analytics done on entire data sets without needing to sample.
- 100+ pre-built SQL-Mapreduce and SQL-Graph functions with the ability to create custom ones easily with the Aster Developer Express visual integrated development environment (IDE).

SAMPLE USE CASES

- Analyze Web use, CRM, and other offline data to understand drivers of churn and methods to reduce churn.
- Perform statistical analysis on market prices to create and improve financial portfolio models.
- Explore purchasing patterns and create market baskets that improve cross-selling abilities.
- Understand deviations in sensor data and quickly isolate problems before large manufacturing losses occur.
- Manage brand value, product sentiment, and company perception by analyzing social media data streams.
- Deliver analytics regarding health care treatment plans so that tailored care can be provided to individuals.
- Identify important network nodes that can enable information sharing, identify disease contagion, and key influencers.

Data Preparation and Transformation Functions

Antiselect

Function Name: antiselect

When a user wants to select all but a few columns on an especially large table it is cumbersome to type each column’s name in the select statement. Besides time consuming the risk of errors in selection is enlarged as the number of columns increase. The antiselect function enables the user to specify the set of columns that are NOT needed. This could be useful when simply selecting data or when doing a join to create a new table.

Function Call:

```
SELECT *
FROM antiselect
(ON sampleTable
EXCLUDE('race')
);
```

The input table for this function is:

ID	SOURCE	AGE	GENDER	RACE	NUM BUYS	NUM SELLS
1	ebay	62	male	white	30	44
2	paypal	29	female	asian	33	23

Table 1: Antiselect Input

The function, when invoked, results in:

ID	SOURCE	AGE	GENDER	NUM BUYS	NUM SELLS
1	ebay	62	male	30	44
2	paypal	29	female	33	23

Table 2: Antiselect Output

Sample Use Cases:

- All scenarios where data is extracted from large tables.

Sample Verticals:

- All verticals.

Apache Log Parser

Function Name: apache_log_parser

TERADATA ASTER DISCOVERY PORTFOLIO

Web logs are footprints that capture a visitor's behavior on a Web site as well as information on the Web server's performance that can be analyzed to troubleshoot problems and manage capacity to satisfy traffic demands. These log files conform to the National Center for Supercomputing Applications (NCSA) standard, and the Apache log parser takes each entry and parses out the data into a structured format. The structured data is then used as inputs to the Teradata Aster nPath function that identifies patterns of customer behavior. The table below illustrates an example of a log file and the parser results:

Function Call:

```
SELECT * FROM apache_log_parser (  
  on sample_web_log_file  
  log_column('data')  
  return_search_info('true')  
);
```

The input dataset is below:

```
75.36.209.106 - - [20/May/2008:15:43:57 -0400] "GET / HTTP/1.1" 200 15251  
"http://www.google.com/search?hl=en&q=%22Aster+Data+Systems%22" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT  
5.1; SV1; YPC 3.2.0; .NET CLR 1.1.4322; .NET CLR 2.0.50727; MS-RTC LM 8)"
```

Figure 3: Raw Apache Log Raw Data

This function, when invoked, results in:

DATESTAMP	IP	PAGE	REFERRAL SITE	SEARCH ENGINE	SEARCH TERMS
2008-05-20 15:43:57.0	75.36.209.106	/	<a href="http://www.google.com/search?hl=en&q=%22Aster+Data+Syst
ems%22">http://www.google.com/search ?hl=en&q=%22Aster+Data+Syst ems%22	1	"Aster Data Systems"
2011-03-27 11:45:47.0	98.210.132.218	/about/management.p hp	<a href="http://www.bing.com/search?q
=aster+data&form=QBLH&qsn
&sk=&sc=8-10">http://www.bing.com/search?q =aster+data&form=QBLH&qsn &sk=&sc=8-10	2	aster data

Table 3: Apache Log Parser Output

Sample Use Cases:

- Customer purchasing behavior on Web sites.
- Most and least efficient paths taken by customers towards completing an event (optimizing the customer experience).
- Web site performance tuning.

Sample Verticals:

- E-Commerce
- Retail
- Insurance/risk, finance.

E-Mail Parser

Function Name: email_parser

The predominant mode of communication within and across organizations is e-mail, today. This parser deconstructs large volumes of text data into meaningful buckets for analysis, monitoring, classification and so on. There are several use cases including customer satisfaction analysis, customer churn analysis, customer service analysis, regulatory compliance for ethical conduct, spam filtering, and more.

TERADATA ASTER DISCOVERY PORTFOLIO

Function Call:

```
SELECT *
FROM email_parser(
    ON public.emaildump
    EMAIL_COLUMN('emailcontent')
    CASE_SENSITIVE('true')
    RECORD_INDICATOR('\n')
    DELIMITER(' ')
    ACCUMULATE('emailcontent')
    EMAIL_HEADER('Date:', 'To:', 'From:', 'Subject:', 'Reply-To:', 'Content-Type:')
    EMAIL_FIELD('msg_date', 'msg_to', 'msg_from', 'msg_subject', 'repty_to', 'content_type')
    BODY_INDICATOR('Content-Transfer-Encoding:')
);
```

When this function is invoked on any e-mail data (.pst file) the following is the result:

MESSAGE ID	THREAD ID	TIMESTAMP	SENDER ID	RECIPIENT ID	SUBJECT TEXT	MESSAGE TEXT	
IP	ACTION ID	MAIL LINK	LANGUAGE	MAIL META DATA	ATTACHMENT FORMAT	ATTACHMENT METADATA	ATTACHMENT INFO

Table 4: Email Parser Output

Sample Use Cases:

- Data usage compliance
- Risk monitoring
- Employee churn activity
- Customer satisfaction

Sample Verticals:

- E-Commerce
- Government
- Human Resources Departments

Identity Matching

Function Name: identityMatch

The primary role of the identity matching function is to match large sets of information where there is *noise* in the information such as variant spellings, typos, variations in transliteration, abbreviations, and inconsistent punctuation. In the absence of performing identity matching it is likely that same entities with, say, different spellings, within a block of text will be assigned different identifiers. This is likely going to inflate counts and misrepresent the nature and the context of the information being presented. This function is also used to match customer data with the user data obtained from external data sources. When matching two users, we can choose several attributes and calculate the similarity scores and then using a formula to compute the final similarity score. Attribute matching can be classified into nominal match and fuzzy matching. For example, If two users have same email or mobile, we can say they are the same user; But for location and profile attributes a weighted matching approach is taken. Each organization has its own criteria for what constitutes a good match. While there is no universal “best match”, Aster’s solution of user-selected combinations of fuzzy and nominal match is ideal for increasing data quality.

TERADATA ASTER DISCOVERY PORTFOLIO

Function Call:

```
select * from IdentityMatch
(
  ON customer as a PARTITION BY ANY
  ON handles as b DIMENSION
  idColumn('a.id: b.id')
  nominalMatchColumns('a.email: b.email')
  fuzzyMatchColumns('a.name: b.name,JARO-WINKLER, 2', 'a.location: b.location, JD, 1')
  accumulate('a.name','b.name','a.email','b.email','a.location','b.location')
  threshold(0.4)
) order by "a.id", score desc;
```

The input tables for this function are:

ID	EMAIL	NAME	LOCATION
1	bg@abc.com	Bob Gates	Beijing china
2	js@efg.com	John Smith	San Carlos USA

Table 5: CRM Data

ID	EMAIL	NAME	LOCATION
1	bg@abc.com	gates bob	Beijing China
2		John	USA
3		Goe	Los angle, USA
4		Smith	San Carlos

Table 6: Twitter Data (We can use Aster parser functions on the Twitter data to create a structured dataset.)

This function, when invoked, results in:

ID 1	NAME 1	EMAIL 1	LOCATION 1	ID 2	NAME 2	EMAIL 2	LOCATION 2	MATCHSCORE
1	Bob Gates	bg@abc.com	Beijing china	1	gates bob	bg@abc.com	Beijing China	1.0000
2	John Smith	js@efg.com	San Carlos USA	2	John		USA	0.6978
2	John Smith	js@efg.com	San Carlos USA	4	Smith		San Carlos	0.5111

Table 7: Identity Matching Output

Sample Use Cases:

- Brand monitoring and reputation management.
- Reduced redundancy in mailings due to better matching.
- Customer satisfaction.
- Product/service recommendations.
- Compliance.

Sample Verticals:

- E-Commerce
- National Security Intelligence Agencies

IpGeo Mapping

Function Name: IpGeo

This function maps IP address to location information. It can be used to identify visitors’ geographical location (i.e. country, region, city, latitude, longitude, ZIP code, time zone, connection speed, ISP, and domain name). There are numerous applications for this function including: targeted online advertising, content localization, geographic rights management, online security and fraud prevention.

Function Call:

```

This call assumes that the IP database is stored as a file in Aster Database
SELECT *
FROM IPGEO(
    ON ipgeo_1
    IpAddressColumn('ip')
    Converter('MaxMindLite.jar','com.asterdata.sqlmr.analytics.location.ipgeo.MaxMindLite')
/*In this case the assumption is that the IP database is stored as a file in Aster Database */
    ACCUMULATE('id', 'ip')
)
    
```

The input table for this function is:

ID	IP
1	10.1.1.27
2	153.65.16.10
3	10.0.0.20
4	202.106.0.20
5	202.106
6	ddd

Table 8: IP Data

This function, when invoked, results in:

ID	IP	COUNTRY CODE	COUNTRY NAME	STATE	CITY	POSTAL CODE
1	10.1.1.27					
2	153.65.16.10	us	United States	Ohio	Miamisburg	45342
3	10.0.0.20					
4	202.106.0.20	cn	China	Beijing	Beijing	
5	202.106					
6	ddd					

LATITUDE	LONGITUDE	ISP	ORGANIZATION	ORGANIZATION TYPE	AREA CODE	METRO CODE	DMA CODE
19.6182	-24.688				917		
39.9289	116.3883				0		

Table 9: IPGEO Output

Sample Use Cases:

TERADATA ASTER DISCOVERY PORTFOLIO

- Enabling market research by identifying GEO regions that are more likely to contain attributes that can be used by organizations for specific purposes (e.g., fund raising).
- Segmenting areas by population and competitor stores to determine new store locations.
- Creating advertising campaigns that are specific to a region.
- Obtaining critical information about anti-social activity by region and extending law enforcement resources to regions that are more susceptible to criminal attacks.

Sample Verticals:

- Retail
- Advertising
- Entertainment
- Telecommunications

JSON Parser

Function Name: JSONParser

The JSON parser extracts element name and text from JSON (JavaScript Object Notation) strings. JSON is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to generate and parse. A JSON string is a string of notations in text format. It is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, and Python. JSON is often used in Ajax techniques and used to pass Ajax updates between the server and the client.

Function Call:

```
select * from JSONParser (  
  ON jparser1  
  TEXT_COLUMN('data')  
  NODES('menu/{id,value}','menuitem/value')  
  DELIMITER(' | ')  
  NODEID_OUTPUTCOLUMN_NAME('ID')  
  PARENTNODE_OUTPUTCOLUMN_NAME('ParentName')  
  ACCUMULATE('id'));
```

The JSON parser takes information from a JSON string and extracts relevant subject content, for example:

```
var jsontext = '{"firstname":"Jesper","surname":"Aaberg","phone":["555-0100","555-0120"]}';  
var contact = JSON.parser(jsontext);  
document.write(contact.surname + ", " + contact.firstname); /* JSON String */
```

// JSON Parser Function Output: Aaberg, Jesper

Sample Use Cases:

- Improving the customer's Web search experience by using JSON objects to pass Ajax updates between client and server.
- Extracting text, customer attributes from JSON strings, and using this structured information in combination with other data structures to address churn, customer retention, brand management, and other use cases.

Sample Verticals:

- E-commerce
- Organizations that use Web portals for any service
- Government

Multicase

Function Name: multi_case

This function iterates through the input data set once, and outputs records whenever a match occurs on some specified criteria (e.g., if income>\$100,000 then 'High'). If multiple matches occur for a given input row, one output row will be output for each match.

Function Call:

```
SELECT * FROM multi_case
(ON
(SELECT
*,
age < 1 as case1,
(age >= 1 AND age <=2 ) as case2,
(age >= 2 AND age <=12) as case3,
(age >=13 AND age <=19) as case4,
(age >=16 AND age <=25) as case5,
(age >= 21 AND age <= 40) as case6,
(age >=35 AND age <= 60) as case7,
(age >=60) as case8
FROM mydata
)
LABELS('case as "infant"',
'case2 as "toddler"',
'case3 as "kid"',
'case4 as "teenager"',
'case5 as "young adult"',
'case6 as "adult"',
'case7 as "middle aged person"',
'case8 as "senior citizens"')
);
```

The input table for the function is:

<u>ID</u>	<u>NAME</u>	<u>AGE</u>
100	Henry Cavendish	12
200	Sir William	15
300	Johann August	19
400	Martin Heinrich	20
500	Ralph Arthur	25
600	Marguerite Catherine	35
700	Philip Hauge	40
800	Joseph Louis	28
900	Marie Curie	12

Table 10: Demographic Data

The function, when invoked, results in:

<u>ID</u>	NAME	AGE	CATEGORY
100	Henry Cavendish	12	kid
200	Sir William	15	teenager
300	Johann August	19	teenager
300	Johann August	19	young adult
400	Martin Heinrich	20	young adult
500	Ralph Arthur	25	young adult
500	Ralph Arthur	25	adult
600	Marguerite Catherine	35	adult
600	Marguerite Catherine	35	middle aged
700	Philip Hauge	40	adult
700	Philip Hauge	40	middle aged
800	Joseph Louis	28	Adult
900	Marie Curie	12	kid

Table 11: Multicase Output

Sample Use Cases:

- This data preparation and transformation function can be used for all use cases that require such transformations

Sample Verticals:

- All verticals

Murmurhash

Function Name: murmurhash

MurmurHash function is a tool to compute the hash value of the input columns. It is a non-cryptographic hash function suitable for general hash-based lookup. The function computes the murmurhash value of each cell in the input columns.

Function Call:

```
select * from murmurhash ( on input_table inputcolumns('t_bytea', '[2:6]', "'t_date'") accumulate('id'));
```

The input table for this function is*:

ID	BYTE A	VARCHAR	CHAR	CHAR 20	TIME	TEXT	DATE
----	--------	---------	------	---------	------	------	------

TERADATA ASTER DISCOVERY PORTFOLIO

ID	BYTE A	VARCHAR	CHAR	CHAR 20	TIME	TEXT	DATE
1	a	a	a	11:11:11	11:11:11	2011-11-12	2011-11-12
2	null	null	null	null	null	null	null

Table 12: Murmurhash Input

*This can also be generated by the following code:

```
create fact table input_table(id int, t_bytea bytea, t_varchar varchar(20), t_char char, t_char20 char(20), t_time time, t_text text, t_date date) distribute by hash(id);
```

The function, when invoked, results in:

ID	BYTE A MURMURHASH	VARCHAR MURMURHASH	CHAR MURMURHASH	CHAR 20 MURMURHASH	TIME MURMURHASH	TEXT MURMURHASH	DATE MURMURHASH
1	-1563381124	-1563381124	-1563381124	684435030	698478580	-1894534554	-1894534554
2	null	null	null	null	null	null	null

Table 13: Murmurhash Output

Sample Use Cases:

- This data preparation and transformation function can be used for all use cases that require such transformations

Sample Verticals:

- All verticals

Outlier Filter

Function Name: outlierFilter

This function isolates all outliers (based on user defined thresholds) from a given data pattern.

Function Call:

```
SELECT *
FROM outlierFilter(
  ON (SELECT 1)
  PARTITION BY 1
  inputTable('input_table')
  outputTable('output_table')
  filterCols('value')
  method('percentile')
  percentileValues('1','90')
  removeTail('both')
  replacementValue('null')
  groupByCols('groupcol', 'Variable_Name')
  userid('beehive')
  password('beehive')
);
```

The input table for this function is:

TERADATA ASTER DISCOVERY PORTFOLIO

ID	GROUP	VARIABLE	VALUE
1	A	Pressure	500
2	A	Pressure	99999
3	A	Temperature	23
4	A	Temperature	80
5	A	Temperature	100
6	A	Temperature	90
7	A	Pressure	-10

Table 14: Outlier Filter Input

The function, when invoked, results in:

ID	GROUP	VARIABLE	VALUE
1	A	Pressure	500
2	A	Pressure	NULL
3	A	Temperature	23
4	A	Temperature	80
5	A	Temperature	100
6	A	Temperature	90
7	A	Pressure	NULL

Table 15: Outlier Filter Output

In the above output, outliers in RowIDs 2 and 7 have been removed.

Sample Use Cases:

- A principal use case is in fraud detection. When money transfers or credit card purchases fall within a certain pattern these transactions can be automatically detected and flagged or declined.

Sample Verticals:

- Banking
- Pharmaceutical Research
- Hi-tech Manufacturing

Pack and Unpack

Function Names: pack, unpack

The pack function transforms the input table columns and merges all columns into a row. The row's contents are separated by a column delimiter string (e.g., comma). Packing columns frees up disk space, and speeds query results retrieval.

TERADATA ASTER DISCOVERY PORTFOLIO

The unpack function expands data from a single packed column into multiple columns. The COLUMN_DELIMITER string is the basis for isolating the individual columns. As opposed to pack, unpack is used at the time of analysis and can focus on a subset of data and make the analytical process more efficient.

Function Call: pack

```
SELECT *
FROM pack
(
  ON to_be_packed
  COLUMN_DELIMITER(',')
  PACKED_COLUMN_NAME('packed_data')
  INCLUDE_COLUMN_NAME('true')
  COLUMN_NAMES('src', 'age', 'gender', 'race', 'numBuys', 'numSells')
);
```

The input table is:

ID	SOURCE	AGE	GENDER	RACE	NUM BUYS	NUM SELLS
1	ebay	62	male	white	30	44
2	paypal	29	female	asian	33	23

Table 16: Pack Input

The function, when invoked, results in:

ID	PACKED DATA
1	src:ebay,age:62,gender:male,race:white,numBuys:30,numSells:44
2	src:paypal,age:29,gender:female,race:asian,numBuys:33,numSells:23

Table 17: Pack Output

Function Call: Unpack

```
SELECT *
FROM unpack
(
  ON unpack_data
  DATA_COLUMN('packed_data')
  COLUMN_NAMES('age', 'gender', 'race', 'numBuys', 'numSells')
  COLUMN_TYPES('integer', 'varchar', 'varchar', 'integer', 'integer')
  COLUMN_DELIMITER(',')
  DATA_PATTERN('(.*)')
  DATA_GROUP(1)
  IGNORE_BAD_ROWS('true')
);
```

The input table is:

ID	SOURCE	PACKED DATA
1	ebay	62,male,white,30,44

ID	SOURCE	PACKED DATA
2	paypal	29,female,asian,33,23
3	Bad_data	THISISINVALIDDATA

Table 18: Unpack Input

The function, when invoked, results in:

AGE	GENDER	RACE	NUM BUYS	NUM SELLS	ID	SOURCE
62	male	white	30	44	1	ebay
29	female	asian	33	23	2	paypal

Table 19: Unpack Output

Sample Use Cases:

- These data preparation and transformation functions can be used for all use cases that require such transformations

Sample Verticals:

- All verticals

Pivot and Unpivot

Function Name: pivot, unpivot

The **pivot** function converts rows of data into columns. This function enables users to partition data in different ways.

The **unpivot** function converts columns of data into rows. This is a reverse of pivot, and enables the user to see the data in its original representation.

Function Call for pivot:

```
SELECT * FROM pivot(
  ON sample_table
  PARTITION BY member_id
  ORDER BY week
  PARTITIONS('week')
  ROWS(3)
  METRICS('value')
);
```

The input table is:

ID	WEEK	VALUE
2	1	202
2	3	205
1	1	100

Table 20: Pivot Input

TERADATA ASTER DISCOVERY PORTFOLIO

The function, when invoked, results in:

ID	VALUE 0	VALUE 1	VALUE 2
1	100	103	107
2	202	205	

Table 21: Pivot Output

Function Call: `unpivot`

```
SELECT *
FROM Unpivot(
  ON inputDataToUnpivot
  colsToUnpivot('sens1','sens2','sens3')
  colsToAccumulate('id1','id2')
)
ORDER BY id2;
```

The input table is:

ID1	ID2	SENS1	SENS2	SENS3
1	1	74	5697	17.3705606244
5	2	19	9581	15.6246871106
5	3	78	8139	14.3446582505
8	4	43	6830	10.8187293343

Table 22: Unpivot Input

The function, when invoked, results in:

ID1	ID2	ATTRIBUTE	VALUE
1	1	SENS1	74
1	1	SENS2	5697
1	1	SENS3	17.37056
5	2	SENS1	19
5	2	SENS2	9581
5	2	SENS3	15.624687
5	3	SENS1	78
5	3	SENS2	8139

ID1	ID2	ATTRIBUTE	VALUE
5	3	SENS3	14.344658
8	4	SENS1	43
8	4	SENS2	6830
8	4	SENS3	10.818729

Table 23: Unpivot Output

Sample Use Cases:

- These data preparation and transformation functions can be used for all use cases that require such transformations

Sample Verticals:

- All verticals

Sampling

Function Name: sample

This function makes a selection of a subset of individuals from within a population to estimate characteristics of the whole population. Some of the techniques used are stratified sampling, random whole data set sampling, cluster sampling, panel sampling, and quota sampling. When a representative dataset is selected by way of any sampling technique, then proper statistical analytics techniques can be applied to determine relationships among variables within that dataset which then can be used to infer those relationships in the larger population. Sampling is easier than analyzing the entire population due to the latter’s size as well as the costs incurred in analyzing large datasets.

Function Calls:

Creating and preparing a dataset to sample from:*

```
create fact table sample_input(id int not null,first_name text,last_name text,country
text,score int,sex varchar,weight decimal(28,5),birth timestamp with time zone,combined double,birth_tms timestamp w
ithout time zone) distribute by hash(id);
```

```
create view view_sample_input_stratified as select *, case when score <= 80 then 'fair' when score >80 and score
<90 then 'very good' when score >=90 then 'excellent' end as stratum from sample_input;
```

*If there is already a dataset that could be used as an input to the sampling function then ignore this step

Applying the sampling function on the dataset created in the previous step:

```
select * from sample( on view_sample_input_stratified as data
partition by any on (select stratum, count(*) as stratum_count from view_sample_input_stratified group by stratum) as s
ummary dimension conditionOnColumn('stratum') conditionOn('fair') approximateSampleSize('10')
seed('200')) order by id;
```

Sample Use Cases:

- When developing new medicines to combat specific diseases, companies routinely take many samples from the population of individuals it is concerned with to initiate clinical trials. The objective behind these various sample extractions is to discover insights that could be inferred to the population.

TERADATA ASTER DISCOVERY PORTFOLIO

- Polling organizations routinely use sampling methodology to infer Election Day voting trends.

Sample Verticals:

- Pharmaceutical research
- Market research

Sessionization

Function Name: Sessionize

In the context of opening up an Internet browser, a session is defined as comprising a series of browsing activities within a certain period of time. For example, an online visitor could start with the landing page, then move to the products page, then click on the legal documentation page, and then click on the check-out cart, and then finally make a purchase. The site administrator determines the time window to mark a session (e.g., 30 minutes). Sessionization is the process of mapping each click in a clickstream to a unique session identifier. A user can be associated with many session identifiers because they may have browsed a Web site more than once in a day or week or month. Stringing together all the various visitor sessions creates a composite picture of the types of a given visitor's activities. Sessionization is the most important function that is implemented (though not always as it is dependent on the data set) before the Teradata Aster nPath function is implemented.

Function Call:

```
SELECT *
FROM SESSIONIZE
(
  ON sessionize_table
  PARTITION BY partition_id
  ORDER BY clicktime
  TIMECOLUMN('clicktime')
  TIMEOUT('60')
  RAPIDFIRE('0.2')
) ORDER BY partition_id, clicktime;
```

The input table for this function is:

ID	CLICK TIME	USER ID	PRODUCT	PAGE TYPE	REFERRER	PRICE
1	1110000	333		home	www.yahoo.com	
1	1112000	333	ipod	checkout	www.yahoo.com	200.2
1	1160000	333	bose	checkout		340
1	1200000	333		home	www.google.com	
1	1203000	67403		home	www.google.com	
1	1300000	67403		home	www.google.com	
1	1301000	67403		home		
1	1302000	67403		home		
1	1340000	67403	iphone	checkout		650

TERADATA ASTER DISCOVERY PORTFOLIO

ID	CLICK TIME	USER ID	PRODUCT	PAGE TYPE	REFERRER	PRICE
1	1450000	67403	bose	checkout		750
1	1450200	80000		home	www.godaddy.com	
1	1450600	80000	bose	checkout		340
1	1450800	80000	itrip	checkout		450
1	1452000	80000	iphone	checkout		650

Table 24: Sessionization Input

The function, when invoked, results in:

ID	CLICK TIME	USER ID	PRODUCT	PAGE TYPE	REFERRER	PRICE	SESSION ID	RAPIDFIRE
1	1110000	333		home	www.yahoo.com		0	f
1	1112000	333	ipod	checkout	www.yahoo.com	200.2	0	f
1	1160000	333	bose	checkout		340	0	f
1	1200000	333		home	www.google.com		0	f
1	1203000	67403		home	www.google.com		0	f
1	1300000	67403		home	www.google.com		1	f
1	1301000	67403		home			1	f
1	1302000	67403		home			1	f
1	1340000	67403	iphone	checkout		650	1	f
1	1450000	67403	bose	checkout		750	2	f
1	1450200	80000		home	www.godaddy.com		2	t
1	1450600	80000	bose	checkout		340	2	f
1	1450800	80000	itrip	checkout		450	2	t
1	1452000	80000	iphone	checkout		650	2	f

Table 25: Sessionization Output

The above output table provides a session ID and each session ID strings together a set of actions on a given site visit.

Sample Use Cases:

- Identifying common traversal paths taken by the customer to complete an event (e.g., purchase cancellation).
- Examining isolated instances of risky behavior and stitching together all instances to assess the likelihood of risk for specific purposes (e.g., insurance, recidivism).

TERADATA ASTER DISCOVERY PORTFOLIO

Sample Verticals:

- Government (cyber fraud agencies)
- Pharmaceutical
- E-commerce
- Social services (drug abuse, crime propensity)

XML Parser

Function Name: XMLParser

XML Parser extracts data (e.g., price, item, shop name) from XML documents and exports them into columns. This enables the inclusion of semi-structured data into the analytics environment.

Function Call:

```
SELECT * FROM XMLParser(  
  ON xml_inputs  
  TEXT_COLUMN('xmlDocument')  
  NODES ('price/member')  
  SIBLING ('author')  
  SIBLING_DELIMITER(';')  
  ACCUMULATE('Xid')  
)
```

The input table for this function is:

ID	XML DOCUMENT
1	<bookstore> <owner> "billy" </owner> <book category="WEB"> <title lang="en">XQuery Kick Start</title> <author>James McGovern</author> <author>Per Bothner</author> <year edtion="1">2003</year> <year edtion="2">2005</year> <price> <member>49.99</member> <public>60.00</public> </price> <reference> <title>A</title> </reference> <position value="1" locate="east"/> </book> <book category="CHILDREN"> <author>Wenny Wang</author> <price> <member>99.99</member> <public>108.00</public> </price> </book> </bookstore>

Table 26: XML Input

This function, when invoked, results in:

ID	NODE ID	PARENT NODE	AUTHOR	MEMBER
1	1	price	James McGovern; Per Bothner	49.99

Table 27: XML Output

Sample Use Cases:

- This data preparation and transformation function can be used for all use cases where XML data is available.

Sample Verticals:

- All verticals

Graph Functions

All Pairs Shortest Path

Function Name: AllPairsShortestPath

The All Pairs Shortest Path function measures the shortest distance between two or more vertices (nodes) on a graph. A variant of this is the Single Source Shortest Path (SSSP) function that finds the shortest paths from a given vertex to all the other graph vertices.

This function is analogous to the problem of finding the shortest path between two intersections on a road map: the graph's vertices correspond to intersections and the edges correspond to road segments, each weighted by the length of its road segment. The aim of this function is to determine what the shortest cumulative distance is between two nodes. The figure below shows an undirected (meaning, one can traverse in both directions) graph where two paths between vertices A and F can be traversed. However, the shortest path is to traverse the vertices A-B-C-F whereas the longer path is A-B-D-E-F.

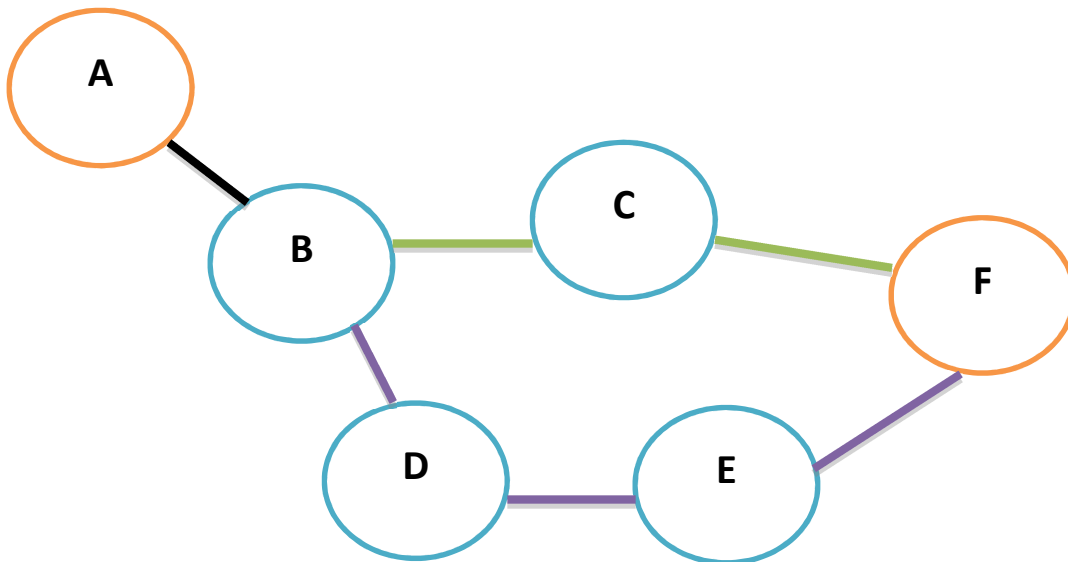


Figure 4: Two paths from Vertices (Nodes) A to F in an unweighted and undirected graph: A-B-D-E-F (longest) and A-B-C-F (shortest).

In the figure below where there are weights associated with each edge (link) between the vertices (nodes), the shortest distance is computed by identifying the path that is least weighted. Now, even though, the non-weighted shortest path is A-B-C-F, the shortest path when weights are assigned is A-B-D-E-F as the total weight is the lowest with this path. In other words, it helps, in this example, to think of weights as the cost of path traversal. If the objective is to take the shortest path regardless of cost (unweighted) then A-B-C-F is the best choice. If costs are considered then A-B-D-E-F becomes the shortest path.

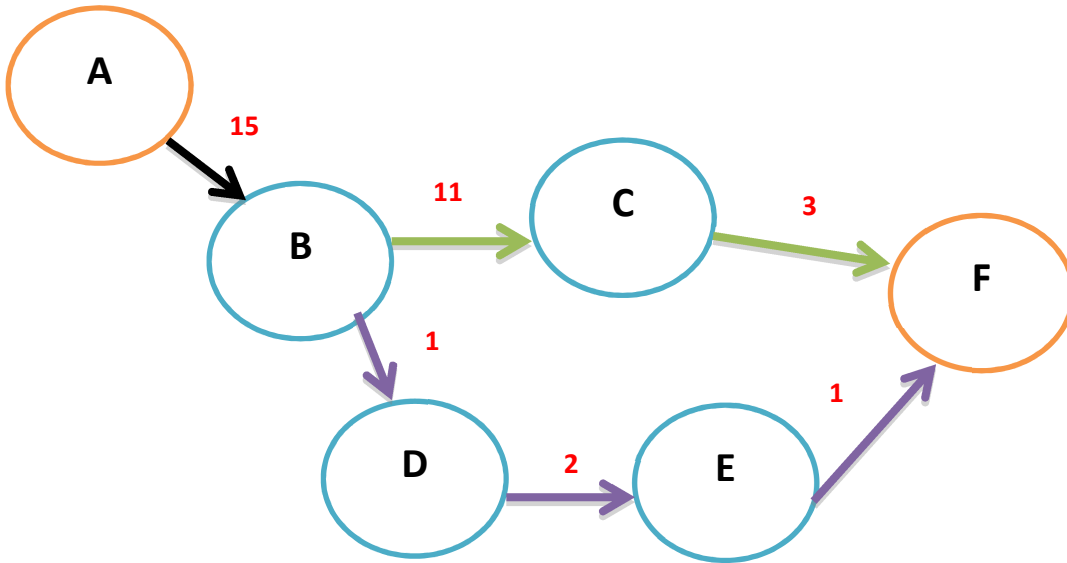


Figure 5: Two paths from Vertices (Nodes) A to F in a weighted and directed graph: A-B-D-E-F (shortest) and A-B-C-F (longest).

The following is an illustration of the steps taken to implement a graph function on a network of phone callers:

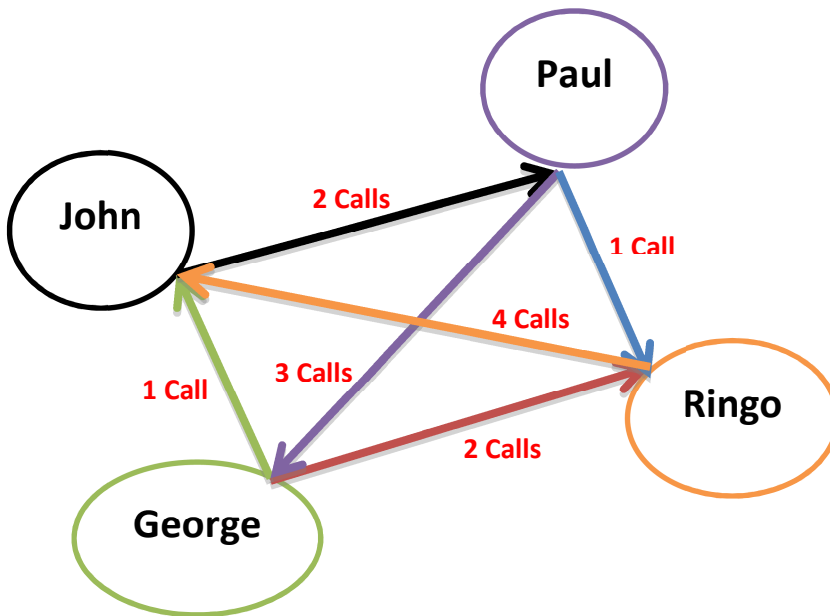


Figure 6: Calling graph

Step 1: Visualize the calling graph (see above).

Step 2: Deconstruct the call graph into two tables.

CALLER ID	CALLER	CALL FROM	CALL TO	# CALLS
-----------	--------	-----------	---------	---------

TERADATA ASTER DISCOVERY PORTFOLIO

1	John	1	2	2
2	Paul	2	3	3
3	George	2	4	1
4	Ringo	3	4	2
		3	1	1
		4	1	4

Tables 28 and 29: Callers and Calls

Function Call 1: Without using edge weights

```
SELECT *
FROM AllPairsShortestPath ( ON "Callers" PARTITION BY "callerId" AS "vertices"
ON "Calls" PARTITION BY "callerFrom" AS "edges"
EDGE_TARGET('callerTo'))
```

Function Call 2: Using edge weights

```
SELECT *
FROM AllPairsShortestPath ( ON "Callers" PARTITION BY "callerId" AS "vertices"
ON "Calls" PARTITION BY "callerFrom" AS "edges"
ON (SELECT "callerId" FROM "Callers" WHERE "CallerId" IN (1,2)) PARTITION BY "CallerId")
EDGE_TARGET('callerTo')
EDGE_WEIGHT ('calls')
```

These functions calls result in the following distance computations:

CALL FROM	CALL TO	DISTANCE
1	2	1
1	3	2
1	4	2
2	1	2
2	3	1
2	4	1
3	1	1
3	2	3
3	4	1
4	1	1
4	2	2

4	3	3
---	---	---

Table 30: Shortest Distance without weights

SOURCE	TARGET	DISTANCE
1	2	2
1	3	5
1	4	3
2	1	4
2	3	3
2	4	1
3	1	1
3	2	3
3	4	2
4	1	4
4	2	6
4	3	9

Table 31: Shortest Distance with weights

Sample Use Cases:

- Shortest path algorithms are applied to automatically find directions between physical locations. Driving and walking directions websites like Mapquest or Google Maps are exemplars of this implementation.
- E-commerce providers can apply graph functions find an optimal sequence of choices to reach a certain goal or end state.
- Gaming solutions could use graph technology to identify possible paths towards a goal and each path could be scored based on the time it takes to traverse it and points can be allotted to participants based on performance.
- In a networking or telecommunications mindset, this shortest path problem is sometimes called the min-delay path problem and usually tied with a boundary condition (e.g., bandwidth). For example, the algorithm may be used to determine the shortest (min-delay) path with the most bandwidth, or choose among several paths that have a high bandwidth but with the shortest delay.
- Determining optimal transportation networks in urban planning.
- Understanding how to design office and residential spaces in a vast layout.

Sample Verticals:

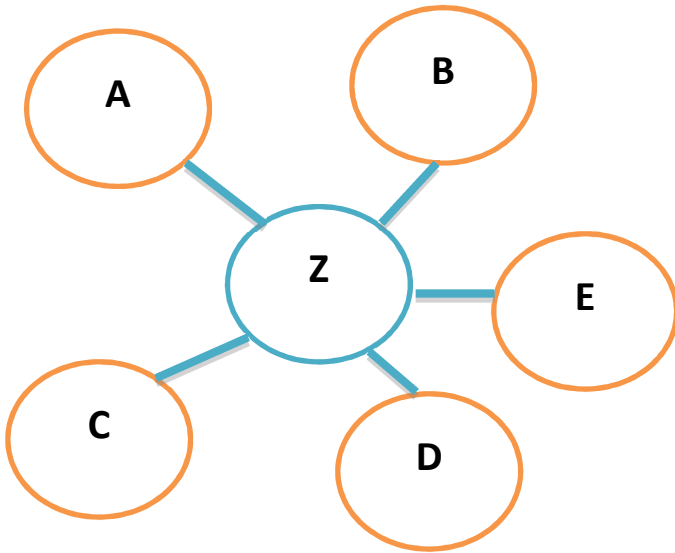
- Manufacturing
- Transportation
- Telecommunication
- Online Gaming

Betweenness

Function Name: betweenness

TERADATA ASTER DISCOVERY PORTFOLIO

In Graph theory, centrality is the notion of the importance of a node in a network. Betweenness is one measure of centrality. The Betweenness centrality function determines the importance of a given node by measuring the distance of all the paths that passes through it. In the figure below node Z has the highest betweenness score as it is the only node that connects every other pair of



nodes. For example nodes AD, BC, CE etc. can connect only through node Z.

Figure 7: Betweenness computation for Node Z

In the example above, it is easy to intuitively know that Z is the most important node (the highest betweenness score) in the network. But, in most networks (e.g., social) interactions between nodes are far more complex. Betweenness of a node is computed by adding up the number of times it appears among the shortest path that is taken to traverse between two other nodes. In the above figure, the betweenness value of node Z, the node that is present in all the shortest paths taken between any 2 nodes, is 10. Higher betweenness scores indicate higher importance.

Function Call :

```
SELECT * FROM  
Betweenness ( ON vertices AS "vertices" partition by vertexid  
              ON edges AS "edges" partition by source  
              TARGET_KEY ('target')  
              ACCUMULATE ('source')      );
```

The tables for the function call are derived from the following network:

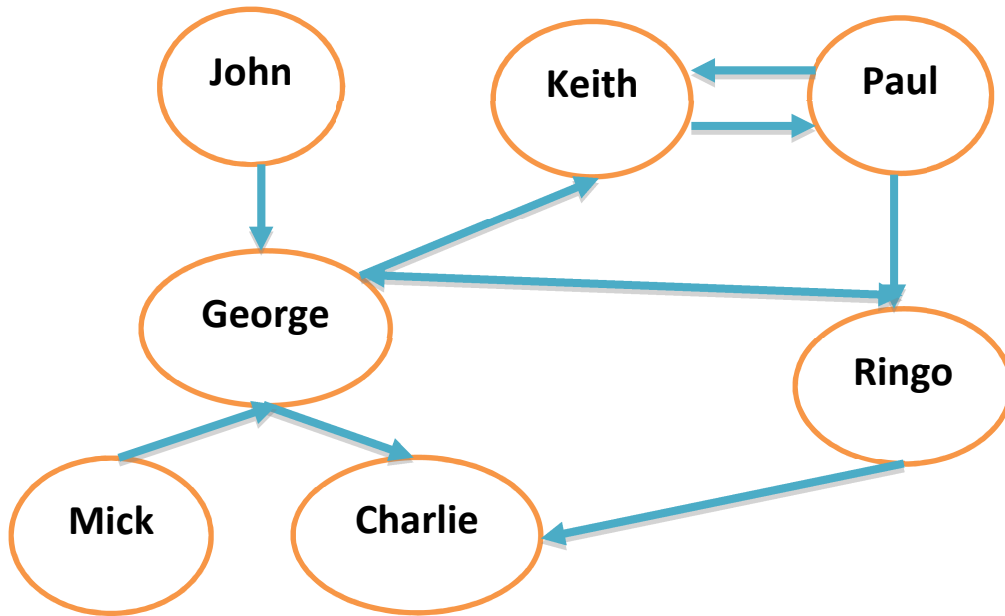


Figure 8: Betweenness Network

ID	SOURCE	TARGET
John	John	George
Paul	Paul	Ringo
George	George	Charlie
Ringo	George	Keith
Mick	Ringo	George
Keith	George	Ringo
Charlie	Mick	George
	Keith	Paul
	Paul	Keith
	Ringo	Charlie

Tables 32 and 33: Nodes and Edges

The function, when invoked, results in:

SOURCE	BETWEENNESS SCORE
John	0.124

SOURCE	BETWEENNESS SCORE
Paul	0.222
George	0.238
Ringo	0.345
Mick	0.412
Keith	0.657
Charlie	0.890

Table 34: Betweenness Score Output

Sample Use Cases:

- Influencer behavior in a network of connections and across departments can be isolated and important drivers of influence can be identified to advance specific actions. For example, when new products or practices are developed, key organizational stakeholders who may also be influential can be recruited to be crucial advocates for information dissemination.
- Transportation decisions that connect large urban centers with small satellite towns can be made such that the shortest commute times and distances are made possible for commuters.

Sample Verticals:

- Social media
- Transportation
- Logistics

Closeness

Function Name: closeness

Closeness, like betweenness, is another measure of centrality (the importance of a given vertex or node in a graph). In layman's terms, closeness measures how quickly a vertex (node) can reach all the other vertices (nodes) in a defined network. Closeness can be seen in the context of information delivery. Node A has a higher closeness index if it is able to transmit information quicker than Node B to all the other nodes in the network in which A and B are a part of. In mathematical terms, closeness is the inverse value of the sum of the total distance from a given node to other nodes in a connected network. Let's take the following graph as an example:

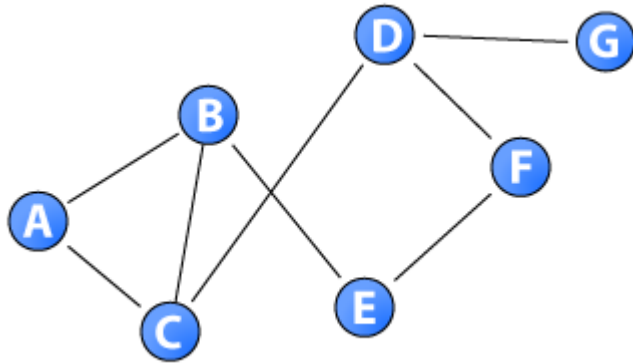


Figure 9: Illustrating Closeness in a connected network

In the above graph, if the distance between each node is 1, the total distance of A to all other nodes is 12. Similarly, for B, the total distance is 10. A’s closeness value is 1/12 (the inverse of the total distance) which is .08, whereas B’s closeness is .12. In other words, the lower the closeness value, the easier it is to disseminate information from a given node to all other nodes in the graph.

Function Call : closeness

```
SELECT *
FROM Closeness ( ON "Callers" as "vertices" PARTITION BY "callerid"
ON "Calls" as "edges" PARTITION BY "callerfrom"
TargetKey('callerto')
MaxDistance ('-1')
Accumulate ('callerid'))
```

The following graph network is used as an input to the function call above:

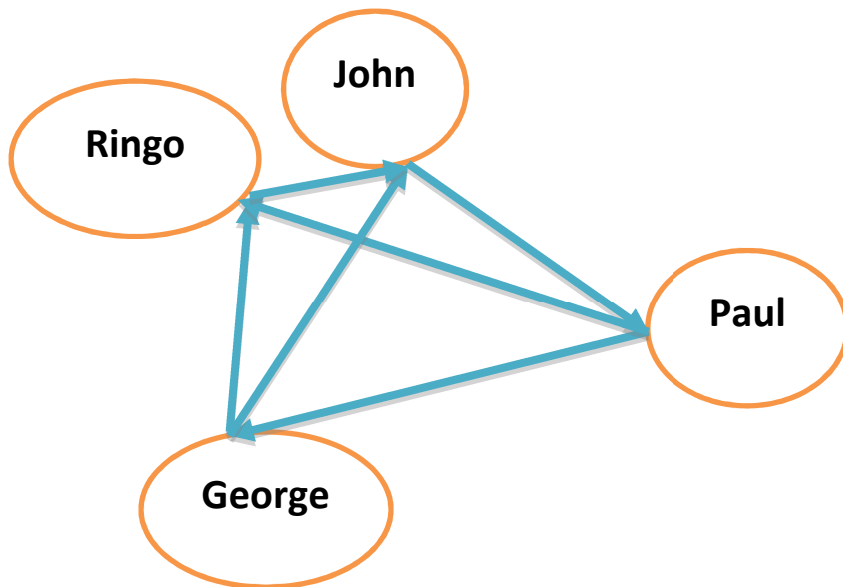


Figure 10: Caller Graph

TERADATA ASTER DISCOVERY PORTFOLIO

CALLER ID	CALLER	CALL FROM	CALL TO	# CALLS
1	John	1	2	2
2	Paul	2	3	3
3	George	2	4	1
4	Ringo	3	4	2
		3	1	1
		4	1	4

Tables 35 and 36: Callers and Call Details

The function, when invoked, results in:

CALLER ID	CLOSENESS SCORE
1	.2
2	.25
3	.25
4	.17 (CLOSEST)

Table 37: Closeness Score Output

Sample Use Cases:

- When it comes to preventing deadly communicable diseases from spreading it helps to know which carriers are likely to spread the virus the quickest to a large population. Understanding interactions among people in a network enables care providers to focus on sections of the population deemed most likely to spread a disease and take immediate action.
- Information dissemination across a vast network of recipients is sped up by targeting key groups within that network who have the greatest ability to reach the most number of people in a short time.

Sample Verticals:

- Healthcare
- Mass Media

Eigenvector Centrality

Function Name: eigenvectorCentrality

Eigenvector centrality is the notion that is described by the saying “important people know other important people”. Eigenvector centrality measures the importance of a node based on its proximity to other nodes that are also deemed important. Google’s PageRank algorithm heavily borrows from the eigenvector centrality concept by ranking pages that are referred to or directed from other highly trafficked pages. Eigenvector centrality takes the entire pattern of the network into consideration, and determines the importance of a node based on its relationship to other nodes that have a high degree of centrality in the network.

TERADATA ASTER DISCOVERY PORTFOLIO

In other words, each node is weighted based on their connection in the network and this weight is based on the number and quality of connections of that node. An interesting effect of this centrality measure is that nodes with fewer connections but high value connections are deemed to be more important than nodes with a greater number of low or medium value connections.

The example below highlights the essence of eigenvector centrality. Node D has a degree of three — that is, it is connected to only three other nodes. Node A, on the other hand, has a degree of four. Node A is more popular (important) in the network if popularity (importance) simply sums up each connection. Eigen vector centrality is more than the summation of all connections. D is connected to nodes that are connected to many other nodes, while A is connected to less-popular nodes. D, therefore, has a higher eigenvector centrality.

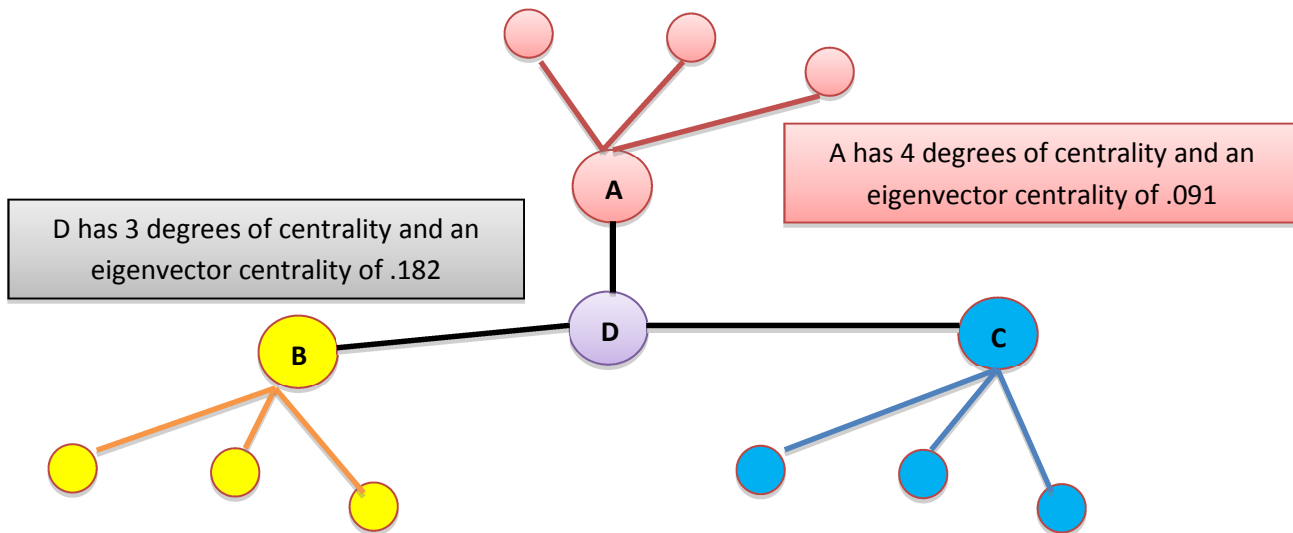


Figure 11: Eigenvector Centrality Graph

Function Call :

```
select * from eigenvectorCentrality(  
  on nodes as vertices partition by id  
  on edges as edges partition by startid  
  TargetKey('endid')  
  Accumulate('id')  
  EdgeWeight('val')  
) order by centrality desc;
```

The following graph network is used as an input to the function call above:

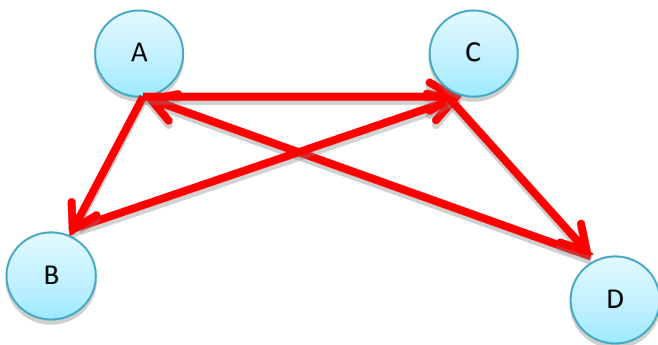


Figure 12: Eigenvector Centrality Input

TERADATA ASTER DISCOVERY PORTFOLIO

The corresponding vertices (nodes) and edges table for the above graph are:

VERTEX (EDGE)	STARTID (FROM)	END ID (TO)	VALUE (EDGE)
A	A	B	1
C	A	C	1
B	B	C	1
D	C	D	1
	D	A	1

Table 38: Eigenvector Centrality Input

When invoked, this function results in the following output:

VERTEX (EDGE)	EIGENVECTOR CENTRALITY
A	.44
B	.38
C	.63 (HIGHEST CENTRALITY NODE)
D	.51

Table 39: Eigenvector Centrality Output

Sample Use Cases:

- In mapping brain connectivity across thousands of neurons eigenvector centrality has been applied to understand how brain functions are changed based on various human states (e.g., hunger, satiation, pain). Once connection patterns are understood then specific effects of various external stimuli can be understood on the brain. This leads to the delivery of more targeted therapies, treatment plans, and medicines.
- Eigenvector centrality measures have been extensively used in studying influencer behaviors in networks of human interaction and often predictions of specific actions or outcomes such as churn or purchase are made.
- Capital flows across tightly coupled market groups and geographies are regulated by issues of network centrality. Capital flows easily from one market to another based on which markets are influential enough in attracting the highest inflows as well as providing the greatest return. The eigenvector centrality measure can be used to determine drivers of influence and predict inflow volumes into specific markets.

Sample Verticals:

- Scientific Research
- E-Commerce
- Capital Markets

K-Degree Centrality

Function Name: kdegree

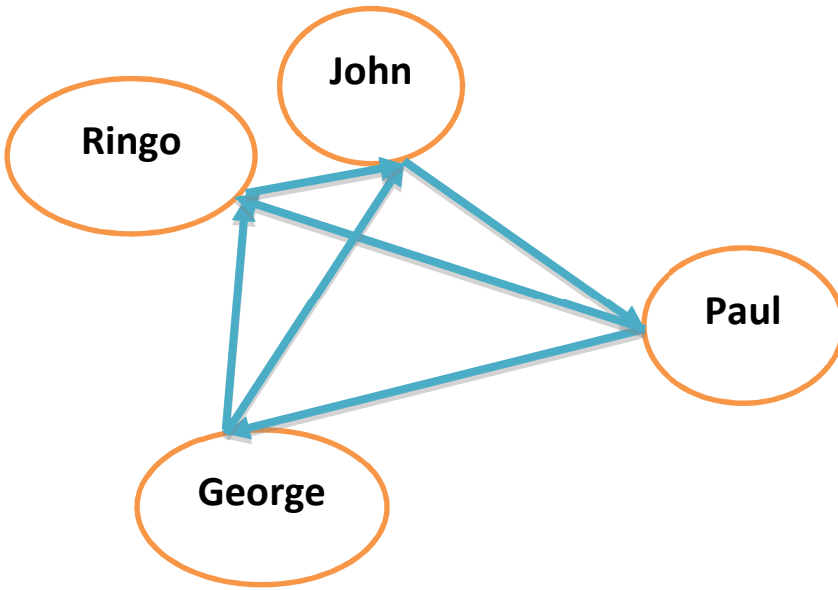
kDegree is one of the fundamental centrality metrics used in network structure analysis. It provides a list and count of all the nodes that are k steps away from a given node, say, A. The K-Degree centrality construct is used to address the question of which node(s) is (are) likely to be the best candidate(s) to be influential without having to traverse the entire network. In other words, such a node represents an entity that, with the smallest number of intermediaries on average, can influence the greatest number of other entities.

Function Call :

```
SELECT *
FROM kDegree ( ON "Callers" PARTITION BY "callerId" AS "vertices"
ON "Calls" PARTITION BY "callerFrom" AS "edges"
EDGE_TARGET('callerTo')
K('2') )
```

The following graph network is used as an input to the function call above:

Figure 13: KDegree Function Input Graph



CALLER ID	CALLER	CALL FROM	CALL TO
1	John	1	2
2	Paul	2	3
3	George	2	4
4	Ringo	3	4
		3	1
		4	1

Tables 40 and 41: Caller IDs and Caller Graphs

When invoked, this function results in the following output:

CALLER ID	CALLER	CENTRALITY COUNT
4	Ringo	2
3	George	2

TERADATA ASTER DISCOVERY PORTFOLIO

CALLER ID	CALLER	CENTRALITY COUNT
2	Paul	3
1	John	3

Nodes 1 and 2 have the highest k-degree centrality by virtue of the fact that they connect to the most other nodes within the specified number of 2 edges (or hops).

Tables 42: K-Degree centrality output

Sample Use Cases:

- In sociopolitical science and health care, researchers strive to understand opinion formation and disease propagation with a focus on the most influential and central people.
- In the case of viral marketing, it is crucial to find a small number of people who can trigger the largest and fastest product adoption through social contact advertising.
- Power management optimization in smart grids is accomplished by looking at localized nodes through which power is supplied without having to get power from a central location. Doing so reduces latencies as well as avoiding a single choke point for power access.

Sample Verticals:

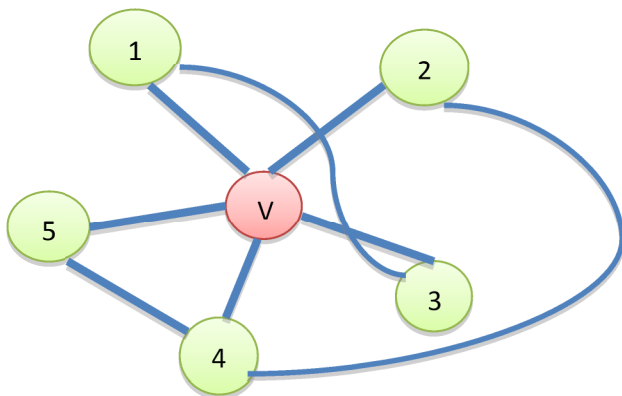
- Utilities
- National Security
- Marketing
- Healthcare Research

Local Clustering Coefficient

Function Name: localclusteringcoefficient

Clustering coefficient is a measure of the degree to which nodes in a graph tend to cluster together. It is the measure of the extent to which one's friends are also friends of each other. Two versions of this measure exist: the global and the local. The global version was designed to give an overall indication of the clustering in the network, whereas the local gives an indication of the embeddedness of single nodes. Embeddedness, in this context, refers to the probability that two randomly selected neighbors are connected to each other.

The local clustering coefficient is measured as the ratio between the number of pairs of neighbors connected to each other and



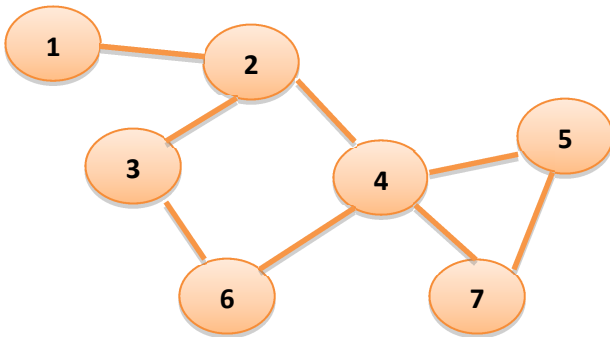
the total number of pairs of relationships. This can be illustrated with the following graph:

Figure 14: Local Coefficient Cluster Graph

In the above graph 3 of v’s neighbors are connected to each other (1-3, 2-4, and 4-5). There could be a total of 10 directly connected pairs in total (1-2, 1-3, 1-4, 1-5, 2-3, 2-4, 2-5, 3-4, 3-5, 4-5). The local clustering coefficient for v is 3/10 or .3. If more pairs are actually connected, the numerator increases and therefore the node “v” will have a larger coefficient which would mean that it would be more embedded as part of a local clique.

Function Call :

```
SELECT * FROM LocalClusteringCoefficient
( ON Friendship as "edges" PARTITION BY fromId
  ON Person as vertices PARTITION BY PersonId
  targetKey('toId')
  directed('f')
  accumulate('personId')
) order by personId;
```



The following friendship connections graph is used as an input to the function call above:

Figure 15: Local Cluster Coefficient Function Input Graph

FROM	TO
1	2
2	3
2	4
3	6
4	5
4	6
4	7
5	7

Table 43: Friendship Graph

When invoked, this function results in the following output:

ID	CLUSTER COEFFICIENT
1	0
2	0
3	0
4	0.1667
5	1
6	0
7	1

Node 4's neighbors are the most connected to each other as compared to all other nodes.

Table 44: Local Cluster Coefficient

Sample Use Cases:

- Isolating local cliques in large participant networks to sell products is more efficient than having to individually approach each member of the network for a separate sales pitch. Members of a clique are likely to behave in lock step with each other and therefore identifying nodes with the largest cluster coefficient helps shrink sales cycles.

Sample Verticals:

- Direct Sales
- Community Relations

Loopy Belief Propagation

Function Name: pmrf (alternate function name is Pairwise Markov Random Field)

In social networks every node has a certain set of attributes and behaviors that can be observed. These attributes could include activities engaged in, gender, geographic location, transaction conducted, and so on. The actions across nodes are observed and certain patterns in behavior can be observed. When new nodes come in there is no information a priori about the kinds of behaviors that are going to be likely engaged in by the new entrants into the network. Loopy Belief Propagation is the algorithm that looks at the associations that are made by the new node in the network and based on the known behaviors and properties of the nodes to which the new node is associated with, certain conclusions about the new node could be made. For example, let's examine the following network graph:

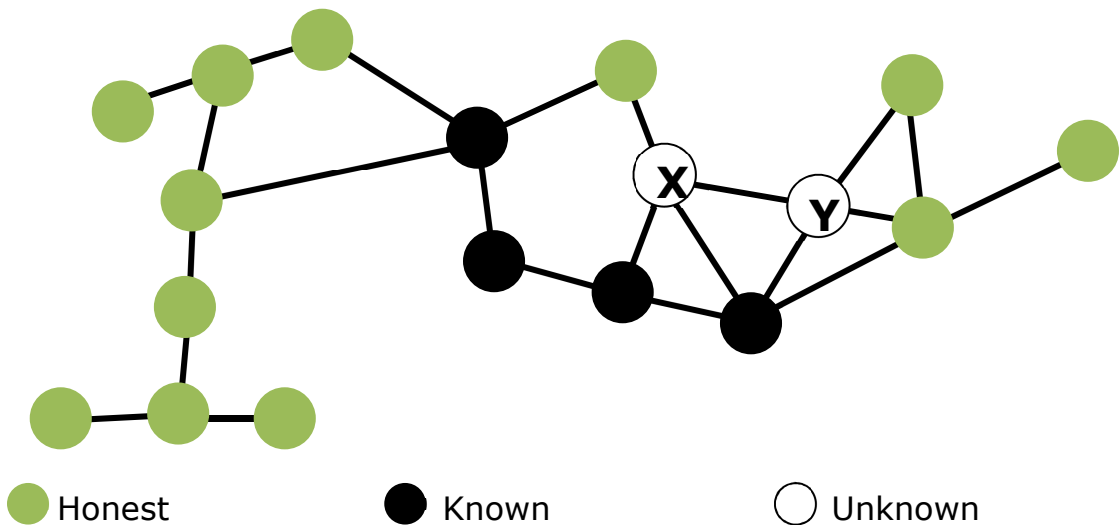


Figure 16: Loopy Belief Propagation

We can see that there is a mix of fraudulent and honest individuals in the network. There are also two new entrants about whom we do not know a lot. Using their associations with both known fraudsters and other honest individuals we can predict the likelihood of both X and Y being fraudsters. Over time, as nodes start passing information to other nodes across the network a certain belief gets propagated and over time that belief does not change. In this case, as X and Y start interacting in the network, other nodes collect information from X and Y and provide some information which could be used to assess the probability that X and Y are fraudsters. This is the underlying concept of Loopy Belief Propagation.

Loopy Belief Propagation provides an approximate (but often good enough) solution. It is an iterative process in which neighboring variables “talk” to each other, passing messages such as: “I (variable x3) think that you (variable x2) belong in these states with various likelihoods...” After a few iterations, this series of conversations is likely to converge to a consensus that determines the marginal probabilities of all the variables. Estimated marginal probabilities are called beliefs.

Function Call :

```
SELECT * FROM pmrf (
  ON edges PARTITION BY source AS "edges"
  ON observation PARTITION BY vertex AS "observation"
  SourceKey('source')
  TargetKey('target')
  PairPotential('potential')
  StateNum('2')
  Observation('obs')
);
```

The following tables are used as an input to the function call above:

SOURCE	TARGET	POTENTIAL
1	2	0.3, 0.2, 0.4, 0.1
2	1	0.3, 0.4, 0.2, 0.1
...
31	37	0.2, 0.5, 0.1, 0.2
37	31	0.2, 0.1, 0.5, 0.2

Table 45: Edges

VERTEX	# OBSERVATIONS
1	1
2	2
...	...
43	2
47	2

Table 46: Observations

The function, when invoked, results in:

SOURCE	PROBABILITY 1	PROBABILITY 2
1	1	0
3	0.2	0.8
...
31	0.3	0.7
37	0.1	0.9

Table 47: Loopy Belief Propagation Probabilities

In the output above, if there are two outcomes (e.g., true and false) an estimate of each is provided. Note that the sum of both probabilities equal 1. In fact, having just one probability value column is enough as the probability of the other value can be computed as an inverse.

Sample Use Cases:

- Another application, besides the fraud use case mentioned above, is in the health sciences area where one could infer whether a new disease is auto-immune by comparing symptoms to known diseases.

Sample Verticals:

- Finance
- Pharmaceuticals
- Healthcare
- Voting Behavior/Policy analysis

Pagerank

Function Name: pagerank

This concept has been explained above in the contexts of the various centrality measures. To repeat, PageRank is a link analysis algorithm. It assigns a numerical weighting (between 0 and 1) to each node in a graph according to the link structure, with the purpose of "measuring" its relative importance within the graph. The sum of PageRanks in a graph is 1. The algorithm may be applied to any collection of entities with reciprocal quotations and references. Pagerank is the algorithm made most famous by Google towards identifying URLs that are the most relevant to a specific search query. The pages are ordered by a relevance metric (page rank) and search results are displayed thus.

Here is an example for a simple network (final results are expressed as percentages):

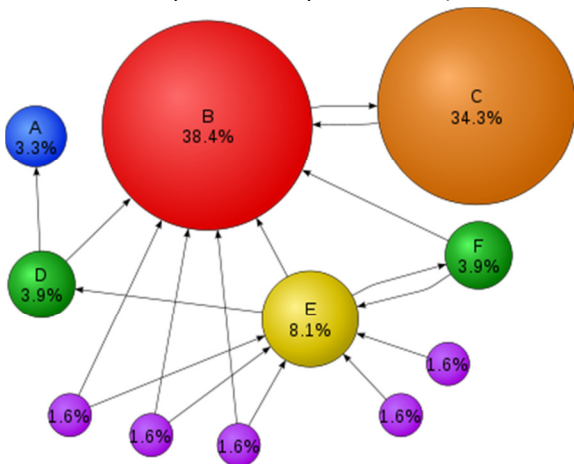


Figure 17: Pagerank

Function Call :

```
SELECT * FROM PAGERANK
(
  ON nodes AS vertices PARTITION BY nodeid
  ON edges PARTITION BY fromnodeid
  Accumulate('nodeid') );
```

The following tables are used as an input to the function call above:

ID	FROM	TO
1	1	2
2	1	3
3	3	1

Tables 48 and 49: Nodes and Edges

The function, when invoked, results in:

ID	PAGERANK
1	0.391620997027892
2	0.304189501486054
3	0.304189501486054

Table 50: Pagerank

Sample Use Cases:

- The core use case for this algorithm, or at least the most popular one, is its application in ranking page content by relevance to search terms.
- Other applications could include web site content optimization and customer experience management

Sample Verticals:

- E-Commerce

Path and Pattern Discovery Functions

Attribution

Function Name: attribution

Attributions provide a credit for the action(s) that result in an event. An event could be, completing a product purchase or downloading a pdf document for example. The traditional approach was to give all the credit to the last action that resulted in the event. This is not a fair method in that it discounts the impact of preceding actions that could've spurred the last action. It also would not include offline events.

There are several approaches to perform attribution and three dimensions to consider: weight, time, and quantity:

- The weight dimension consists of four models (single event, equal, weighted, exponential). In a single event approach, one action gets 100 percent credit; In the equal approach, all activities are treated equally; In a weighted approach, each activity is given a certain percentage attribution to the outcome; In the exponential approach, the nearest temporal actions are considered more important.
- The time dimension puts a time limit on the event set. Any event that falls out of the range is excluded.
- The quantity dimension puts a limit on the number of events preceding the outcome.

The following table shows the weighted approach in which events are weighted (attribution) based on their business importance:

ID	EVENT	TIMESTAMP	ATTRIBUTION	TIME TO CONVERSION
1	Store Visit	9/27/2001 23:00	0.09	-19
1	Paper Ad	9/29/2001 23:00	0.09	-17
1	Online visit	10/01/2001 23:00	0.09	-15
1	Online Ad Click	10/03/2001 23:00	0.09	-13
1	Browse offers	10/05/2001 23:00	0.09	-11
1	Store visit	10/07/2001 23:00	0.09	-9
1	Call Center Question	10/09/2001 23:00	0.09	-7
1	Email Click	10/11/2001 23:00	0.19	-5
1	Visit Online	10/15/2001 23:00	0.18	-1
1	Purchase Product	10/16/2001 23:00		

Table 51: Weighting events to determine attribution

Function Call:

```

SELECT *
  FROM attribution(
    ON logtable
    PARTITION BY event_id
    ORDER BY trans_ts desc
    METRIC ( 'COUNT:4' )
    MODEL ( 'NODUPE:EQUAL' )
    TARGETCOL ( 'TRANS_ID' )
  )
    
```

Sample Use Cases:

TERADATA ASTER DISCOVERY PORTFOLIO

- Understanding where to spend scarce marketing and sales dollars on channels in proportion to the value that each channel delivers to the enterprise. Value is determined by revenue per quarter, for example.
- Identifying and either discarding and/or improving inefficient channels.
- Discovering new and/or popular channels that are most favorable for customer acquisition and targeting these channels for extra treatments (e.g., campaigns, discounts).

Sample Verticals:

- E-Commerce
- Rretail
- Telecommunications
- Banking.

Basket Generator

Function Name: basket_generator

A market basket contains a set of items (products, services, actions) that are connected to each other through some attribute. For example, cars and tires are likely to be part of a basket of purchases as one is needed to use the other. Alternatively, soaps and toothpaste are part of the basket and are connected in that they are used for daily hygiene. The information from a market basket thus generated can be used for purposes of cross selling and up selling, in addition to influencing sales promotions, loyalty programs, store design, and discount plans.

Market baskets, typically in retail enterprises, are generated by affinity analysis. Affinity analysis is a data analysis and data mining technique that discovers co-occurrence relationships among activities performed by (or recorded about) specific individuals or groups. The basket generator function generates subsets of data from multiple rows that are connected by a common identifier. For example, if you have multiple rows recording purchase transactions of items by users, you can *connect* the items purchased based on some attribute that is common to all the rows or a majority subset of the rows. Once this basket is generated and compared across all the baskets among a group of purchasers, it becomes easier to see pairs, triplets, and other n-ordered groupings. It is from these groupings that the above use cases can be addressed.

Function Call:

```
SELECT store_id, sku1, sku2, sku3, count(1)
FROM basket_generator
(
  ON transactions
  PARTITION BY store_id, dt, reg_id, tran_no
  BASKET_SIZE(3)
  BASKET_ITEM('sku')
  ACCUMULATE('store_id')
  ITEM_SET_MAX(200)
)
GROUP BY 1,2,3,4;
```

The input data will include the following fields: Store ID, Transaction Date, Product SKU, and Transaction ID. Based on these data, the output generated will be the following:

STORE ID	SKU1	SKU2	SKU3	BASKET COUNT
1	83	97	1213	123
2	78	122	7812	88

Table 52: Market Basket by Store ID

In the above table, each store is associated with a specific set of baskets that have been identified based on transaction data. Some stores may have more than one basket of goods.

Sample Use Cases:

- Optimizing store shelf space and including products that are likely to quickly be sold as well as creating space for new products that are likely to show significant market penetration.
- Increasing market adoption for new products by bundling multiple products together (e.g., a purchase of toothpaste will be accompanied by a 50 percent discount on the purchase of a new perfume).
- Creating logical clusters of products and determining *bridge* products that enable customers to jump from one product basket to another.

Sample Verticals:


- Consumer retail
- Product merchandising
- Automotive manufacturing

Collaborative Filter

Function Name: cfilter

Collaborative filtering (cFiltering) is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The underlying assumption of the collaborative filtering approach is that if a person *A* has the same opinion as a person *B* on an issue, *A* is more likely to have *B*'s opinion on a different issue *x* than to have the opinion on *x* of a person chosen randomly. For example, a collaborative filtering recommendation system for television tastes could make predictions about which television show a user should like given a partial list of that user's tastes (likes or dislikes). Note that these predictions are specific to the user but based on preference information gleaned from many users. The table below shows how salsa and chips are highly collaborative items (meaning more frequently seen in market baskets versus other combinations).

ITEM 1	ITEM 2	CNTB	CNT1	CNT2	CFILTER SCORE
chips	salsa	1	2	2	0.25
salsa	chips	1	2	2	0.25
beer	chips	2	9	2	0.22222222222222
chips	beer	2	2	9	0.22222222222222
flour	sugar	3	7	7	0.183673469387755
sugar	flour	3	7	7	0.183673469387755
sugar	milk	3	7	9	0.142857142857143



TERADATA ASTER DISCOVERY PORTFOLIO

flour	milk	3	7	9	0.142857142857143
milk	flour	3	9	7	0.142857142857143
milk	sugar	3	9	7	0.142857142857143

Low collaboration

Table 53: Determining market baskets

The Cfilter function outputs a zscore column which is used by the recommender to filter out items that fall below a certain threshold (i.e. below the mean). The function automatically calculates the value of the zscore filter such that larger transaction tables with lots of items results in a more aggressive (higher zscore number) filter. The user can override the zscore filter entirely and specify a custom value with the FILTER_ZSCORE parameter. However, caution needs to be exercised when applying custom thresholds as the output may show too many items or too few items in the final recommendation table.

The Nearest Neighbor Algorithm is a type of a cFilter. cFiltering systems have many forms, but the two common ones are *User-based* and *Item-based*. In user-based collaborative filters (cFilters), these are the steps taken:

- Look for users who share the same rating patterns with the active user (the user whom the prediction is for).
- Use the ratings from those like-minded users found in step 1 to calculate a prediction for the active user.

In item-based cFilters (users who bought x also bought y), these are the steps:

- Build an item-item matrix determining relationships between pairs of items.
- Using the matrix, and the data about the current user, infer his taste.

Alternate illustration of affinity purchases:

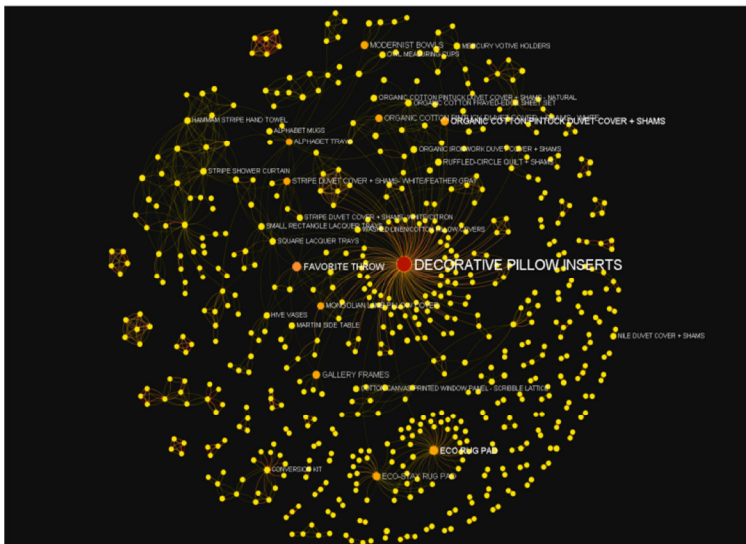


Figure 18: Affinity purchase illustration

Function Call:

```
SELECT *
FROM CFILTER (
  ON (SELECT 1)
  PARTITION BY 1
```

```

database('***')
userid('***')
password('***')
inputtable('cfilter_test')
outputTable('cfilter_test2')
inputColumns('item')
joinColumns('tranid')
otherColumns('storeid')
droptable('true')
);
SELECT * FROM cfilter_test2 ORDER BY score DESC;

```

Sample Use Cases:

- Understanding consumer behavior based on their specific purchasing patterns is important to create appropriate product mixes by specific segments (e.g., demography).
- Seeing which services are related allows companies to minimize costs of service, follow up visits, and other resources spent towards customer service.
- Building recommendation engines (e.g., music or book recommendations).

Sample Verticals:

- Manufacturing
- Healthcare
- Social media

Frequent Paths (aka Sequential Paths)

Function Name: SequentialPattern

The frequent paths function identifies a finite number of paths that lead to outcomes (see the nPath section later in the paper for a detailed description). Once these paths are identified, the path order which a new customer is likely to follow to reach said outcome is predicted (e.g., purchasing a product online; browsing for products online, and then going to the store offline for a purchase). This has broad applications in online/offline behavioral analysis, disease treatments, DNA sequencing, and stock movements.

The Teradata Aster nPath function identifies a finite set of paths that users take to reach an outcome (e.g., service cancellation). The frequent paths function provides a list of all those paths/sub-paths, generated by the Teradata Aster nPath function, that appear frequently. The sequence data are stored in a database with each pattern or sequence having its own ID. These sequences are subsequently analyzed to obtain the most frequent patterns. To illustrate this, let’s look at the following table that contains all the paths that are derived from nPath:

SEQUENCE ID	TIME (seconds)	ITEM (in sequence)
1	11	Login page
1	12	Books
1	13	Sign Off
2	21	Legal page
2	22	Furniture

TERADATA ASTER DISCOVERY PORTFOLIO

2	23	Purchase cart
3	31	Login page
3	32	Clothing
3	33	Sign Off

Table 54: Teradata Aster nPath sequences

A frequent pattern algorithm that asks for a list of all paths that start with the Login and end with a Sign Off will give a list of Sequence IDs 1 and 3. A count of these sequences will also be provided. On the other hand, if no path is specified, the algorithm can also mine all the sequences and create a subset of all high frequency paths and sub paths.

Function Call:

```
select * from SequentialPattern(  
    ON (select 1)  
    PARTITION by 1  
    INPUTTABLE('seq_4')  
    SEQUENCECOLUMN('sequence')  
    MINSUPPORT(2)  
    USERID('***')  
    PASSWORD('***') );
```

Sample Use Cases:

- Identifying specific sequences that are seen in customer purchasing patterns that can be used for marketing campaigns and customer experience enhancement initiatives.
- Isolating specific subsequences of DNA so that disease onset triggers can be understood and addressed pharmacologically and/or with other targeted treatment options.
- Determining Web access patterns such that all relevant content, depending on the site visitor attributes, can be provided in a customized manner.

Sample Verticals:

- E-commerce
- Pharmaceuticals
- Medical research
- Education

nTree

Function Name: ntree

nTree creates a hierarchical map of all paths in a decision tree to enable the discovery of the root node that results in a cascade of other sub nodes. For example, uncovering the first stock transaction that triggers other sell or buy trades is one application.

Function Call:

```
SELECT *  
FROM ntree  
( ON emp_table_aster  
  PARTITION BY department  
  ORDER BY order_column  
  ROOT_NODE(mgr_id = 'none')
```

```
PARENT_ID(mgr_id)
NODE_ID(id)
STARTS_WITH('ROOT')
MODE('DOWN')
OUTPUT('ALL')
RESULT(PATH(name) as path,
       path(id) as path2)
ALLOW_CYCLES('true')
) order by path, path2;
```

When the above function is invoked on a set of paths taken by individuals, the following is the output:

ID	PATH	PATH ID MAP
1	Dave	1
5	Dave->Mark	1->5
9	Dave->Mark->Kim	1->5->9
6	Dave->Rob	1->6
3	Dave->Rob->Donna	1->6->3
2	Dave->Rob->Donna->Pat	1->6->3->2
7	Dave->Rob->Donna->Pat->Don	1->6->3->2->7
5	Dave->Rob->Donna->Pat->Don->Mark	1->6->3->2->7->5
9	Dave->Rob->Donna->Pat->Don->Mark->Kim	1->6->3->2->7->5->9

Table 55: nTree Output

In the above example, for each path taken we can traverse back to the root node that triggered that path and determine what action was taken by that node (could be an individual or any other entity) and determine if that node triggered other actions and all the sub actions that were triggered.

Sample Use Cases:

- Call center behaviors can be analyzed to determine the primary reason behind certain calls and appropriate interventions can be taken at the point of first call capture. For example, an individual who starts by asking a question about a quality issue in a purchased product could either be directly sent over to the service unit or could be transferred to sales for alternate purchases.

Sample Verticals:

- Retail
- Finance
- Telecommunications
- Manufacturing

Path Generator, Path Starter, and Path Summarizer

Function Names: path_generator, path_start, path_summarizer

Together, the Path Generator, Path Summarizer, and Path Starter functions are used to perform clickstream analysis of common sequences of users’ page views on a website.

The **Path Generator** function takes as input a set of paths (e.g., series of page views) and generates the correctly formatted sequence and all possible sub-sequences. Together, the path generator, path summarizer, and path starter functions (see below) are used to obtain common sequences of users’ page views on a Web site.

TERADATA ASTER DISCOVERY PORTFOLIO

The **Path Summarizer** function takes the output of the path generator function and counts the number of times various paths were traveled and for each path generates a depth statistic, which is, the number of page views seen.

The **Path Starter** function generates all the child paths for a particular parent path and sums up the count of times each child path was traveled. The output of Path Summarizer function is the input to this function. This function generates all the children for a particular parent and sums up their count. Note that the input data has to be partitioned by the parent column.

The **Path Analyzer** function runs all three of these functions in order, passing the appropriate output of one function as input to the next function in the chain.

Function Call: Path Generator

```
SELECT *
FROM PATH_GENERATOR
(
  ON user_flows
  SEQ('path')
  DELIMITER(',')
);
```

The input table is:

ID	PATH	COUNT
1	a,b,c,d	1
2	a,b	2
3	b,e,g	5
4	a	7
5	a,e	5

Table 56: Path Generator Input

The function, when invoked, results in:

ID	PATH	COUNT	PREFIX	SEQUENCE
1	a,b,c,d	1	^,a	^,a,b,c,d
1	a,b,c,d	1	^,a,b	^,a,b,c,d
1	a,b,c,d	1	^,a,b,c	^,a,b,c,d
1	a,b,c,d	1	^,a,b,c,d	^,a,b,c,d
2	a,b	2	^,a	^,a,b
2	a,b	2	^,a,b	^,a,b
3	b,e,g	5	^,b	^,b,e,g

TERADATA ASTER DISCOVERY PORTFOLIO

ID	PATH	COUNT	PREFIX	SEQUENCE
3	b,e,g	5	^,b,e	^,b,e,g
3	b,e,g	5	^,b,e,g	^,b,e,g
4	a	7	^,a	^,a
5	a,e	5	^,a	^,a,e
5	a,e	5	^,a,e	^,a,e

Table 57: Path Generator Output (and input to Path Summarizer – see below)

Function Call: Summarizer:

```
SELECT *
FROM PATH_SUMMARIZER
(
ON output_of_path_generator
PARTITION BY (prefix)
SEQ('sequence')
PREFIX('prefix' )
PARTITIONNAMES('prefix')
DELIMITER(',')
CNT('cnt')
HASH('false')
);
```

This function, when invoked with the input from the Path Generator function above, results in:

NODE	PARENT	CHILDREN	COUNT	DEPTH	PREFIX
^,a	^	[(^,a,\$),(^,a,b),(^,a,e)]	15	1	^,a
^,a,\$	^,a		7	2	^,a
^,a,b	^,a	[(^,a,b,\$),(^,a,b,c)]	3	2	^,a,b
^,a,b,\$	^,a,b		2	3	^,a,b
^,a,b,c	^,a,b	[(^,a,b,c,d)]	1	3	^,a,b,c
^,a,b,c,d	^,a,b,c	[(^,a,b,c,d,\$)]	1	4	^,a,b,c,d
^,a,b,c,d,\$	^,a,b,c,d		1	5	^,a,b,c,d
^,a,e	^,a	[(^,a,e,\$)]	5	2	^,a,e
^,a,e,\$	^,a,e		5	3	^,a,e
^,b	^	[(^,b,e)]	5	1	^,b
^,b,e	^,b	[(^,b,e,g)]	5	2	^,b,e

TERADATA ASTER DISCOVERY PORTFOLIO

NODE	PARENT	CHILDREN	COUNT	DEPTH	PREFIX
^,b,e,g	^,b,e	[(^,b,e,g,\$)]	5	3	^,b,e,g
^,b,e,g,\$	^,b,e,g		5	4	^,b,e,g

Table 58: Path Summarizer Output (and input to Path Starter – see below)

Function Call: Path Starter

```

SELECT *
FROM PATH_START
(
  ON _user_flow_subpaths_
  PARTITION BY (parent)
  CNT('cnt')
  DELIMITER(',')
  PARENT('parent')
  PARTITIONNAMES('partitioned')
  NODE('node')
);

```

The function, when invoked, results in:

NODE	PARENT	CHILDREN	COUNT	DEPTH	PARTITION
^,b,e,g	^,b,e	[(^,b,e,g,\$)]	5	3	^,b,e,g
^,b,e	^,b	[(^,b,e,g)]	5	2	^,b,e
^,b	^	[(^,b,e)]	5	1	^,b
^,a,e	^,a	[(^,a,e,\$)]	5	2	^,a,e
^,a,b,c,d	^,a,b,c	[(^,a,b,c,d,\$)]	1	4	^,a,b,c,d
^,a,b,c	^,a,b	[(^,a,b,c,d)]	1	3	^,a,b,c
^,a,b	^,a	[(^,a,b,\$),(^,a,b,c)]	3	2	^,a,b
^,a	^	[(^,a,\$),(^,a,b),(^,a,e)]	15	1	^,a
^		[(^,a),(^,b)]	20	0	^

Table 59: Path Starter Output

Sample Use Cases:

- Attribution analysis
- SEM/SEO
- Path to Purchase
- Clickstream analysis

Sample Verticals:

- Retail
- Advertising

Teradata Aster nPath

Function Name: nPath

Teradata Aster nPath is one of Aster’s core functions that has been patented and is used extensively in behavioral analytics. It is an out-of-the-box SQL-MR function that allows sequential analysis through a single pass through of multi-structured data to link an outcome (e.g., churn, cart abandonment) with a path or set of paths. The Teradata Aster nPath is implemented by first selecting the set of page views comprising a given individual’s session. It then stitches together multiple sessions and ascertains the path taken. This function allows the use of a regular expression to specify a pattern you want to see and isolate all patterns that match. It also enables the computation of aggregates or the discovery of particular values in the matched pattern.

For example, let’s take the case of a mobile telecommunications services company that interacts with its customers directly through their brick and mortar store, online portal, and the call center. Furthermore, let’s also say that this company also wants to determine why an increasing proportion of its customers are cancelling their service contracts. The Teradata Aster nPath application allows the analyst to look at all the activities across the call center and associate a specific path for each customer over a defined period of time. While one cannot take action on **all** identified paths, it is easy to iteratively determine a handful of paths or a specific group of drivers that lead to customer cancellations. This allows the business to monitor these finite paths and implement specific intervention plans such as incentives, marketing campaigns, customer service improvements, pricing changes, or even doing nothing. The figure below shows a Sankey visualization.

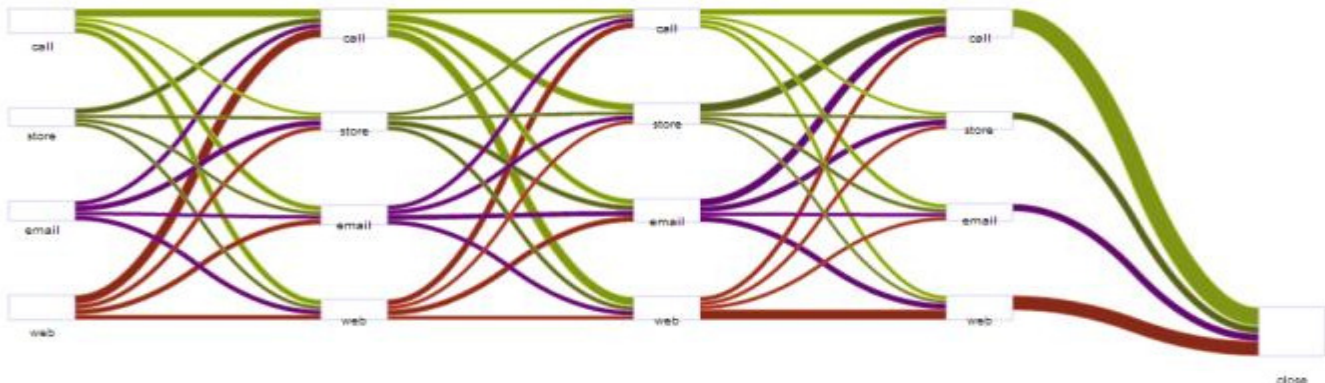


Figure 19: Sankey visualization of customer paths.

Function Call:

```
select * from npath( on aggregate_clicks
    partition by sessionid
    order by clicktime asc
    mode( nonoverlapping )
    pattern( 'H+.D*.X*.P1.P2+' )
    symbols('true' as X, pagetype = 'home'as H, pagetype <>'home' and pagetype<>'checkout' as D,
    pagetype='checkout' as P1,
    pagetype='checkout' and productprice>100 and productprice >
    lag(productprice, 1,100::real)as P2)
    result(first(productprice of P1) as first_product, max_choose(productprice, productname of P2) as
    max_product,
    first(sessionid of P2) as sessionid )
)
order by sessionid ;
```

TERADATA ASTER DISCOVERY PORTFOLIO

Other Teradata Aster nPath use cases include:

- Categorizing entities based on observed patterns (e.g., “loyal customers” or “price sensitive shoppers”).
- Identifying complex networks of transactions and relationships over time and geography.
- Finding links between behavioral patterns and business outcomes such as fraud or loan defaults.
- Identifying influencing patterns of activity that can be used to make recommendations (e.g., music purchases).
- Identifying multiple negative experiences of the customer or violations in SLAs at various customer touch points.
- Managing high-value customers’ experience. For example, in the telecommunications business route analysis of frequently used cell sites by high-value customers is done to assign a rule-based preferential quality of service. (QoS) for a cell site ahead of the customer’s arrival at that site.
- Optimizing marketing spend by focusing only on channels with high customer traffic.

Sample Verticals:

- Manufacturing
- Retail
- Insurance (risk)
- network security
- Gaming

WSRecommender

Function Name: WSRecommender

The WSRecommender function Suggests items that can be recommended to individuals depending upon their previous activities and the similarities between various products that have been purchased. This function underlies the concept of a Recommendation System. Recommendation (or recommender) systems are a subclass of information filtering system that seek to predict the 'rating' or 'preference' that user would give to an item (such as music, books, or movies) or social element (e.g. people or groups) they have not yet considered.

Function Call:

```
SELECT * FROM WSrecommender
(
ON (
  SELECT * FROM WSrecommenderreduce
  (
    ON <item_table_name> AS item_table
    PARTITION BY <item_table_partition_column>
    [ ORDER BY <item_table_ordering_columns> ]
    ON <user_table_name> AS user_table
    PARTITION BY <user_table_partition_column>
    [ ORDER BY <user_table_ordering_columns> ]
    [ ITEM1('<item1_column>') ]
    [ ITEM2('<item2_column>') ]
    [ ITEMSIMILARITY('<similarity_column>') ]
    [ USERID('<user_column>') ]
    [ USERPREF('<preference_column>') ]
    [ USERITEM('<item_column>') ]
    [ ACCUMULATEITEM('<accumulate_item_column1>', '<accumulate_item_column2>', ...) ]
    [ ACCUMULATEUSER('<accumulate_user_column1>', '<accumulate_user_column2>', ...) ]
  )
) AS temporary_table
```

TERADATA ASTER DISCOVERY PORTFOLIO

```
PARTITION BY <temporary_table_partition_columns>
)
```

The input tables for this function are:

ITEM 1	ITEM 2	SIMILARITY	STORE ID	REGION
milk	bread	4	1	west
milk	butter	3	1	west
milk	milk	4	1	west
bread	bread	5	null	west
bread	butter	3	1	west
bread	milk	4	1	west
butter	bread	3	1	west
butter	butter	3	1	west
butter	milk	3	1	west

Table 60: Store Items by similarity

ITEM	USRID	PREF	MONTH
milk	Bob	4	NULL
milk	Alice	0	NULL
bread	Alice	2	Jan
bread	Bob	0	Feb
butter	Alice	0	NULL
butter	Bob	2	NULL

Table 61: Individual Preference

The function, when invoked, results in:

ITEM	USER	RECOMMENDATION	VIEWED	STORE ID	REGION	MONTH
bread	Alice	2	1	NULL	west	Jan
bread	Bob	3.14286	0	NULL	west	Feb

Table 62: Recommendations

Based on Bob's high preference for milk, in the above example, he has been given a high recommendation of bread.

TERADATA ASTER DISCOVERY PORTFOLIO

Sample Use Cases:

- Product and service recommendations

Sample Verticals:

- Entertainment
- Education,
- Retail
- E-Commerce
- Banking & Finance

Statistics and Machine Learning Functions

Approximate Distinct Count

Function Name: approx._dcount_combine

This function computes a count value, based on probabilistic counting algorithms, that is considered a fair approximation of the actual value of the count. For large populations where it may be impossible or hard to know actual numbers (e.g., population of the USA) an approximated number would be a reasonable proxy for the actual.

Function Call:

```
SELECT *
FROM APPROX_DCOUNT_COMBINE
(
  ON APPROX_DCOUNT_PARTIAL
  (
    ON page_tracking
    COLUMNS ('member_id', 'page_key', '(member_id:page_key)')
    ERROR (1)
  )
  PARTITION BY column_name
);
```

The input table for the function is:

MEMBER ID	PAGE KEY	REFERRER	PAGE SEQUENCE
1	Home	http://google...	1
1	Profile		2
2	Jobs		1
4	News	http://yahoo...	1
5	Profile		1

Table 63: Clickstream Information

The function, when invoked, results in:

COLUMN NAME	COUNT
member_id	4
member_id_page_key	5
page_key	4

Table 64: Distinct Counts

Sample Use Cases:

- Any use case where there is a need for summary statistics.

Sample Verticals:

- Any vertical where there is a need for summary statistics

Average (Simple, Moving, Weighted)

Function Names: avg, cmavg, emavg, smavg, wmvavg, vwap

Simple Average -- Computes an arithmetic mean on the data. An arithmetic mean is the sum of all the values in the data stream divided by the total number of values seen. For example, if the data set is {1, 5, 8, 10, 17, 18}, the sum of all values is 59 and the mean is 59 divided by the total number of observations, 6. The mean, therefore, is 9.83.

Cumulative Moving Average -- Computes an average on data that arrive in an ordered stream. When more data arrive, the interval could be moved, and a new metric calculated on the same number of observations. For example, when calculating average weather for a 24-hour time period, the window is advanced as each hour passes.

Exponential Moving Average -- Computes the average over a number of points in a time series while applying an exponentially decaying or damping (weighting) factor so that more recent values are given a heavier weight.

Simple Moving Average -- Computes the average over time of a data series and plots the averages in a line graph.

Weighted Moving Average -- Computes the average of time series data for different points in time with each time calculation weighted differently (e.g., peak seasons are weighted more). The values are plotted on a line graph.

Volume-Weighted Average Price -- Computes the ratio of the value of a traded item to the total volume traded over a time horizon (e.g., day). It is a measure of the average price a traded at over the trading horizon.

Sample Use Cases:

- Any use case where there is a need for summary statistics.

Sample Verticals:

- Any vertical where there is a need for summary statistics.

Canopy

Function Name: canopy

This function speeds up the clustering operation on large data sets by grouping objects (pre-clustering) into overlapping subsets (*canopies*). Canopies have the advantage of preparing the data for more rigorous clustering techniques (e.g., k-means). Therefore data that are outside a given canopy need not be considered for the advanced techniques and so they can be applied in parallel across all canopies and hence speeds time to value. By starting with an initial partitioning into canopies, the number of more expensive distance measurements can be significantly reduced by ignoring points outside of the initial canopies. Clustering is a mechanism that involves partitioning of the object into groups (called as clusters) such that the similarity between members of the same group is maximized and similarity between members of different group is minimized. Clustering can be considered as a most important unsupervised learning problem, and thus deals with finding a structure in a collection of unlabeled data (e.g., document classification, product ratings).

Let's illustrate canopies with a simple example. In the healthcare industry, one way by which insurance premiums are determined is based on clustering "similar" patients into buckets. Similarity, of course, can depend on a number of dimensions such as age, physical traits, medical history, year of birth, gender, geographic locations lived in, and many other variables. Instead of

TERADATA ASTER DISCOVERY PORTFOLIO

considering ALL such variables to cluster like patients a pre-processing step is taken to cluster patients on one or two dimensions, say, DOB and geographic locations lived in. This pre-processing step results in canopies with multiple patients simultaneously being part of multiple clusters or canopies. Once these pre-processed canopies are created then more advanced clustering techniques, such as k-means or Greedy agglomerative Clustering (GAC) can be applied within these clusters to create tighter groups.

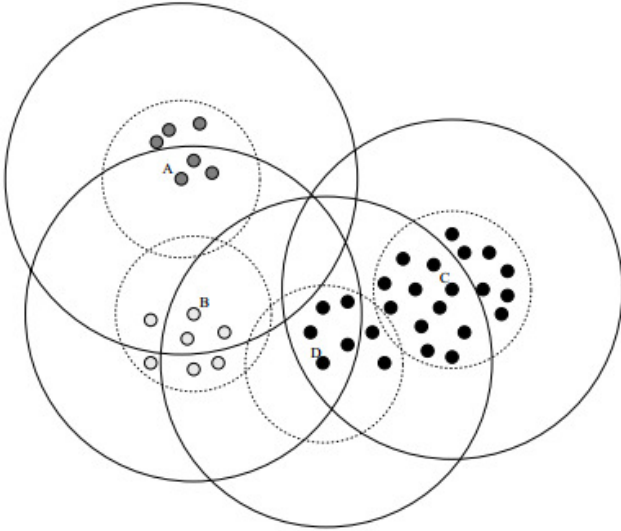


Figure 20: Overlapping Canopies

The above figure illustrates the creation of canopies based on some distance metric. In the above example, the distance metric was year of birth and geographical locations lived in. In other examples, it could literally be the numerical difference between, say, reviewer ratings. Two reviewers can be considered to be part of the same canopy if their ratings are separated only by 1 point, and so on.

Sample Use Cases:

- New document classification.
- Product recommendations

Sample Verticals:

- Retail
- Healthcare
- Medical Imaging

Confusion Matrix

Function Name: ConfusionMatrix

This function provides visualization, in a table, of the performance of a supervised learning algorithm. Each column of the matrix represents the instances in a predicted class, while each row represents the instances in an actual class. The name stems from the fact that it makes it easy to see if the system is confusing two classes (i.e., commonly mislabeling one as another). Ideally, if the predicted and the actual classes match perfectly, there is no confusion or mislabeling. The greater the proportion of mislabeled categories, the lower the predictive capability. The following table illustrates this concept:

	Predicted	
	Negative	Positive

TERADATA ASTER DISCOVERY PORTFOLIO

Actual	Negative	10	20
	Positive	15	5

Table 65: Confusion Matrix

From the above table we can compute the following metrics:

- **Accuracy** = Proportion of the sum of accurate outcomes = $15/50 = 30\%$
- **True Positive Rate (Recall)** = Proportion of positive cases that were classified correctly = $5/20 = 25\%$
- **True Negative Rate (Recall)** = Proportion of negative cases that were classified correctly = $10/30 = 33\%$
- **False Positive Rate** = Proportion of negative cases that were incorrectly classified as positive = $20/30 = 67\%$
- **False Negative Rate** = Proportion of positive cases that were incorrectly classified as negative = $15/20 = 75\%$
- **Precision** = Proportion of the predicted positive cases that were correct = $5/25 = 20\%$

Function Call:

```
select * from ConfusionMatrix(  
  on test_table  
  PARTITION BY expect  
  expectcolumn('expect')  
  predictcolumn('predict')  
);
```

Sample Use Cases:

- Fraud alerting (recognizing an e-mail to be spam versus non-spam)
- Transaction classification (purchaser versus browser)

Sample Verticals:

- Law enforcement (crime prevention)
- Manufacturing
- Environmental Studies
- Bioinformatics

Correlation

Function Names: Corr_reduce, Corr_Map

Correlation measures the strength of the association between two variables. For example, it is easy to imagine that quality of a vegetable (freshness) and the number of days it is left outside the refrigerator is inversely related - the more the number of days outside the refrigerator, the lesser the freshness. Similarly, when looking at the crime statistics, one can see the positive relationship between post-jail rehabilitation programs and crime recidivism. However, not all correlation signifies causation. It is equally easy to imagine the association between rising temperatures and the appetite for rice. While there may be a numeric association between higher temperatures and increasing numbers of rice eaters, it is a spurious relationship.

Function Call:

```
SELECT *  
FROM CORR_REDUCE  
(  
  ON CORR_MAP
```



```
(
  ON income_statistics
  COLUMNPAIRS ('[1:2]','income:years_of_education')
  KEY_NAME ('key')
)
PARTITION BY key
);
```

The input table for this function is:

INCOME	YEARS OF EDUCATION
125000	19
100000	20
40000	16
35000	16
41000	18
29000	12
35000	14
24000	12
50000	16
60000	17

Table 66: Income and Education Data

The function, when invoked, results in:

CORRELATION VARIABLES	CORRELATION COEFFICIENT
Income:Years_of_Education	0.788726

Table 67: Correlation Coefficient

There is a high correlation between income levels and years of education.

Sample Use Cases:

- Any use case where there is a need for summary statistics.

Sample Verticals:

- Any vertical where there is a need for summary statistics.

Distribution Matching

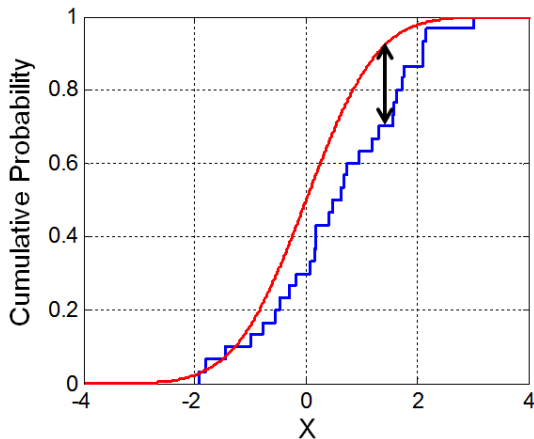
Function Name: DistnmatchReduce

TERADATA ASTER DISCOVERY PORTFOLIO

Distribution matching measures goodness of fit and summarizes the discrepancy between observed values and the values expected under the model in question. Such measures can be used in hypothesis testing, testing whether two samples are drawn from identical distributions, or testing whether outcome frequencies follow a specified distribution. The four tests noted below are used to determine the match:

- Anderson-Darling Test, Kolmogorov-Smirnov test, Cramer-von Mises Criterion, Pearson's Chi-Squared Test.

The other application of the distribution matching algorithm is to find out the underlying distribution of a sample data set and create alternate data sets that match the same distribution. The figure below explains the difference between an actual



distribution (BLUE) and the extrapolated distribution (RED) and the difference between the two.

Figure 21: Actual and Extrapolated Distributions

Function Call:

```
SELECT * FROM DistnmatchReduce (  
  ON DistnmatchMultipleinput (  
    ON (SELECT RANK() OVER (PARTITION BY distribution, times, storeid, variable_name ORDER BY price) AS rank, *  
      FROM distnmatch_continuous WHERE price IS NOT NULL) AS input PARTITION BY ANY  
    ON (SELECT distribution, times, storeid, variable_name,  
      COUNT(*) AS group_size,  
      AVG(price) AS mean,  
      STDDEV(price) AS sd,  
      CASE  
        WHEN MIN(price) > 0 THEN AVG(LN(CASE WHEN price > 0 THEN price ELSE 1 END))  
        ELSE 0  
      END AS mean_of_ln,  
      CASE  
        WHEN MIN(price) > 0 THEN STDDEV(LN(CASE WHEN price > 0 THEN price ELSE 1 END))  
        ELSE -1  
      END AS sd_of_ln,  
      MAX(price) AS maximum,  
      MIN(price) AS minimum  
    FROM distnmatch_continuous  
    WHERE price IS NOT NULL  
    GROUP BY distribution, times, storeid, variable_name)  
  AS groupstats DIMENSION  
  VALUECOLUMN('price')  
  TESTS('KS', 'AD', 'CHISQ')  
  GROUPINGCOLUMNS('distribution', 'times', 'storeid', 'variable_name')
```

TERADATA ASTER DISCOVERY PORTFOLIO

```
MINGROUPSIZE('50')
CELLSIZE('10')
)
PARTITION BY distribution, times, storeid, variable_name
);
```

Sample Use Cases:

- Creating an A/B testing environment to test the impact of specific products and services. A simple comparison of key metrics will help organizations identify specific drivers of action (e.g., employee retention options that would include a combination of financial incentives and flexible work hours or a different combination).
- Predicting combinations of therapies likely to be successful for groups of patients afflicted by a disease.
- Segmenting images for more granular analysis (e.g., left ventricle of the heart) that would ensure a detailed physiological analysis and provide timely care.

Sample Verticals:

- Healthcare
- Insurance
- Pharmaceuticals
- Advertising

F-Measure

Function Name: fmeasure

This function measures a hypothesis test's accuracy. The F measure can be interpreted as a weighted average of the precision (proportion of correct results from all returned results) and recall (proportion of correct results from all results that should have been returned) and its best value is a 1, and worst is a 0. When all predicted values conform to the actuals then the hypothesis is fully proven. This degree of accuracy worsens as the mismatch increases.

In the section above on Confusion Matrix, we reviewed the concepts of Precision and Recall. To recap, precision is the proportion of the predicted positive cases that were correct. Recall is the proportion of positive cases that were classified correctly. If we value, precision more than recall or vice versa then the choice of a metric is easy. However, if we want to evaluate how well a metric does on both precision and recall then the f-measure is used in that evaluation. The f-measure is calculated as:

$(2 * \text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})$. So, if the precision and recall for a given algorithm are respectively 75% and 56%, the f-measure is .64. This can be computed for another algorithm and the f-measure can be computed for that and the higher f-measure will be chosen and will be indicative of a better algorithm.

Sample Use Cases:

- Fraud alerting (recognizing an e-mail to be spam versus non-spam)
- Transaction classification (purchaser versus browser)

Sample Verticals:

- Law enforcement (crime prevention)
- Manufacturing
- Environmental Studies
- Bioinformatics

Generalized Linear Model (GLM)

Function Name: glm

Generalized linear models (GLMs) are a large class of statistical models for relating responses to linear combinations of predictor variables. Examples of GLMs are linear regression, multiple logistic regressions, Poisson regressions, and other such models. GLMs are an extension of the linear modeling process that allows models to be fit to data that follow probability distributions other than the normal distribution (e.g., Poisson, Binomial, and Multinomial). GLMs also relax the requirement of equality or constancy of variances that is required for hypothesis tests in traditional linear models. To provide a layman's explanation of the GLM, let's consider a cooking analogy:



....start with finished pizza and try to explain how it is made...

- We specify which *ingredients* to add (X) - the independent variable (also called the systematic component)..
- For each ingredient, GLM finds the *quantities* (β) that produce the best reproduction - β is the degree of influence that each independent variable exerts.
- Then if you tried to *make the pizza* (Y) (the dependent variable) with what you know about X and β , then the error (e) would be the difference between the original pizza/data and the reconstructed version.

This gives us the basic GLM equation: $Y = X_1 \beta_1 + X_2 \beta_2 + e$. The variance (error) is due to a set of uncontrolled influences that have not been anticipated or easily observable to include into the model. These could be one-time random events such as a bad quality ingredient or an oven malfunction. The GLM assumes that the errors follow a normal distribution pattern, is uncorrelated with the actual observations, and there's no directional change over time.

Function Call:

```
SELECT * FROM GLM (  
  ON (SELECT 1)  
  PARTITION BY 1  
  database('beehive')  
  userid('beehive')  
  password('beehive')  
  inputTable('glm_test2')  
  outputTable('glm_output2')  
  columnNames('response', 'col1', 'col2', 'col3', 'col4')  
  categoricalColumns('col1', 'col3 : (yes)')  
  family('LOGISTIC')  
  link('CANONICAL')  
  weight('1')  
  threshold('0.01')  
  maxIterNum('10')  
);
```

Sample Use Cases:

- Isolating a small group of critical attributes that influence individual behavior and being able to effect changes to these attributes (e.g., increases in floating vacation days versus having a long but mandatory vacation timeline, increases employee retention).
- Identifying spurious relationships that do not add any value to understanding customers, and therefore focusing all enterprise activity on key drivers that directly affect customer behaviors.

Sample Verticals:

TERADATA ASTER DISCOVERY PORTFOLIO

- Manufacturing
- Entertainment
- Government (service departments like the DMV)
- Insurance

Histogram

Function Names: histogram_reduce, histogram_map

Histogram is the process of the mapping data points to different bins based on some criteria and then calculating the frequency of each bin. The following is an illustration of a histogram:

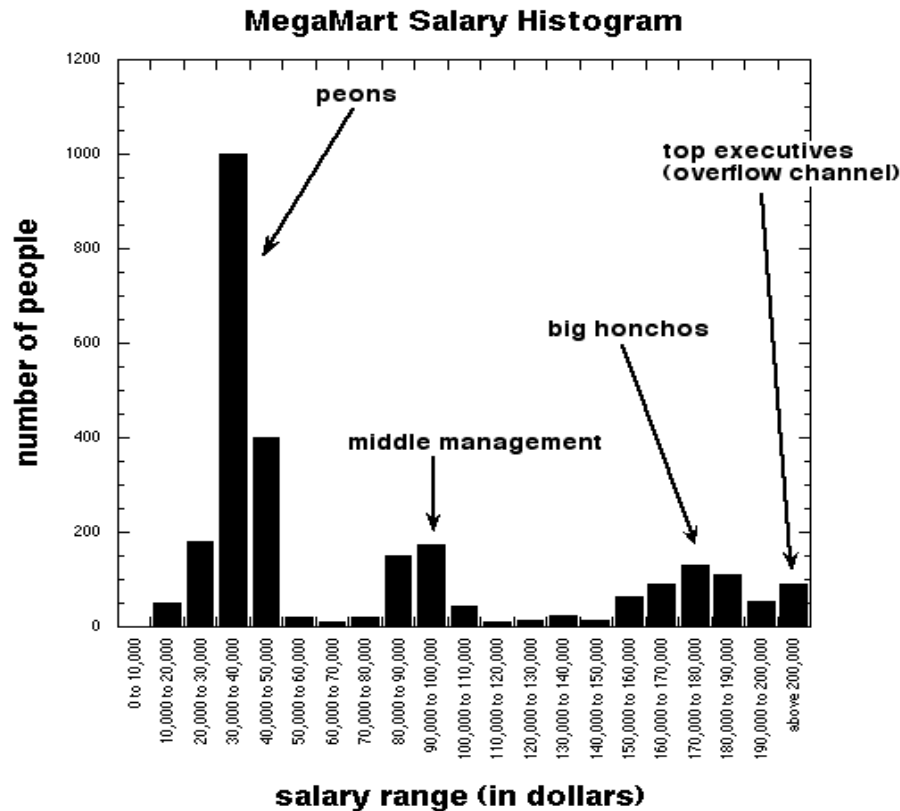


Figure 22: Salary Histogram

Function Call:

```
SELECT * from histogram_reduce(  
ON histogram_map  
  (  
    ON am_histogram_data  
    BIN_SIZE('10')  
    START_VALUE('0')  
    VALUE_COLUMN('age')  
  ) partition by(bin)  
ACCUMULATE('bin','start_bin','end_bin')  
) order by bin;
```

Sample Use Cases:

- Any use case where there is a need to understand data distributions.

Sample Verticals:

- Any vertical where there is a need to understand data distributions.

K-Means (Clustering and Plotting)

Function Name: kmeans

The k-Means clustering function performs a method of cluster analysis which aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean. The main idea is to define k centroids, one for each cluster - the shorter the distance to the centroid, the greater the likelihood of membership in that cluster. The k-means plotting function uses the k-Means output (centroids) to cluster new data points around these centroids (mean points).

The k-means algorithm is implemented in 4 steps:

- Partition objects into k nonempty subsets
- Compute seed points as the centroids of the clusters of the current partition (the centroid is the center, i.e., *mean point*, of the cluster)
- Assign each object to the cluster with the nearest seed point
- Go back to Step 2, stop when no more new assignment

Function Call:

```
SELECT * FROM kmeans
(
  ON (SELECT 1)
  PARTITION BY 1
  DATABASE('beehive')
  USERID('beehive')
  PASSWORD('beehive')
  INPUTTABLE('kmeanssample')
  OUTPUTTABLE(kmeanssample_centroid')
  NUMBERK(3)
  threshold('0.01')
  maxIterNum('10')
);
```

Sample Use Cases:

- New document classification.
- Product recommendations.

Sample Verticals:

- E-Commerce
- Retail
- Legal (document classification)
- Pharmaceuticals

K-Nearest Neighbor Classification Algorithm

Function Names: knn

The K-Nearest Neighbor (KNN) Classification algorithm classifies an object by a majority vote of its neighbors, with the object being assigned to the class most common amongst its k nearest neighbors (k is a positive integer, typically small). To describe it

TERADATA ASTER DISCOVERY PORTFOLIO

differently, the algorithm's main purpose is to figure out what the decision boundary of a given data set data is. A decision boundary separates one set of objects from the other. All values, for example, that are closer to one average are put together in one cluster, and other values that are closer to the second average are clustered elsewhere. Classifications become more complex as the boundaries between data points become opaque. The more complex the boundaries are, the greater the effort required to compute the clusters.

One practical application of KNN is in grouping new customers into pre-existing groups. Let's say that all existing customers have a few attributes, and they must all be categorized, based on their previous purchases as *big spender*, *medium spender*, *small spender*, and *will never buy anything*. Let k (the number of existing customers to compare to) be five, and the new customer is "Customer ZZ". This algorithm searches for the five customers closest to "Customer ZZ", i.e. most similar in terms of attributes. If four of them were *medium spenders* and one was *small spender*, then the best guess for "Customer ZZ" is *medium spender*.

While this is intuitive and easy to understand, there are many hidden challenges like scale normalization, multi-collinearity between attributes that may actually create incorrect clusters. For example, if the attributes of comparison are age, income, and number of times shopped, "Customer ZZ" would be represented as (25, 55K, 0), and the nearest neighbor would be (75, 54K, 35). This erroneously equates a 25-year-old to a 75-year-old purely based on income alone. The scaling and choice of attributes is important to get the right classification for each new customer. Other uses of KNN include content retrieval, pharmaceutical interactions, and many more. The figure below illustrates how new members to a group are evaluated based on the closeness to the centroid of existing clusters.

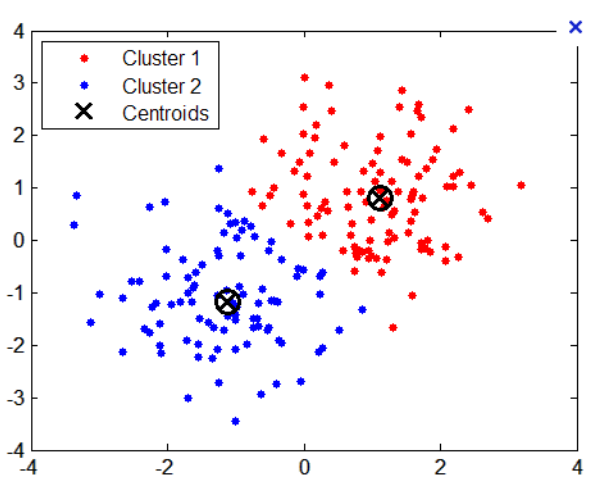


Figure 23: Nearest Neighbor classification

The closer the distance is to one centroid, the more likely the membership in that cluster. Also, the greater the number of dimensions, the harder it is to calculate the centroid.

Function Call:

```
SELECT * FROM knn(  
  ON (SELECT 1)  
  PARTITION BY 1  
  DATABASE('beehive')  
  USERID('beehive')  
  PASSWORD('beehive')  
  TRAINING_TABLE('knntesting')  
  TEST_TABLE('knntest')  
  NUMBERK(3)  
  
  RESPONSECOLUMN('category')  
  DISTANCE_FEATURES('x','y')
```

```
VOTING_WEIGHT(1)
TEST_POINT_KEY('id')
DISTANCE('a.jar','com.xxx.MyDistance')
```

```
OUTPUT_TABLE('knn_output')
);
```

Sample Use Cases:

- Classifying products into logical clusters such that item proximities can be identified and marketed better to customers.
- Creating a set of new products that could be suggestions for customer upsell.
- Intelligently retrieving content from large unstructured data based on some prior context (e.g., documents pertaining to a Supreme Court decision from 1952).
- Understanding voting behavior and segmenting voting data by various attributes (e.g., demographics) that could be used for fundraising and *get out the vote* campaigns.

Sample Verticals:

- Branding and advertisement
- Legal
- Management consulting
- Government (redistricting, elections management)

Least Absolute Shrinkage and Selection Operator (LASSO)

Function Name: lars (the method is lasso)

This function estimates the coefficients for each input variable which is used to make predictions for the response variable. It is different from the ordinary least squares approach in that the sum of the regression coefficients is constrained. The advantage of the LASSO function lies in this coefficient sum constrained in that only the most important variables will be included in the model. When there are many possible predictors, many of which actually exert zero to little influence on a target variable, the lasso can be especially useful in variable selection.

Function Call:

```
SELECT * FROM LARS (
  ON (SELECT 1)
  PARTITION BY 1
  database('***')
  userid('***')
  password('***')
  inputTable('diabetes')
  outputTable('diabetes_lasso')
  columnNames('y', 'age', 'sex', 'bmi', 'map', 'tc', 'ldl', 'hdl', 'tch', 'ltg', 'glu')
  method('lasso')
  intercept('true')
  normalize('true')
  maxIterNum('20')
);
```

Sample Use Cases:

- Understanding key predictors for bond risk premiums.
- Determining key variables that predict disease outbreak

Sample Verticals:

- Finance
- Healthcare

- Manufacturing, Insurance

Linear Regression

Function Name: Linreg

This function determines a value contributed to the dependent variable by each independent variable. The dependent variable (response) is a linear function of all the attributes that affect the response value (e.g., higher prices and lower income lead to lower automobile purchase volumes). Linear regression models are expressed in a formula such as

$$Y = I + aX_1 + bX_2 + cX_3 + \epsilon$$

- $X_1, X_2,$ and X_3 are independent variables based on whose values the value of Y can be predicted.
- $A, b, c,$ are constant values (coefficients) which are multiplied by the values of their respective independent variables. An independent variable that has an inverse relationship with the dependent variable will have a negative coefficient. The stronger that relationship the higher will be the number.
- ϵ is the margin of error.
- I is the intercept, a constant value.

Let's take the following illustration that shows the relationship between High School GPA and University GPA:

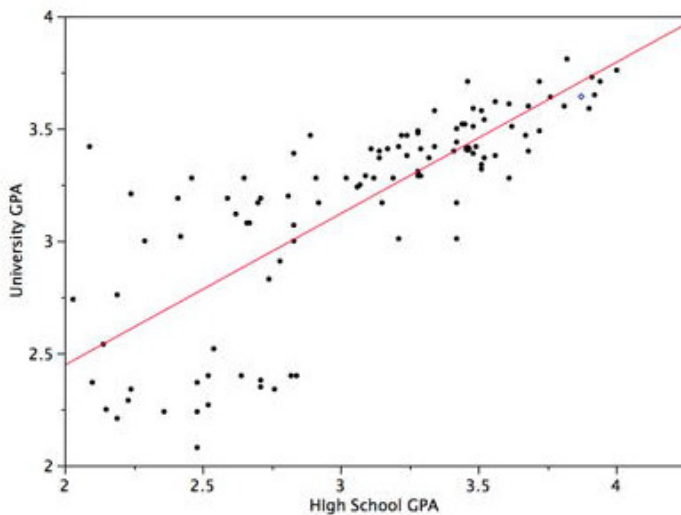


Figure 24: University GPA Prediction Regression Model

The line that is fit through the current observations (scatter plot) is the regression line. The principle underlying the regression line is called least squares. Following the least squares principles ensures that we get the best fit line in the scatter plot only if it is the line which makes the vertical distances from the data points to the line as small as possible. As one can see, with any line that is fit into the scatter plot there is an error. In other words, not all the observations fall on the line. The least square approach to finding the right line will ensure that the best line minimizes the sum of the squares of the errors made in the results of every single equation.

In the above figure, the point where the red line touches the y axis (university GPA) is called the intercept. Accordingly, our regression equation could be expressed as:

$$\text{University GPA} = (0.675)(\text{High School GPA}) + 1.097$$

Therefore, a high school GPA of 3.2 will likely result in a University GPA of 3.26.

Function Call:

```
SELECT *
FROM LINREG
(
  ON LINREGMATRIX
  (
    ON data_set
  )
  PARTITION BY 1
);
```

Sample Use Cases:

- Large medical trials that determine relationships between childhood obesity and long term chronic illnesses.
- Predicting risk among drivers based on age and socio economic backgrounds.

Sample Verticals:

- Medical research
- Education
- Insurance

Logistic Regression

Function Name: Logisticreg

This function predicts the outcome of a categorical dependent variable (a dependent variable that can take on a limited number of values – e.g., 0, 1, or 2). For example, we can use logistic models to predict credit default (0) versus or solvency (1) based on other discrete (gender) or continuous variables (income levels). Logistic regression is often used because the relationship between the DV (a discrete variable) and a predictor is non-linear.

Medical research is one area in which such non-linear relationships are found. For example, the probability of heart disease changes very little with a ten-point difference among people with *low-blood pressure*, but a ten point change can mean a drastic change in the probability of heart disease in people with *high blood-pressure*. If the relationship were assumed to be linear then a 10 point difference regardless of where in the blood pressure scale a patient is located would not affect treatment options, which could be a big mistake.

The ultimate goal of the logistic model is to estimate the risk/probability of a particular discrete outcome (pass/fail; diseased/healthy; democrat/republican) based on a set of independent variables. The logistic function (L) is usually expressed as follows:

$L = \text{logarithm}(\text{probability of event}/\text{e-probability of event}) = 1 + aX_1 + bX_2 + cX_3 + \epsilon$; The (probability of event/e-probability of event) is called the **odds ratio** and the logarithm of the odds ratio is called **logit**.

Let's illustrate this with a simple example. Let's assume that a person's risk of a heart attack (0=no heart attack; 1=heart attack) is based on her age (continuous variable), cholesterol level (0=low; 1=high); ECG (0=normal; 1=abnormal). The logistic model for this function, let's say, is as follows:

Probability (heart attack) = $1/(1 + e^{-(a + b*\text{CHOLESTEROL} + c*\text{AGE} + d*\text{ECG})})$ – by the way, this is the transformed version of the equation provided earlier in this section. The values we have for a, b, c, and d can be obtained by a process called maximum likelihood estimation. The values we have here are as follows:

a = -3.9; b = .65; c = .02; d = .34. The above equation then would be expressed as:

TERADATA ASTER DISCOVERY PORTFOLIO

Probability (heart attack) = $1/(1 + e^{-(3.9 + .65*CHOLESTEROL + .02*AGE + .34*ECG)})$. A patient who is then 40 years old, with high cholesterol (1) and normal ECG will then have a risk of getting a heart attack of .1090. This translates to about a 11% probability of heart attack or low risk.

Function Call:

```
SELECT *
FROM LOGISTICREG
(
  ON (select 1)
  PARTITION BY 1
  DATABASE('beehive')
  INPUTTABLE('log_reg_driver_data')
  USERID('beehive')
  PASSWORD('beehive')
  OUTPUTTABLE('log_reg_output')
  INITCOEFF('0.1','0.1','0.1','0.1','0.1','0.1')
  COLUMNNAMES('Y','X1','X2','X3','X4','X5')
  MAXITERNUM('20')
);
```

Sample Use Cases:

- Disease prediction.
- Recidivism prediction.
- Graduation prediction.

Sample Verticals:

- Healthcare
- Government (crime research)
- Education

Minhash

Function Name: minhash

Minhash estimates the similarity of two or more clusters. The MinHash scheme is an instance of locality sensitive hashing -- a collection of techniques for using hash functions to map large sets of objects down to smaller hash values. The result is that when two objects are close to each other, their hash values are likely to be the same. Minhash is applied to analyze transaction data and identify clusters of similar items frequently appearing in a transaction. The same transaction data can also be analyzed to generate clusters of similar users based on the purchased items.

Minhash is a derivation of associative rule learning methodology, which discovers interesting relations between variables in large datasets and relationship strengths are measured from strong to weak. The Minhash function was originally developed by the Alta Vista search engine to remove duplicate Web pages from search results. Minhashing generates a signature for items. An item could be a Web page, keyword, or a document name. The more similar two items are, the higher the chance that they share the same Minhash. Now, we need a definition for *similarity* between two items or more. Similarity is calculated by using the Jaccard Similarity formula. The formula, for example, that computes the similarity and therefore assigns the Minhash for the automobile parts and hardware sections is calculated as a ratio and the value can fall between 0 (no similarity or items in common) and 1 (identical item sets):

**Similarity [Items in the Automobile Parts Section [A], Hardware Section [H]] =
Total number of common items between [A] and [B] / Total number of unique items in both departments.**

TERADATA ASTER DISCOVERY PORTFOLIO

Another early implementation was in point-of-sale (POS) systems in supermarkets where items in a basket could be paired with similar items (e.g., onions and potatoes paired with hamburger meat and ketchup). Such information can be used as the basis for decisions about marketing activities such as promotional pricing or product placements. Other applications include image processing, security intrusion detection, and bioinformatics.

Function Call:

```
SELECT *
FROM minhash (
  ON (SELECT 1)
  PARTITION BY 1
  DATABASE('***')
  USERID('***')
  PASSWORD('***')
  INPUTTABLE('minhash_test')
  OUTPUTTABLE('minhashoutput')
  IDCOLUMN('user_id')
  ITEMSCOLUMN('items')
  NUMHASHFUNCTIONS('1002')
  KEYGROUPS('3')
  HASH_INPUT_TYPE('integer')
  MINCLUSTERSIZE('3')
  MAXCLUSTERSIZE('5')
);
```

Sample Use Cases:

- De-duplicating documents so that user Web searches or other document retrieval actions enable quick querying.
- Recognizing images from fuzzy copies and matching them to existing data stores to identify individuals. This could be used for crime prevention, locating individuals, or comparing works of art.
- Recognizing brand new information that may be similar with existing information but which contains large amounts of differences. This enables enterprises to add to their data base and powers better research.

Sample Verticals:

- Bioinformatics
- Government (national security)
- Scientific research (including healthcare and pharmaceuticals)
- Education

Naïve Bayes Classifier

Function Names: naiveBayesReduce, naiveBayesMap, naiveBayesPredict

The naive Bayes classifier is probabilistic classifier based on Bayes' theorem and advocates a very strong (naïve) assumption of the independence of each attribute that contributes towards an outcome. For example, a fruit may be considered to be an apple if it is red, round, and about four inches in diameter. A naive Bayes classifier considers all of these properties to independently contribute to the probability that this fruit is an apple.

There are two main components to the naive Bayes model:

- *Bayes' Theorem* -- A classical law, stating that the probability of an outcome is proportional to the probability of observing the data given the outcome, times the prior probability of the outcome.
- *The "naive" probability model* -- This model assumes that the input data are independent of one another and

TERADATA ASTER DISCOVERY PORTFOLIO

conditional on the outcome. This is a very strong assumption, and never true in real life, but it makes computation of all model parameters simple, and violating the assumption does not hurt the model much.

Let's illustrate a naïve Bayes' application in the figure below by calculating the likelihood of playing golf in sunny weather. The probability can be calculated by first constructing a frequency table for each attribute (weather outlook, times played for each outlook type). Then, the frequencies are transformed into likelihoods, and then the naive Bayesian equation calculates the probability for each class. The class with the highest probability is the likely outcome.

Classification algorithms such as this are important because they save as cheap and efficient heuristics to assess outcome probabilities without having to engage in complex calculations over time. The classification approach is strictly data driven and devoid of human biases. It is important to have a robust training data set to test, validate, and make corrections to the model. The validity and reliability of the model is enhanced as the training data set becomes more representative.

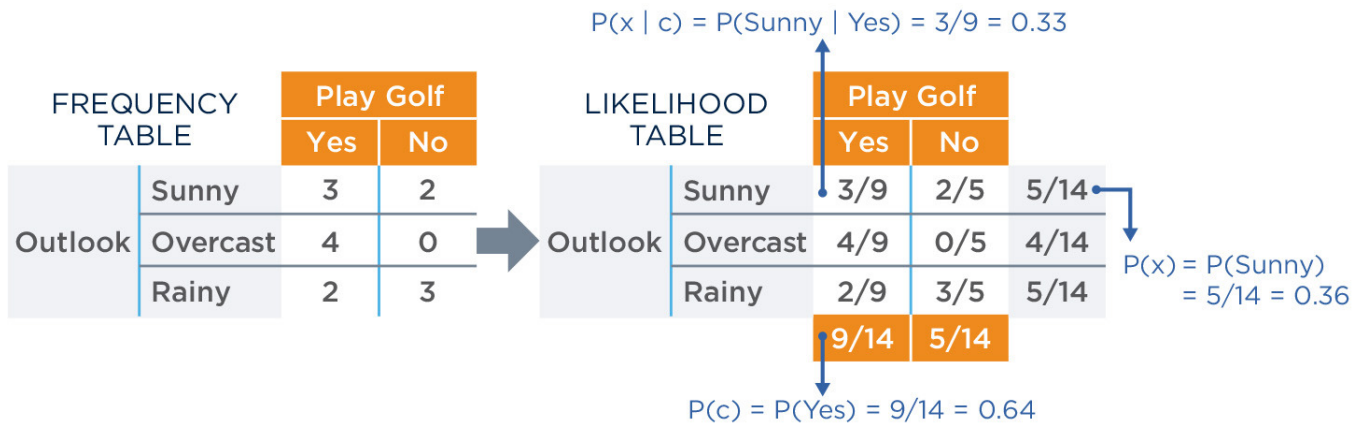


Figure 25: Naive Bayes likelihood computation

In the above example, the probability of playing golf when the weather is sunny is ~60% $((.64 * .33) / .36)$

Function Call #1: Creating the model based on training data.

```
CREATE TABLE nb_stolencars_model (PARTITION KEY(class)) AS
SELECT * FROM naiveBayesReduce(
  ON(
    SELECT * FROM naiveBayesMap(
      ON nb_sample_stolenCars
      RESPONSE('stolen')
      NUMERICINPUTS('year')
      CATEGORICALINPUTS('[2:4]')
    )
  )
  PARTITION BY class
);
```

Function Call #2: Implementing the model on new data.

```
SELECT * FROM naiveBayesPredict(
  ON nb_test_stolenCars
  PASSWORD('***')
  MODEL('nb_stolencars_model')
  IDCOL('id')
  NUMERICINPUTS('year')
  CATEGORICALINPUTS('[2:4]')
```

```
)  
ORDER BY test_id;
```

Sample Use Cases:

- Predicting the likelihood of customer churn, retention, renewals, and membership upgrades.
- Classifying items or individuals into unique segments for specific treatments (e.g., Healthcare).
- Detecting spam data or malicious digital properties that compromise user security and privacy.
- Identifying viewers' tastes and preferences when specific advertising campaigns (e.g., political advertisements) are created and/or planned. This is called digital behavioral advertising.
- Creating an integrated health analytics platform for the government where disparate sources of data are used to understand gaps in treatment, outbreaks of deadly diseases, reducing costs, improving treatment metrics, and increase savings to programs such as Medicare, Medicaid, and military health.

Sample Verticals:

- Government (cyber fraud,
- Healthcare
- Manufacturing
- Electoral analysis
- Insurance (risk)
- Natural sciences
- Energy

Principal Component Analysis (PCA)

Function Names: pca_reduce, pca_map

Principal component analysis (PCA) is a variable reduction procedure. This function performs an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called **principal components**. PCA is appropriate when you have obtained measures on a number of observed variables and wish to develop a smaller number of artificial variables that will account for most of the variance in the observed variables. The principal components may then be used as predictor variables in subsequent analyses. The number of principal components is less than or equal to the number of original variables. The first principal component has the largest possible variance and each succeeding component in turn has the next highest variance while also being uncorrelated with principal preceding components. The advantage of PCA is that only the most important variables are retained for model creation. The other advantage of PCA can be seen by its application on data with an unusually large number of dimensions (e.g., age, race, geography, gender, time).

Let's take the following employee questionnaire as an example:

Question 1: I have a lot of flexibility in my current job (scale of 1-10 with 10 indicating full agreement)
Question 2: I get to make many important decisions in my current job (scale of 1-10 with 10 indicating full agreement)
Question 3: I attend many important meetings in my current job (scale of 1-10 with 10 indicating full agreement)
Question 4: My pay level is consistent with market expectations (scale of 1-10 with 10 indicating full agreement)
Question 5: My team is constantly growing with new employees added consistently (scale of 1-10 with 10 indicating full agreement)

Figure 26: Employee Questionnaire

Now, let's suppose that this questionnaire (presumably containing more questions not shown here) is administered to 1000 employees to measure job satisfaction, churn likelihood, talent depth etc. If the questionnaire is administered as is, the sheer

TERADATA ASTER DISCOVERY PORTFOLIO

volume of data will be large and inefficiencies are likely seen because more than one question measures the same construct. The total number of questions can be reduced to principal components by first doing a simple correlation analysis (see below).

	Question 1	Question 2	Question 3	Question 4	Question 5
Question 1	1				
Question 2	0.92	1			
Question 3	0.87	0.91	1		
Question 4	0.84	0.86	0.84	1	
Question 5	0.08	0.09	0.11	0.06	1

Table 68: Correlation Table

As can be seen, questions 1 through 4 are all highly correlated with each other and show a very low correlation with question 5. This indicates that questions 1 through 4 could be substituted by 1 question and the total number of questions in this part of the questionnaire shown in the above figure can be reduced to two questions (principal components) – one component measures job satisfaction and the other component could measure team strength. Similarly, more components can be added and each component will contain a minimum set of questions that are orthogonal to each other and contribute to the overall measurement of the component it belongs to.

As a caveat, it must be mentioned that PCA does not actually reduce the number of components. So, in the above example, with 5 questions, PCA will not “reduce” the number of questions to 2. PCA will extract all 5 questions as principal components but will indicate that the first two components are the ones that contribute mostly to the variance. The rest of the questions will contribute minimally or in other words, will have negligible impact on the outcome variable.

Function Call:

```
SELECT * FROM pca_reduce (
  ON (
    SELECT * FROM pca_map (
      ON <target_table>
      [ TARGET_COLUMNS( <target_columns> ) ]
    )
  )
  PARTITION BY 1
  [ COMPONENTS( <num_components> ) ]
) ORDER BY component_rank;
```

Sample Use Cases:

- Image compression.
- Outlier recognition.

Sample Verticals:

- Data research projects across various verticals.

Percentile (Actual and Approximate)

Function Names: percentile, approx._percentile

TERADATA ASTER DISCOVERY PORTFOLIO

A **percentile** is a measure that indicates the value below which a given percentage of observations in a group of observations fall. For example, the 20th percentile is the value (or score) below which 20 percent of the observations may be found. The term percentile and the related term percentile rank are often used in the reporting of scores from norm-referenced tests. For example, if a score is in the 86th percentile, it is higher than 86% of the other scores.

An **Approximate Percentile** computes approximate percentiles for one or more columns of data where the accuracy of the approximation is a parameter specified by the user.

The following illustration shows that you are in the 8^{0th} percentile of all people based on height.

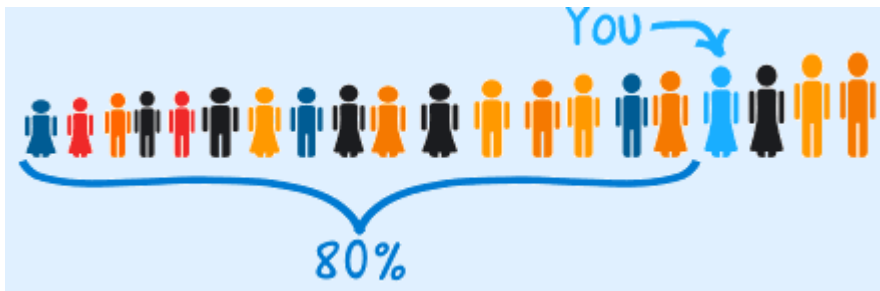


Figure 27: Percentile Illustration

Sample Use Cases:

- Any use case where there is a need for summary statistics.

Sample Verticals:

- Any vertical where there is a need for summary statistics.

Random Forest

Function Names: forest_drive, forest_predict, forest_analyze

Random forests are generated by decision trees with each tree building on one or more decisions to form a final outcome. A decision tree is a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. Decision trees, which include predictive capabilities, are used to resolve classification problems. Random forests are derived from the *ensemble learning* construct. Ensemble learning occurs when more than one model (tree) is used to predict an outcome as opposed to using a single model. In this approach, many decision trees are constructed at training time, and the class (outcome) that is the mode of the classes output by each individual tree is registered as the final class (outcome). Some of the modern applications of random forest decision trees include credit approval, target marketing, medical diagnosis, and medical treatment effectiveness analysis. The following illustration is a basic example of a random forest implementation where two decision trees are shown. These trees analyze the data about heart disease and make predictions of outcomes based on the conditional occurrences of various attributes.

The decision tree algorithm that generates the visual shown in the figure below depicts an attribute to test at each decision node in the tree (e.g., is the age >50 or <50, if >50 is the gender =female; if <50 is gender = male). The goal is to select the attribute (e.g., age, gender, color, expected revenue) that is most useful for classifying examples. The goodness of an attribute is measured by the construct *information gain* which indicates how its set of values significantly influences the direction of decision paths.

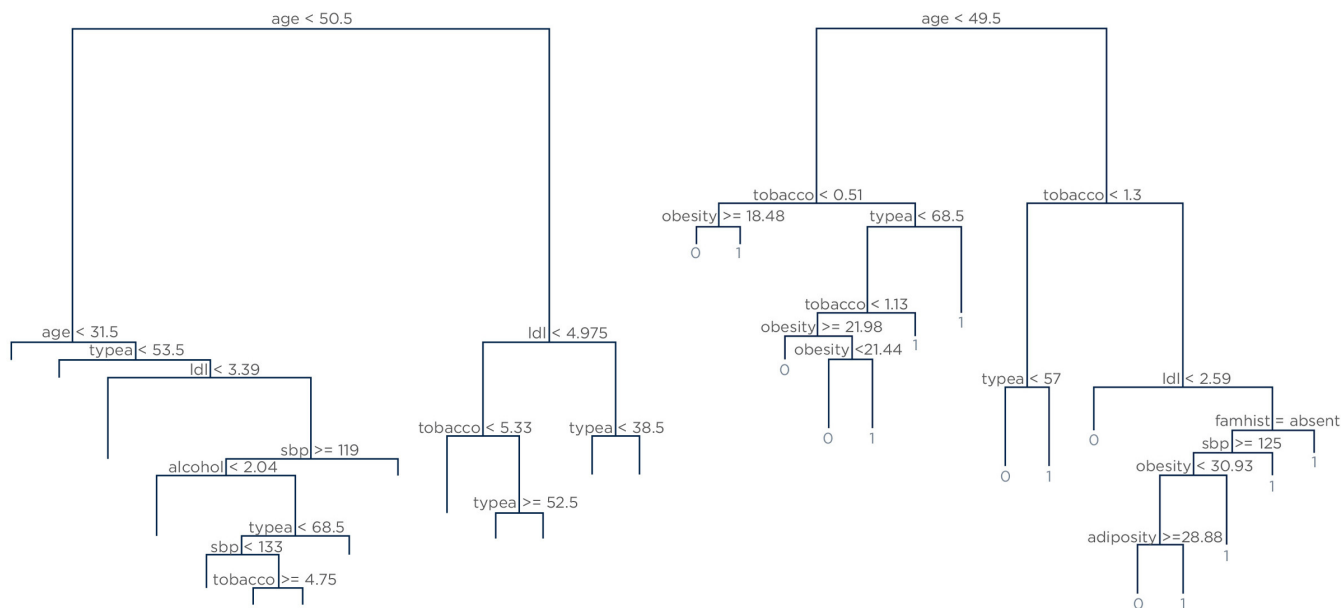


Figure 28: Random Forest illustration

Function Call #1: Creating the model from training data.

```

SELECT * FROM forest_drive(
  ON (SELECT 1)
  PARTITION BY 1

  DATABASE('***')
  USERID('***')
  PASSWORD('***')

  INPUTTABLE('wikilogs')
  OUTPUTTABLE('wikilogs_forest')

  RESPONSE('pageviews')
  NUMERICINPUTS('bytes','len_name')
  CATEGORICALINPUTS('hour','projectcode')
);
    
```

Function Call #2: Implementing the model on new data.

```

CREATE TABLE predictions (PARTITION KEY(pagename)) AS
SELECT *
FROM forest_predict
( ON one_day
  DATABASE('***')
  USERID('***')
  PASSWORD('***')
  FOREST('wikilogs_forest')
  NUMERICINPUTS('bytes','len_name')
  CATEGORICALINPUTS('hour','projectcode')
  idcol('pagename')
);
    
```

Function Call #3: Analyzes the structure of the model

TERADATA ASTER DISCOVERY PORTFOLIO

```
SELECT * FROM forest_analyze(
  ON {table_name|view_name|{query}}
  [NUM_LEVELS(<number_of_levels>)]
)
```

Sample Use Cases:

- Making revenue maximizing choices when faced with an array of options. The probability of each option needs to be calculated and decisions need to be forecast far in advance. Manufacturing organizations, for example, need to forecast inventory storage by predicting seasonal demand and estimating the impact of specific product capabilities on demand.
- Accurately predicting travel times during rush hours. Transportation departments regularly struggle with providing commuters service that minimizes their commute time. A decision tree approach that looks at various route choices conditional upon the attribute values for time or speed of traffic allows them to manage traffic as well as plan for additional infrastructure.
- Enhancing air quality predictions by including various environmental and made-made variables to make predictions on an hourly basis. For example, if one condition is pollen count, then depending on the counts, patients with particular allergies can stay indoors. Also, patients afflicted by significant respiratory illnesses and their medical care providers use these indices regularly to reduce exposure to harmful industrial pollutants.

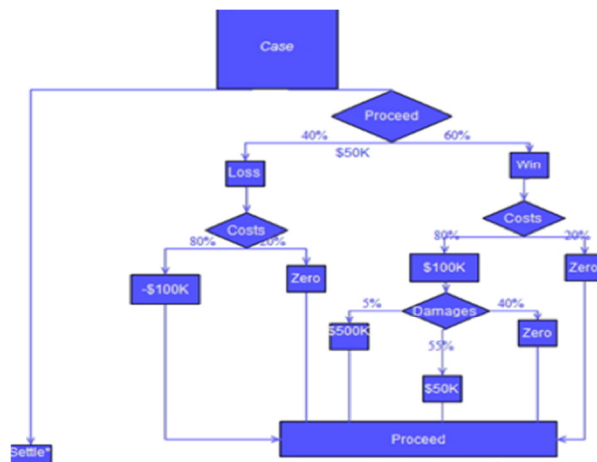
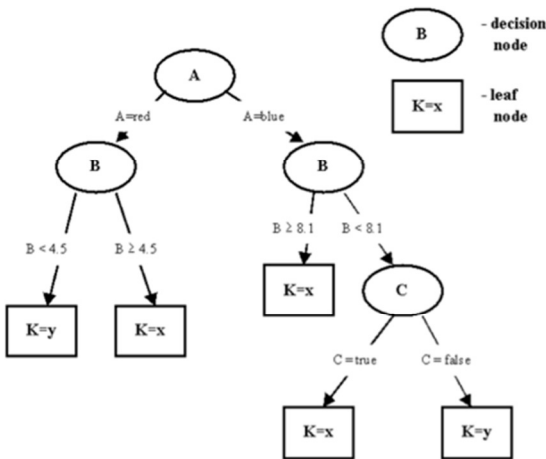
Sample Verticals:

- Manufacturing
- Transportation
- Smart phone/tablet applications
- Agriculture

Single Decision Tree

Function Names: single_tree_drive, single_tree_predict

A single decision tree, a variant of the Random Forest function described above, creates a single tree structure, as opposed to a forest of trees, and splits the decisions and data into branch like segments. The two figures below illustrate a single decision tree that consists of two paths that lead to sub paths where outcomes are seen. Each outcome is based on a decision taken on the assessment of an attribute's value (e.g., color, true/false, value above or below a threshold, demographics). The flow chart representation is the likelihood of a decision and its expected value.



Figures 29 and 30: Single Decision Tree illustrations

TERADATA ASTER DISCOVERY PORTFOLIO

The advantage of a single decision tree is that it maps out all possible paths that can be taken for a given use case and each path is associated with an expected value. Expected values reflect the risk levels for each path and allow analysts to separate the low risk paths (lower payoff) from higher risk (higher payoff) alternatives. Once the risk assessment is made a course of action where risk and payoff expectations fall within a certain acceptable threshold is chosen.

Function Call #1:

```
SELECT * FROM single_tree_drive(  
  ON (SELECT 1)  
  PARTITION BY 1  
  ATTRIBUTETABLENAME('glass_attribute_table')  
  OUTPUTTABLENAME('glass_attribute_table_output')  
  MATERIALIZESPLITSTABLEWITHNAME('splits_small')  
  RESPONSETABLENAME('glass_response_table')  
  NUMSPLITS('3')  
  IMPURITYMEASUREMENT('gini')  
  MAX_DEPTH('10')  
  IDCOLUMNS('pid')  
  ATTRIBUTENAMECOLUMNS('attribute')  
  ATTRIBUTEVALUECOLUMN('attrvalue')  
  RESPONSECOLUMN('response')  
  MINNODESIZE('3')  
  PASSWORD('***')  
  USERID('***')  
  DATABASE('***')  
  USEAPPROXIMATESPLITS('false'));
```

Function Call #2: Implements the single tree decision model to new data predicts likely outcomes.

```
SELECT * FROM single_tree_predict (  
  ON glass_attribute_table as attribute_table partition by pid order by attribute  
  ON glass_attribute_table_output as model_table DIMENSION  
  ATTRTABLE_GROUPBYCOLUMNS('attribute')  
  AttrTable_pidColumns('pid')  
  AttrTable_valColumn('attrvalue')  
  ModelTable_nodeColumn('node_id')  
  ModelTable_sizeColumn('node_size')  
  ModelTable_leftSizeColumn('left_size')  
  ModelTable_rightSizeColumn('right_size')  
  ModelTable_attrColumns('attribute')  
  ModelTable_splitColumn('split_value')  
  ModelTable_labelColumn('node_label')  
  ModelTable_leftLabelColumn('left_label')  
  ModelTable_rightLabelColumn('right_label'));
```

Sample Use Cases:

- Verifying current models of outcome prediction by matching them with the actual classes in which such outcomes land. For example, a person who is 60 years old, and has smoked for more than 25 years is likely to be classified as a high-risk patient based on longitudinal data. Decision trees help classify new patients, and these models can be refined by model verification.
- Enabling critical decisions based on an association of each outcome with a probability of occurrence that would allow businesses to evaluate the value of an outcome against the risk it needs to assume to achieve it. For example, enterprises routinely make decisions around whether to acquire new companies or invest in internal R&D by making value assessments of these decisions.

Sample Verticals:

- Manufacturing
- Scientific research

Support Vector Machines (SVM)

Function Names: SparseSVMTrainer and SparseSVMPredictor

Support vector machines (SVM) are a class of machine learning algorithms. Machine learning, in general, derives a structure from data. SVM, in particular, is used to classify (structure) data into only two categories. Each category is separated from the other by a boundary that is determined by this function. Unlike probabilistic models of classification, this function does not estimate likelihoods but focuses on creating the maximum possible distance between two clusters of data.

SVMs are based on the concept of decision planes that define decision boundaries. A decision plane is a separator between a set of objects having different class memberships. In the illustration below, the objects belong either to class GREEN or RED. The separating line defines a boundary on the right side of which all objects are GREEN and to the left of which all objects are RED. Any new object (white circle) falling to the right is labeled, i.e., classified, as GREEN (or classified as RED should it fall to the left of the separating line).

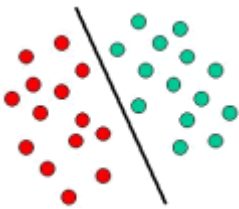


Figure 29: Linear Classifier

The above is a classic example of a linear classifier, i.e., a classifier that separates a set of objects into their respective groups (GREEN and RED in this case) with a line. Most classification tasks, however, are not that simple, and often more complex structures are needed in order to make an optimal separation, i.e., correctly classify new objects (test cases) on the basis of the examples that are available (train cases). This situation is depicted in the illustration below. Compared to the previous schematic, it is clear that a full separation of the GREEN and RED objects would require a curve (which is more complex than a line). Classification tasks based on drawing separating lines to distinguish between objects of different class memberships are known as hyperplane classifiers. Support Vector Machines are particularly suited to handle such tasks.

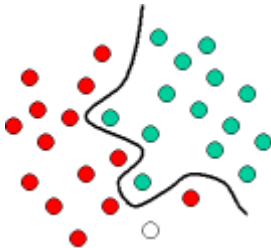


Figure 30: Hyperplane Classifier

The illustration below shows the basic idea behind Support Vector Machines. Here we see the original objects (left side of the schematic) mapped, i.e., rearranged, using a set of mathematical functions, known as kernels. The process of rearranging the objects is known as mapping (transformation). Note that in this new setting, the mapped objects (right side of the schematic) is linearly separable and, thus, instead of constructing the complex curve (left schematic), all we have to do is to find an optimal line that can separate the GREEN and the RED objects.

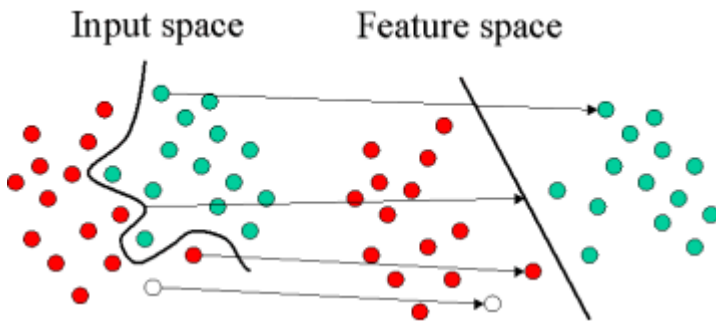


Figure 31: Support Vector Machines Implementation

Function Call #1: SparseSVMTrainer

```
select * from SparseSVMTrainer(
on (select 1) partition by 1
inputtable('news20_train_tfidf')
modeltable('news20_model')
sampleidcolumn('id')
attributecolumn('token')
valuecolumn('value')
labelcolumn('label')
maxstep(20)
database('***')
password('***')
);
```

Function Call #1: SparseSVMPredictor

```
select * from SparseSVMPredictor(
on news20_test_tfidf as input partition by id
on news20_model as model dimension
sampleidcolumn('id')
attributecolumn('token')
valuecolumn('value')
accumulatelabel('label')
);
```

Sample Use Cases:

- New treatment identification.
- Image recognition.

Sample Verticals:

- National Security
- Medical Research

Text Analytics and Sentiment Extraction Functions

Chinese Text (Word) Segmentation

Function Name: texttokenizer

This function identifies the sequence of words in Chinese language (*hanzi*) text. In English and other languages that use an alphabet, word boundaries are white spaces. In Chinese, a word may be one, two, or more adjacent characters; the Chinese Text Segmentation function finds the word boundaries and returns the words in the text. Implementing this function is a necessary first step in Chinese language processing.

Function Call:

```
select * from textTokenizer(
  on cn_input partition by any
  on cn_dict as dict dimension
  language('zh_CN')
  outputDelimiter(' ')
  outputByWord('false')
  accumulate('id')
  textColumn('txt')
);
```

Sample Use Cases:

- Processing Chinese Text

Sample Verticals:

- All verticals that require Chinese language processing

Latent Dirichlet Allocation (LDA)

Function Names: LDATrainer, LDAInference

The LDA function creates something called a topic model. In machine learning and natural language processing, a topic model is a type of statistical model for discovering the abstract "topics" that occur in a collection of documents. Intuitively, given that a document is about a particular topic, one would expect particular words to appear in the document more or less frequently: "tympani", "drums", and "snare" are likely to appear more often in documents about percussion instruments, "lions" and "cats" will appear in documents about felines, and "the" and "is" will appear equally in both. A document typically concerns multiple topics in different proportions; thus, in a document that is 10% about cats and 90% about music, there would probably be about 9 times more musical instrument words than, say, words about animals. A topic model, such as the one created by LDA captures this intuition in a mathematical framework, which allows examining a set of documents and discovering, based on the statistics of the words in each, what the topics might be and what each document's balance of topics is.

Another illustration of this follows in the following sentences:

- I like to eat broccoli and bananas.
- I ate a banana and spinach smoothie for breakfast.
- Chinchillas and kittens are cute.
- My sister adopted a kitten yesterday.
- Look at this cute hamster munching on a piece of broccoli.

When applied, LDA automatically discovers topics that these sentences contain. These topics would be categorized as follows:

TERADATA ASTER DISCOVERY PORTFOLIO

- Sentences 1 and 2: 100% Topic A (Food)
- Sentences 3 and 4: 100% Topic B (Animals)
- Sentence 5: 60% Topic A, 40% Topic B

A real world application of LDA on a set of political e-mails revealed the following categories:

- Family/Inspiration: family, web, mail, god, son, from, congratulations, children, life, child, baby, birth, love, you, syndrome, very, special, bless, old, husband, years, thank, best,
- Wildlife/BP Corrosion: game, fish, moose, wildlife, hunting, bears, polar, bear, subsistence, management, area, board, hunt, wolves, control, department, year, use, wolf, habitat, hunters, caribou, program, denby, fishing,
- Energy/Fuel/Oil/Mining: energy, fuel, costs, oil, alaskans, prices, cost, nome, now, high, being, home, public, power, mine, crisis, price, resource, need, community, fairbanks, rebate, use, mining, villages
- Gas: gas, oil, pipeline, agia, project, natural, north, producers, companies, tax, company, energy, development, slope, production, resources, line, gasline, transcanada, said, billion, plan, administration, million, industry
- Education/Waste: school, waste, education, students, schools, million, read, email, market, policy, student, year, high, news, states, program, first, report, business, management, bulletin, information, reports, 2008, quarter
- Presidential Campaign/Elections: mail, web, from, thank, you, box, very, good, great, hope, president, sincerely, work, keep, make, add, family, party, support, doing, p.o

In the above example, the categories shown above are not created by the function and were instead created manually for clarity.

Function Call: Idatrainer

```
select * from Idatrainer(
on (select 1) PARTITION by 1
inputtable('ap_input_table')
modeltable('lda_model_table')
outputtable('lda_ap_output>')]
topicnumber(10)
alpha(0.1)
docidcolumn('docid')
wordcolumn('word')
outputtopicnumber('all')
)
```

Function Call: Idainference

```
select * from Idainference(
on (select 1) PARTITION by 1
inputtable('ap_input_table')
modeltable('lda_model_table')
outputtable('lad_ap_output2')
outputtopicnumber(2)
outputtopicwordnumber(5)
)
```

Sample Use Cases:

- Assigning subject categories, topics, or genres.
- Identifying document authorship and other individual contributors.
- E-Mail and text parsing
- Collaborative filtering

Sample Verticals:

- Health Care (Medical Transcription)
- E-Commerce

Levenshtein Distance

Function Name: ldist

This function computes the number of edits (distance) needed to transform one string into the other (e.g., from 'leaf' to 'leave' or 'orange' and 'oranje'). Edits include insertions, deletions, or substitutions of individual characters. The greater the distance value, the more the latitude that the function has to make fuzzy matches. When this distance value is 0, only exact matches are allowed. This function is important because it helps match text variations due to typos, misspellings, OCR errors, differences in transliteration, and so on. These variations occur in many sources of text, especially when the data comes from manual key-entry, making this function essential to the accuracy of analytics.

Function Call:

```
SELECT *
FROM ldist
(
  ON sample_lev_input
  SOURCE('col1','col2', 'col3')
  TARGET('company')
  ACCUMULATE('company_id')
);
```

The input table for this function is:

COL 1	COL 2	COL 3	COMPANY	COMPANY ID
Astre	Astter	Astur	Aster	749

Table 69: Levenshtein Distance Input

The function, when invoked, results in:

COMPANY ID	TARGET	SOURCE	LEVENSHTEIN DISTANCE
749	Aster	Astre	2
749	Aster	Astter	1
749	Aster	Astur	1

Table 70: Levenshtein Distance Output

Where, distances can be interpreted as the degree of closeness to the word "Aster". In this case, distances of 1 are closer to the target than distances greater than 1.

Sample Use Cases:

- Fuzzy data matching.
- Image recognition.

- Text analysis.

Sample Verticals:

- Government (National Security)
- Insurance

Naïve Bayes Text Classifier

Function Names: naiveBayesText, naiveBayesTextPredict

See the Naïve Bayes Classifier function above (Statistical Functions) for a description of the importance of classification algorithms as well as the training and testing cycle embodied in these algorithms.

The text classifier algorithm classifies texts in documents and other sources into specific categories (e.g., named entity, sentiment). In this implementation, a training set of classifications is created, and all future documents are classified based on the training set. For example, the training set classification is as follows:

```
insert into test_docs values(1, 'Economics is not as good as last year.', 'finance');
insert into test_docs values(2, 'Football is popular.', 'sport');
```

The two document classifiers noted here are Finance and Sport. All future content is evaluated against this list and classified accordingly. Documents can be tagged with multiple classifiers. The naïve Bayes text classifier approach treats every document as a bag of words. The following four steps comprise this approach:

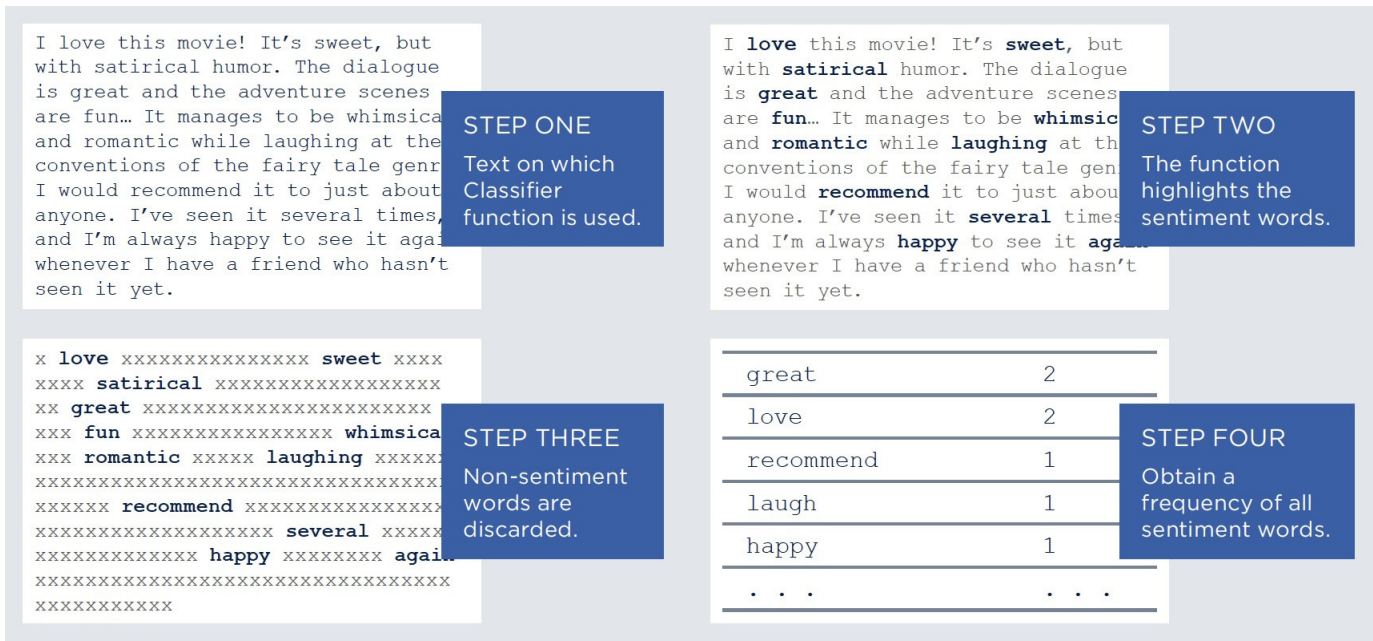


Figure 32: Sentiment Classification Process

Function Call 1: Creating the model based on training data.

```
CREATE TABLE complaints_nb_model (PARTITION KEY(token)) AS
SELECT token, SUM(crash) AS crash, SUM(no_crash) AS no_crash FROM naiveBayesText(
ON complaints
TEXT_COLUMN ( 'text_data' )
CATEGORY_COLUMN ( 'category' )
```

```
CATEGORIES ( 'crash', 'no_crash' )
)
GROUP BY token;
```

Function Call 2: Using the model to make predictions on new data.

```
SELECT * FROM naiveBayesTextPredict(
  ON complaints_test
  DOCUMENT_ID ( 'doc_id' )
  TEXT_COLUMN ( 'text_data' )
  CATEGORIES ( 'crash', 'no_crash' )
  MODEL_FILE ( 'complaints_nb_model.csv' )
);
```

Sample Use Cases:

- Assigning subject categories, topics, or genres.
- Identifying document authorship and other individual contributors.
- Identifying age, gender, and other demographics.
- Identifying languages used in the document.
- Isolating specific sentiments, reactions, and reviews that can be used as input into decision making (e.g., restaurants changing the menu based on negative customer feedback to the options provided earlier).

Sample Verticals:

- Health Care (Medical Transcription)
- E-Commerce (Call Center Analysis)

Named Entity Recognition (NER)

Function Names: FindNamedEntity, TrainNamedEntityFinder, NameFindEvalPartition

This function finds mentions of specified entities in text. For example, a simple NER for English might find the person mention *Joe J. Peters* and the location *London* in the text “Joe J. Peters lives in London and loves football.” The NER function includes three SQL-MR functions:

- Find Named Entity -- Extracts all the specified name entities from the input document by statistical models, regular expressions or a dictionary.
- Train Named Entity Finder -- Takes a training data set, and generates predicted classifiers on new data.
- Evaluate Named Entity Finder Partition/Evaluate Named Entity Finder Row -- Takes a set of data and generates the precision, recall, and f-measure values that is used to evaluate the classification model. The format for the input labeled data is the same as is used for training. The best outcome of the evaluation will be an exact label match to the training data labels. In other words, when new data is given a label (e.g., positive sentiment, high income) the label is compared to what the equivalent is in the training data set (based on which the model was created) to determine match accuracy.

There are four options that can be exercised in implementing this function: Rule, Dictionary, Regexp, and Statistical. In other words, entity extractions can be performed by applying one or more of these options on the data.

- (a) Rule: Any rule or heuristic that is manually created. For e.g., “((Obj <denoting organization>) (Ver) (denoting action) (Subj <denoting the geographic location>))”. A rule can thus be applied to recognize entities in a sentence.

TERADATA ASTER DISCOVERY PORTFOLIO

- (b) **Dictionary**: This is a lexicon that keeps getting added to with information as new sentences appear and is referred to periodically to identify entities (subject, object, verb etc.). As I understand it, the dictionary can also contain rules for execution.
- (c) **Regex**: This is standard parsing that can be applied to say, extracting US phone numbers or extracting data from standard web pages (e.g., product name from an Amazon URL).
- (d) **Statistical**: This is the application of machine learning types of algorithms (derived from being trained on large scale manually annotated data) to establish the likelihood of an entity in a sentence or the nature of such an entity (e.g., object, subject). These statistical models can be created on training data sets and then as new data arrive they can be applied and refined.

The choice of each option depends on the nature of the data. For example, if there are standard web pages with a pre-defined structure that are generated then the regexp option is used for parsing. On the other hand, if there is a large amount of data then a statistical approach is called for to extract information.

Function Call: FindNamedEntity

```
SELECT *
FROM FindNamedEntity(
  ON mydocs
  text_column('content')
  model('all')
  accumulate('id', 'src')
  SHOW_POSITION(3)
);
```

Function Call: TrainNamedEntityFinder

```
SELECT *
FROM TrainNamedEntityFinder(
  ON myTraining
  PARTITION BY 1
  type('location')
  text_column('content')
  model_file('location.dataset2')
  DOMAIN('153.65.197.90')
  DATABASE('sqlmr_test')
  USERID('db_superuser')
  PASSWORD('db_superuser')
);
```

Function Call: NameFindEvalPartition

```
SELECT * FROM NameFindEvalPartition(
  ON NameFindEvalRow
  (
    ON {table_name|view_name|(query)}
    TEXT_COLUMN('<text_column_name>')
    MODEL_FILE('<model_data_file>')
  )
  PARTITION BY 1
);
```

Sample Use Cases:

- Event information extraction from chat logs or e-mails.

- Customer complaint management.

Sample Verticals:

- Call center management
- Banking

nGram

Function Name:nGram

The nGram model is part of a class of models that purports to understand how a language works. In other words, nGrams are one way by which the rules that govern a language can be understood. nGram models can be imagined as placing a small window over a sentence or a text, in which only n words are visible at the same time. nGrams are thus a set of consecutive words that appear in a text. The value of an nGram is that they capture words and larger structures (phrases and names) that have meaning. nGrams can be matched (exactly or fuzzily) with other nGrams or with other sources (dictionaries, ontologies, etc.) and trends in their occurrence over time could be captured.

The n -gram model is a type of probabilistic language model for predicting the next item in such a sequence and is widely used in natural language processing, biological sequence analysis and data compression. The two core advantages of n -gram models (and algorithms that use them) are relative simplicity and the ability to scale up – by simply increasing n , a model can be used to store more context. This function is useful for performing sentiment analysis, topic identification, and classification. The n -grams typically are collected from a text or speech corpus.

Function Call:

```
SELECT *
FROM nGram
(
  ON my_docs
  TEXT_COLUMN('txt')
  DELIMITER(' ')
  GRAMS(2)
  OVERLAPPING('true')
  CASE_INSENSITIVE('true')
  PUNCTUATION('[.,?!\\]')
  RESET('[.,?!\\]')
  ACCUMULATE('id','src')
);
```

The input table for this function is:

ID	SOURCE	TEXT
1	wikipedia	the Quick brown fox jumps over the lazy dog
2	sampledoc	hello world. again, I say hello world

Table 71: nGram Input

The function, when invoked, results in:

ID	SOURCE	NGRAM	FREQUENCY
1	wikipedia	the quick	1
1	wikipedia	quick brown	1
1	wikipedia	brown fox	1
1	wikipedia	fox jumps	1
1	wikipedia	jumps over	1
1	wikipedia	over the	1
1	wikipedia	the lazy	1
1	wikipedia	lazy dog	1
2	sampledoc	hello world	2
2	sampledoc	again I	1
2	sampledoc	I say	1
2	sampledoc	say hello	1

Table 72: nGram Output

Sample Use Cases:

- Sentiment extraction.
- Reputation management.

Sample Verticals:

- Social media management
- E-Commerce
- Media

Sentiment Extraction

Function Names: ExtractSentiment, EvaluateSentimentExtractor, TrainSentimentExtractor

This function deduces an opinion (positive, negative, neutral) from text-based content. The sentiment extraction functions include three SQL-MR functions:

- Extract Sentiment -- Extracts opinion from text and categorizes them into distinct buckets (e.g., happy).
- Evaluate Sentiment Extractor -- Evaluates the precision and recall of extract sentiment function.
- Train Sentiment Extractor -- Trains a maximum entropy classifier for sentiment analysis. A maximum entropy classifier provides the least biased estimate possible based on the given information. It makes no assumptions about missing data, and all predictions are anchored to only the data that are seen.

There are two options that can be exercised in implementing this function: Dictionary, and Statistical. In other words, entity extractions can be performed by applying one or more of these options on the data.

TERADATA ASTER DISCOVERY PORTFOLIO

- (a) **Dictionary:** This is a lexicon that keeps getting added to with information as new sentences appear and is referred to periodically to identify entities (subject, object, verb etc.). As I understand it, the dictionary can also contain rules for execution.
- (b) **Statistical:** This is the application of machine learning types of algorithms (derived from being trained on large scale manually annotated data) to establish the likelihood of an entity in a sentence or the nature of such an entity (e.g., object, subject). These statistical models can be created on training data sets and then as new data arrive they can be applied and refined.

Function Call #1: ExtractSentiment

```
select *
from ExtractSentiment
(
  on kindleView
  text_column('content')
  model('dictionary')
  level('document')
);
```

Function Call #2: EvaluateSentimentExtractor

```
select *
from EvaluateSentimentExtractor
(
  on ExtractSentiment
  (
    on pos_train
    text_column('content')
    accumulate('category')
    model('dictionary')
  )
  PARTITION BY 1
  expect_column('category')
  result_column('out_polarity')
);
```

The output of this function will be metrics such as recall and precision.

Function Call #3: TrainSentimentExtractor

```
SELECT *
FROM TrainSentimentExtractor(
  ON (select * from pos_train where mod(id, 2)=0)
  PARTITION BY 1
  text_column('content')
  sentiment_column('category')
  model_file('model1.bin')
  DATABASE('***')
  USERID('***')
  PASSWORD('***')
);
```

Sample Use Cases:

- Streamlining the creation of products and services to focus solely on where the majority demand comes from.
- Accelerating the identification of events that trigger negative brand perceptions (e.g., product recalls).

- Reducing the time from incident detection to resolution and proactively planning for likely scenarios.

Sample Verticals:

- Branding and advertising
- Consumer products
- Government (service providers, security)
- e-Commerce.

Attensity® Text and Sentiment Analytics Functions

Function Names: AsasNameFinder, AsasCategorization, AsasTripleFinder

Teradata Aster Database's sentiment analytics functions are both natively developed as well as developed in a partnership with Attensity. These functions enable us to exhaustively extract the facts, context and causalities contained in freeform text -- everything from e-mails, Web forums and blogs, field reports, customer surveys, cables, twitter feeds, Facebook preferences, RSS feeds and a long list of other documents that have been difficult to analyze due to inadequate technology. Attensity's strength lies in its knowledge based approach where text and sentiment extraction rules are hand built and are as nuanced as the data they analyze. These functions isolate discrete data and insert them into relational tables (structured data). These parsed data (e.g., good, delighted) are joined with other structured data to obtain a 360-degree view of customer behavior.

There are three functions that are provided:

- *Name Finder* -- Extracts named entities such as location, person's name, and date.
- *Categorization and Sentiment Analysis* -- Identifies the categories (one or more) to which a document belongs to (Positive, Negative, and Neutral).
- *Triple Finder* -- Extracts triples (source, link and target) from a sentence. Triples play a significant role in semantic Web technologies and question/answer systems.

The Attensity functions use machine learning techniques to extract data. Machine learning relies on a computer's ability to automatically learn the language used for expressing sentiment regardless of how *good* or *normal* the language is. NLP has three advantages: focus on the most common use cases; statistical inference procedures are used to classify hitherto unseen or erroneous data; models can be made more accurate by supplying more feedback as opposed to manual processes where complexity is difficult to manage with more inputs.

Function Call #1: Extracting Names

```
SELECT * FROM AsasNameFinder(
  ON {table_name | view_name | (query)}
  [Accumulate('column1' [, 'column2', ...])]
  InputColumn('text_column_name')
  [InputLanguage('english | other')]
  Model('model_file_name1:domain [; username, password ]') // file name without directory
  AsasLibraries('installed_native_library_zip_file_name')
  OutputFormat('annotations: column_name, ... | 'xml:[output_column_name]' | 'models') );
```

Function Call #2: Categorizing Sentiments

```
SELECT * FROM AsasCategorization
( on {table_name|view_name|query}
  INPUTCOLUMN('<text_column_name>')
  MODEL('<model_file_name>:<domain_name1>[; username, password]')
```

```
[USECATEGORIES('<category1>[',<category2>'])]
[INPUTLANGUAGE('english|other')]
[USEANNOTATIONTYPES('<AnnotationType>[',<AnnotationType>'])]
[OUTPUTCATEGORYCOLUMNNAME('<output_category_column_name>')]
[ACCUMULATE('<column_name>[',<column_names>'])]
[OUTPUTFORMAT('models')]
[ANALYSISLEVEL('document|phrase')]
[ASASLIBRARIES('<asasLibrary>')];
```

Function Call #3: Triple Finder

```
SELECT * FROM AsasTripleFinder(
ON {table_name | view_name | (query)}
[Accumulate('column1' [, 'column2', ...])]
InputColumn('text_column_name')
[InputLanguage(['english | other'])]
[UseCategories(['category_name1' [, 'category_name2', ...]])]
Model('model_file_name1: domain [; username, password]') // file name without directory
AsasLibraries('installed_native_library_zip_file_name')
OutputFormat('annotations: column_name, ... | 'xml:[output_column_name]' | 'models'));
```

Sample Use Cases:

- Streamlining the creation of products and services to focus solely on where the majority demand comes from.
- Accelerating the identification of events that trigger negative brand perceptions (e.g., product recalls).
- Reducing the time from incident detection to resolution and proactively planning for likely scenarios.

Sample Verticals:

- Branding and advertising
- Government (service providers, security)
- E-Commerce

Text Categorization/Classification

Function Names: TextClassifierTrainer, TextClassifier, TextClassifierEvaluator

The text categorization/classification function puts text into specific categories based on attributes gleaned from it. These classifications could be “spam/no spam”, “HR”, “Finance” etc. A given model needs to be specified for predicting the various categories. There are three functions associated with it

Function Call#1: TextClassifierTrainer – Creates a trainer model that could be used on uncategorized data

```
SELECT *
FROM TextClassifierTrainer(
  on (select 1) PARTITION BY 1
  InputTable('inputTableName')
  textColumn('textColumnName')
  categoryColumn('catColumnName')
  modelFile('modelFileName')
  classifierType('classifierType')
  [classifierParameters(<name:value>)]
  [NlpParameters(<name:value>)]
  [FeatureSelection('DF:[min:max]')]
```



```
[DATABASE('beehive')]
[USERID('dbUserId')]
PASSWORD('dbUserPassword')
[DOMAIN('ip')]
);
```

Function Call#2: TextClassifier – Categorizes/classifies new text data.

```
SELECT *
FROM TextClassifier(
  on test_docs
  textcolumn('content')
  model('knn.bin')
);
```

This function can be invoked using the model (knn.bin) we need to specify a training data set with pre-existing categories. Examples of these categories are:

Economics is not as good as last year – categorized as Finance;
Football is popular – categorized as sport
...and so on.

As new categorizations are built upon, the training data set expands and can be used to predict future categories.

This function, when invoked, results in:

CATEGORY
finance
sport

Table 73: Text Categories

Function Call#3: TextEvaluator – Evaluates the categorizations for fit.

```
select * from TextClassifierEvaluator(
  on TextClassifier(
    on (select * from test_docs)
    textcolumn('content')
    accumulate('category')
    model('knn.bin')
  )
  PARTITION BY 1
  expectcolumn('category')
  predictcolumn('out_category')
);
```

Sample Use Cases:

- Document analysis
- Spam Filters
- Fraud detection

Sample Verticals:

- Finance
- Government (Military and National Security)
- Call Center Operations

Text Chunker

Function Name: TextChunker

Text chunking consists of dividing a text into phrases in such a way that syntactically related words become member of the same phrase. These phrases are non-overlapping which means that one word can only be a member of one chunk. For example, the sentence **He reckons the current account deficit will narrow to only # 1.8 billion in September**, can be " chunked" as follows:

[Noun Phrase He] [Verb Phrase reckons] [Noun Phrase the current account deficit] [Verb Phrase will narrow] [Prepositional Phrase to] [Noun Phrase only # 1.8 billion] [Prepositional Phrase in] [Noun Phrase September].

Chunks, in the above example, have been represented as groups of words between square brackets. A tag next to the open bracket denotes the type of the chunk.

Text Chunking is an example of a strategy that helps with breaking down difficult text into more manageable pieces. Dividing content into smaller parts helps in the identification of key words and ideas, develops the ability to paraphrase, and makes it easier for to organize and synthesize information.

Function Call:

```
select * from TextChunker(
  on PosTagger(
    on pos_input
    accumulate('id')
    textColumn('txt')
  ) partition by id order by word_sn
  wordColumn('word')
  posTagColumn('pos_tag')
);
```

Let's take the following text as an example:

Michelle Pirout , 61 years old , will join the board as a nonexecutive director Nov. 29.

If we invoke the above function on this text, the following is the result, broken up into manageable chunks:

CHUNK	CHUNK TAG (TYPE)
Michelle Pirout	NP (noun phrase)
,	O (punctuation)
61 years	NP (noun phrase)
old	ADJP (adjective)
,	O (punctuation)
will join	VP (verb)

CHUNK	CHUNK TAG (TYPE)
the board	NP (noun phrase)
as	PP (preposition)
a nonexecutive director	NP (noun phrase)
Nov. 29	NP (noun phrase)
.	O (punctuation)

Table 74: Text Chunk Output

Sample Use Cases:

- E-Mail categorizations
- Spam Filters

Sample Verticals:

- Finance
- Government (Military and National Security)

Text Parser

Function Name: TextParser

Text parser is a function that tokenizes an input stream of words, optionally stem them, and then emit the individual words and counts for the each word appearance. Tokenization is the conversion of a sequence of characters into a sequence of tokens or meaningful character strings. For example, the '+' symbol is the token for an addition operator. Text parsers are used as an initial step in collecting word (or stem) occurrences so that we can do frequency analysis, including TF-IDF (understanding how important a word is in a given document or text). Once we know how important a word is we could then determine the weight to be placed on it for future information retrieval.

Function Call:

```
SELECT * FROM text_parser(
  ON my_docs
  TEXT_COLUMN('txt')
  CASE_INSENSITIVE('true')
  PUNCTUATION('\[.,?!\\]')
  ACCUMULATE('id','src')
)
```

The input table for this function is:

ID	SOURCE	TEXT
1	wikipedia	the Quick brown fox jumps over the lazy dog
2	sampledoc	hello world. again, I say hello world

Table 75: Input Text

The function, when invoked, results in:

ID	SOURCE	TOKEN	FREQUENCY
1	wikipedia	the	2
1	wikipedia	quick	1
1	wikipedia	brown	1
1	wikipedia	fox	1
1	wikipedia	jumps	1
1	wikipedia	over	1
1	wikipedia	lazy	1
1	wikipedia	dog	1
2	sampledoc	hello	2
2	sampledoc	world	2
2	sampledoc	again	1
2	sampledoc	i	1
2	sampledoc	say	1

Table 76: Parsed Text

Sample Use Cases:

- Word count applications

Sample Verticals:

- E-Commerce
- Advertising

Time Series Analysis Functions

Wavelet Transformations (Discrete, 2D, and Inverse)

Function Name: `dwt2d`, `dwt`, `idwt`

The Discrete Wave Transformation (DWT) and the 2D Wavelet Transformation algorithms transform a wavelet by decomposing it into multiple components. A wavelet is a small wave that oscillates and decays in the time domain. A wave is an oscillating function of time or space and is periodic. In other words, wavelets are localized waves and are used for analyzing transient signals. The popularity of wavelet transform is growing because of its ability to reduce distortion in the reconstructed signal while retaining all the significant features present in the signal.

The transformation that occurs is a mathematical function that is applied to a wavelet (signal) to obtain further information that is not usually available in the raw data. A transformation function is based off a *thresholding* parameter that is set in the function to eliminate data noise. The key attribute to note is that when implementing DWT, the most prominent information in the signal appears in high amplitudes, and the less prominent information appears in very low amplitudes. Data compression can be achieved by discarding these low amplitudes. The wavelet transforms enables high compression ratios with good quality of reconstruction. In other words, the output of the transformation is a more granular representation of the prominent signals that are represented in the form of discrete panels. The typical DWT implementation on a raw signal will follow the process noted in Figure 6:



Figure 33: DWT Process Flow

There is a wide range of applications for wavelet transforms. One of the prominent applications is in fingerprint matching. Wavelet transforms are used to compress the fingerprint pictures for storage in their data bank. The ridge lines in the fingerprints appear more prominent and retain all of the data after the transformation. Wavelets also find application in speech compression, which reduces transmission time in mobile applications. They are used in denoising, edge detection, feature extraction, speech recognition, echo cancellation, real-time audio and video compression applications, digital communication, and biomedical imaging as well. Another example is the measurement of ECG signals that are analyzed using wavelets or compressed for storage. The Inverse Discrete Wavelet Transformation (IDWT) function applies the inverse of the DWT on multiple sequences simultaneously. After the IDWT implementation, the output sequences are the sequences in time domain. As the output is comparable with the input of DWT, the transformation is also called reconstruction.

See the following example that shows an original fingerprint and the reconstructed version after applying wavelet transformation:



Figure 34: Original Fingerprint



Figure 35: Reconstructed Fingerprint

A fingerprint (or almost any other picture) has a certain structure, which implies that the sequence of gray – tones associated with the pixels are not random numbers. Consider for example a picture showing a house: the gray -tones will be almost constant i.e. the walls and only around corners and windows will there be large variations. Similarly, in the case of fingerprints large areas with almost constant gray-tones (and therefore small differences) give good options for compression and as can be seen in the above example, excellent results are obtained. The wavelet transformation therefore has the ability to save on storage, reconstruct pictures from the original with clarity even if the original is partially damaged, and gives many other benefits.

Function Call for dwt2d:

```
SELECT *
FROM dwt2d(
ON (SELECT 1) PARTITION BY 1
INPUTTABLE('wavelet2d')
OUTPUTTABLE('wavelet2d_coef')
METATABLE('wavelet2d_meta')
INPUTCOLUMNS('value')
PARTITIONCOLUMNS('seqid')
INDEXCOLUMNS('y','x')
RANGE('(1,1),(5,6)')
WAVELETNAME('db2')
COMPACTOUTPUT('true')
LEVEL(2)
);
```

Function Call for dwt:

```
select * from public.dwt(
on (select 1) partition by 1
password('beehive')
inputtable('public.temperature')
outputtable('temperature_coef')
metatable('temperature_meta')
inputcolumns('value')
sortcolumn('hour')
partitioncolumns('city','date')
waveletname('db2')
level(2)
);
```

Function Call for idwt:

```
SELECT *
FROM idwtreduce(
ON <INPUTTABLENAME> PARTITION BY <PARTITIONCOLUMNNAME> ORDER BY <SORTCOLUMNNAME>
ON <METATABLENAME> as metatable PARTITION BY <PARTITIONCOLUMNNAME>
INPUTCOLUMNS( '<col1>', '<col2>',..., '<colN>')
```

TERADATA ASTER DISCOVERY PORTFOLIO

```
SORTCOLUMN('<sort_column_name>')  
[DISTRIBUTECOLUMN('<distributed_column_name>')]  
[PARTITIONCOLUMNS('<partition_column_name1>','<partition_column_name2>',..., '<partition_column_nameN>')  
];
```

Sample Use Cases:

- Image recognition.
- Fingerprint matching: In Singapore, a new security system was introduced in Hitachi Tower (a 37- storey office building) in 2003: now, the ~1500 employees get access to the building by scanning their fingers. The scanner uses infrared rays to trace the hemoglobin in blood in order to capture the vein patterns in the finger; these patterns determine the person uniquely. After comparing with the scanned data in an electronic archive, it is decided whether the person can get in or not Christensen. This is made possible by wavelet compression technology.
- Audio and video compression.

Sample Verticals:

- Consumer Electronics
- Law Enforcement
- Arts
- Communication
- Information Technology
- Government (National Security and Crime Prevention)

Dynamic Time Warping

Function Name: dtw

Dynamic time warping (DTW) is an algorithm for measuring similarity between two sequences which may vary in time or speed. For instance, similarities in walking patterns would be detected, even if in one video the person was walking slowly and if in another he or she were walking more quickly, or even if there were accelerations and decelerations during the course of one observation. DTW has been applied to video, audio, and graphics — indeed, any data which can be turned into a linear representation can be analyzed with DTW. A well-known application has been automatic speech recognition to cope with different speaking speeds. Other applications include speaker recognition and online signature recognition. DTW allows a non-linear mapping of one signal to another by minimizing the distance between the two. It is now slowly being introduced into the Big Data Discovery community to address various time series problems including classification, clustering, and anomaly detection. The key metric that measures the dissimilarity between two time series is called the warping distance.

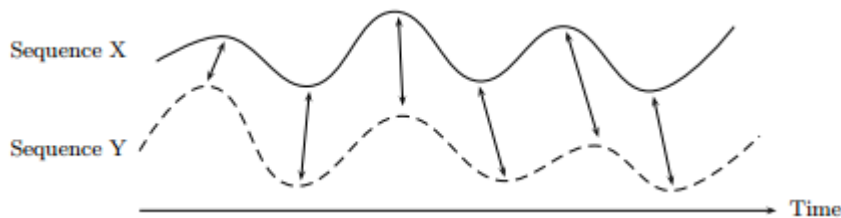


Figure 36: Comparison of two series – aligned points indicated by arrows.

Function Call:

```
on timeseriesdata as input_table  
partition by TimeSeriesID  
order by timestamp1  
on templatedata as template_table
```

```

dimension
order by timestamp2
on mappingdata as mapping_table
partition by TimeSeriesID
input_table_value_column_names('value1', 'timestamp1')
template_table_value_column_names('value2', 'timestamp2')
timeseriesid('timeseriesid')
templateid('templateid')
--radius('12')
--warpPath('true')
);

```

Sample Use Cases:

- Speech recognition
- Biological process unfolding studied through RNA time series
- Genomic signal classification

Sample Verticals:

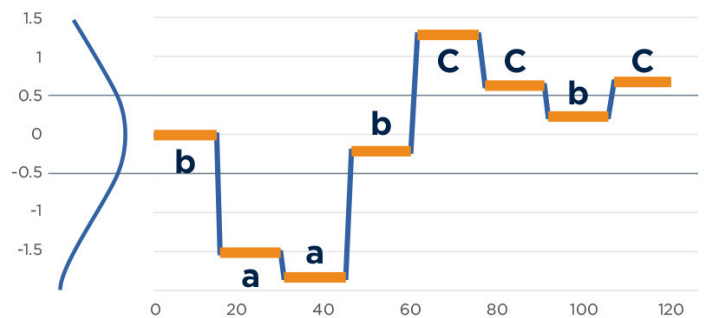
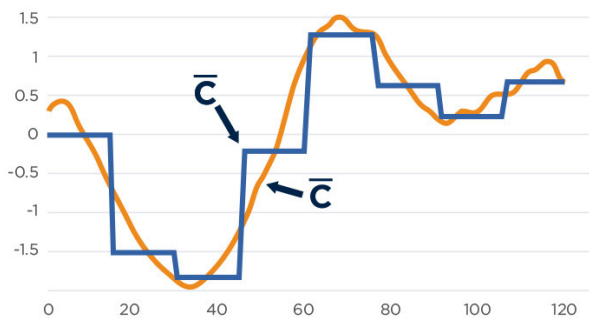
- Bioinformatics
- Audiovisual processing
- Pharmaceuticals

Symbolic Aggregation Approximation

Function Names: saxify_reduce, saxify_jdbc

Symbolic Aggregation Approximation (SAX) is a cutting-edge algorithm that powers quick, scalable processing of large-scale time series data sets to identify data anomalies and trends. Time series data are observations made sequentially over time. Examples of time series data include stock values and popularity ratings. The SAX function is implemented to classify, cluster, and discover motifs (reoccurring patterns), rule identification, novelty detection, and querying. Conventional methods for time-series analysis, unlike SAX, either rely on processing the raw time series data multiple times or rely on simplifying the time-series -- approaches that lose valuable information and are not conducive for tackling root-cause analysis problems.

The first step in the SAX process is to reduce a time series into a linear combination of box basis functions – called lower bounding. In the figures below, dimensionality reduction results in eight dimensions instead of 128 time series points. The second step occurs when the dimensions are classified into breakpoints -- a boundary that separates the dimensions of a time series graph. The final step is when each dimension within a break point is assigned an alphabet.



Figures 37 and 38: Dimensionality reduction with SAX

The illustration in the left shows dimension reduction and the right shows break point identification and alphabet assignment. The figure below shows the results when these three steps are implemented on two time series data sets.

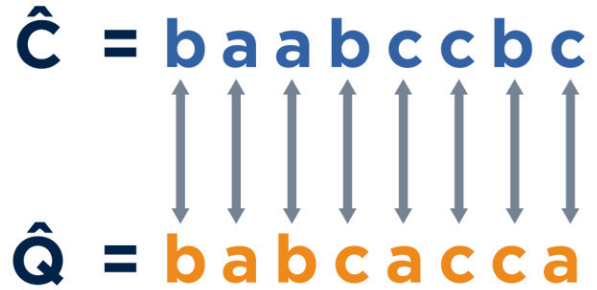
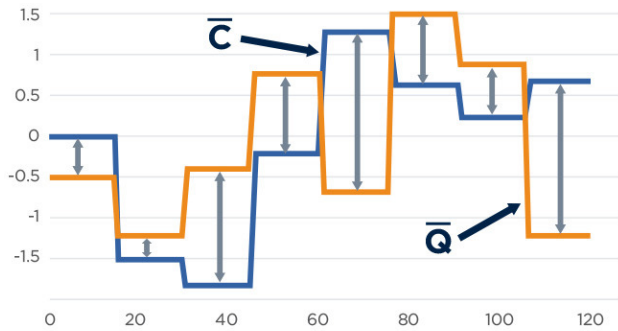


Figure 39: Comparing two time series using SAX

The above output of SAX with each series deconstructed into a set of symbols makes it very easy to determine pattern differences, data deviations, sensitivity analysis, and many more.

Function Call:

```
select * from saxify_reduce(on(
  select * from saxify_jdbc( on temp
    partition by 1
    TS_COLUMN_NAME('time_stamp')
    VALUE_COLUMN_NAME('value')
    UNIT_COLUMN_NAME('unit')
    MEASURE_COLUMN_NAME('measure')
    RANK_COLUMN_NAME('dim_id')
    START_TIME('2004-01-01 00:00:00')
    END_TIME('2004-12-31 23:59:00')
    CHILD_TABLE_BASE_NAME('ge_timeseries' )
    CHILD_TABLE_SIZE('43200000')
    METADATA_TABLE_NAME('ge_metadata')
    USER_ID('beehive')
    PASSWORD('beehive')
    DATABASE('ge11')
    DOMAIN('10.60.0.100')
  )
  partition by dim_id
  order by start_time
  TS_COLUMN_NAME('start_time')
  SAX_CODE_COLUMN_NAME('sax_code')
  ACCUMULATE_COLUMNS('unit','measure','dim_id')
);
```

Sample Use Cases:

- Using machine data to analyze manufacturing deficiencies and isolating groups of abnormalities that would ensure reduced product recalls, minimize returns, and increase durability.
- Tracking minute changes in data that could indicate the onset of issues that could have a large impact. For example, in the aviation industry, key components of an airplane (e.g., engine) are constantly monitored to see if key performance indicators differ from the norm.
- Introducing condition-based maintenance (CBM) for large organizations that deal with heavy, large, and expensive equipment (e.g., Army). Vibration analysis, motor current signature, ultrasound and oil analysis, can be used to help assess the efficacy of machinery and predict future failures. Some advantages of condition-based monitoring include

TERADATA ASTER DISCOVERY PORTFOLIO

improved system reliability, increased production, decreased maintenance costs, and less human intervention. The exchequer also benefits substantially from the savings that are realized through this function's implementation.

Sample Verticals:

- Consumer electronics
- Aviation
- Medical systems
- Manufacturing
- Military
- Energy

Visualization Functions

Collaborative Filter Visualization

Function Name: cFilterViz

The cFilter visualization function represents the output of the Cfilter function call as visuals. The forms of visualization are unique to the Teradata solution and are not available as features in any of the BI solutions (e.g., Tableau) that the Aster Discovery Portfolio is integrated with. In fact, these visualizations are complementary to the BI visualizations and extend the ability of the business user to get value out of seeing critical insights demonstrated in novel forms. The sigma graph in the figure below is one type of visualization:



Figure 40: Visualization showing the *togetherness* of certain Web pages that site users are likely to visit more often than not.

Function Call:

```
select id,time_stamp,graph_type from cfilterViz(
    on cfilterViz_data
    partition by 1
    score_col('score')
    item1_col('col1_item1')
    item2_col('col1_item2')
    cnt1_col('cnt1')
    cnt2_col('cnt2')
    title('My graph'));
```

Sample Use Cases:

- Identifying purchasing patterns (e.g., dyads of goods or services) so that enterprises can understand how to market clusters of products.
- Identifying action associations (e.g., when an automobile is brought in for tire service, then the electronics also should be checked) that ensures reduced repeat actions and leads to more efficient outcomes.
- Making product recommendations (e.g., users who bought gift paper also purchased a tapes, glue, and scissors).
- Showing social linkages that enable enterprises to isolate key influencers in a given community. Thereafter, enterprises can take appropriate steps to work with the key influencers for a given purpose (e.g., greater awareness of health care issues, increased adoption of specific services).

Sample Verticals:

- Manufacturing
- Automotive

TERADATA ASTER DISCOVERY PORTFOLIO

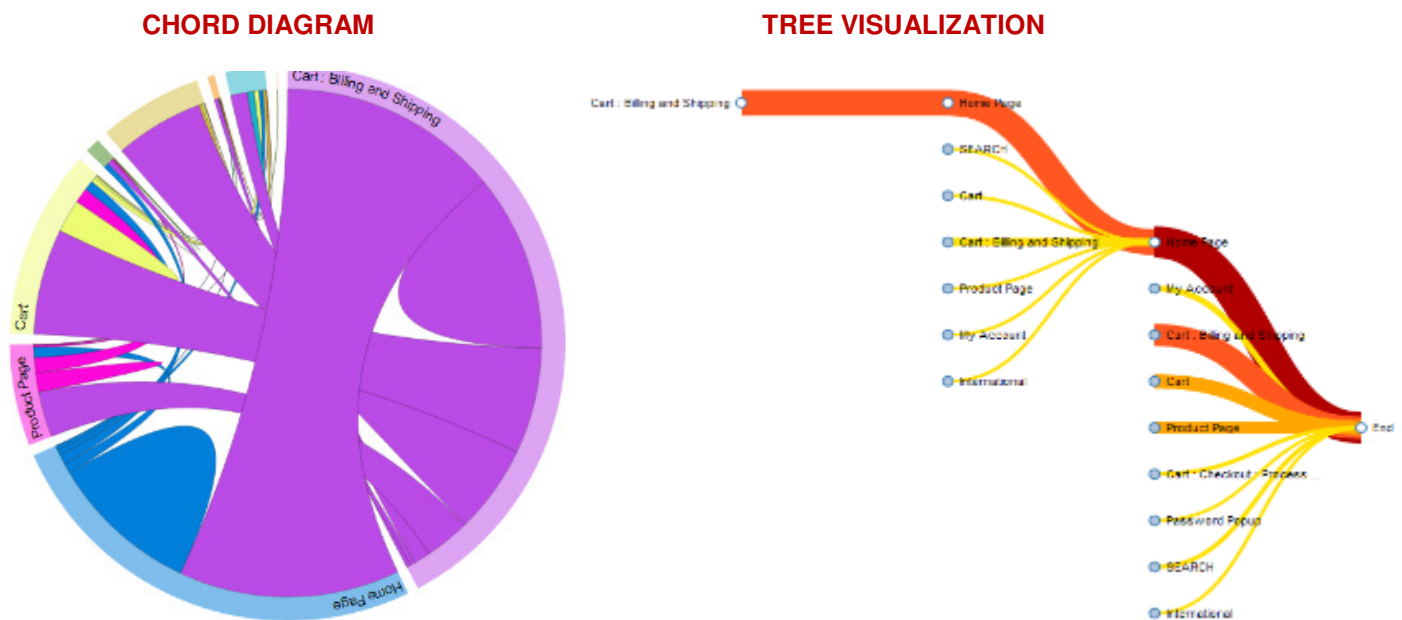
- Banking
- Retail
- Social media
- Government (disease control)

Teradata Aster nPath Visualization

Function Name: nPathViz

nPathViz is a SQL-MR function that transforms the output of the Teradata Aster nPath™ function into compelling visualizations that include Chord, Sankey, Sigma, and Tree diagrams. nPathViz extends Aster's capabilities to allow the end user to visualize their Teradata Aster nPath™ output through natively built applications. The output of these visualizations is in the form of a BLOB (Binary Large Object) containing all the data required for visualization along with some metadata related to that BLOB. A BLOB is a collection of binary data stored as a single entity in a database management system. Blobs are typically images, audio or other multimedia objects.

As in the case of the cFilterViz function noted above, the nPathViz function is yet another unique addition to the Teradata Database capabilities. These visualizations are unique to the discovery platform and are not available through any solution integration. They are intended to be complementary to the BI visualizations that come with integrations to BI solutions such as Tableau®, Spotfire®, Microstrategy®, and more. Examples of the Chord and Tree visualizations are below (Sankey and Sigma visualizations have been rendered elsewhere in this document):



Figures 41 and 42: Chord and Tree visualization examples

Function Call:

```
create table output_table distribute by hash(geo_country) as (  
  select * FROM nPathViz(  
    ON npath_output as input  
    partition by geo_country  
    ON (select geo_country ,max(visit_num) from npath_output group by geo_country) as aggregates  
    partition by geo_country  
    frequency_col('cnt')  
    graph_type('sankey')
```

```
path_col('path')
arguments('start date=10/12/2013','end date=10/30/2013')
accumulate('geo_country')
)
)
```

Sample Use Cases:

- Visualizing a customer's Web site traversal as she purchases products online.
- Showing, through the thickness of the chords, the relationship between nodes in the purchase path (e.g., the hardware items page has the strongest linkage with the Home and Garden section).
- Displaying how customer calls to the call center can be classified and accordingly addressed with a combination of automated processes and human intervention resulting in huge costs savings.

Sample Verticals:

- Telecommunications
- Retail
- E-commerce
- Healthcare

Writing custom SQL-MR functions in Aster

The preceding sections provided a glimpse into the nature and types of SQL-MR and SQL-GR functions that are pre-built and made available at purchase time for customers of the Teradata Aster solution. While Teradata has aimed to create the most important and representative set of such functions that could address the broadest possible set of use cases, it may still not address *all* possible use cases. While new functions will be added with regular cadence, for more urgent needs, customers will be able to address any gaps between current capabilities and their needs by developing custom functions using the Aster Developer Express (ADE). ADE is a crucial component of the Aster solution and is an easy to use, visual development environment for writing new SQL, SQL-MR and SQL-GR functions. Key features include:

- Familiar Eclipse-based IDE with powerful Aster Plug-in.
- Automatically generates code template with SQL glue code.
- Can import existing analytic logic and 3rd Party functions.

The goal behind including ADE in the Aster solution is to ensure that an intuitive development environment can be leveraged to create new analytics functions that are performant and custom built to suit unique business needs.

Teradata Aster and R Language Integration

Teradata will introduce Teradata Aster R, new software that delivers open source R without limitations through the Teradata Aster Discovery Platform. Teradata Aster R allows programmers to continue to operate within R and lift the data and processing limitations of open source R making what was previously impossible, a reality. The general customer availability of Aster R is scheduled for Q1 2015.

Teradata Aster R benefits include:

Open-source R without limitations: Teradata Aster R lifts memory and data limitations by leveraging the Aster database as the high performance computing R engine. Aster's MPP architecture and massive scalability enables big data analytics across all your data.

Unmatched ease-of-use and productivity for R users: R analysts can now use familiar R client and R language to access a library of parallel functions including the rich and powerful Aster Discovery Portfolio.

Powerful analytics combining Aster and R: Integrating open source R engine into the SNAP Framework delivers a single powerful analytic environment for your analytic community – including R programmers, business analyst and data scientist. By combining over 100 Teradata Aster Discovery Portfolio functions, 5500+ open source R packages and Teradata Aster R parallel constructors, analyst now have the tools to conquer business problems.

Conclusion

The Teradata Discovery Portfolio is a compelling part of the Teradata Aster Discovery Platform. The Aster Discovery Platform, which includes the Discovery Portfolio and the Teradata Aster Database, significantly enhances an enterprise's ability to perform the discovery process (data acquisition – data preparation – analysis – visualization) in a fail fast manner. This ensures that insights are delivered when it really matters as well as creating an ability to iterate on crucial insights by honing them or uncovering other unrelated insights not seen in alternate approaches. While enterprise customers are benefiting substantially from this solution, other categories of businesses (e.g., SMBs, Departments within enterprises) can take equal advantage of this solution by implementing the built in functions without much investment in data science skills or without having to pay for expensive support and consulting services to get the benefits of advanced analytics. The Teradata Aster Discovery solution is available in an appliance form factor and has been tested with various hardware configurations and is performant in any analytics environment. The functional capabilities have been developed with a view towards addressing numerous use cases across many industry verticals.

For More Information

To find out how the Teradata Aster Discovery Portfolio can help you take advantage of big data volumes in a fast, efficient, and cost-effective way while you improve your decision-making capabilities and grow a stronger, more productive business or to learn more about Teradata Aster Discovery Platform, contact your local Teradata representative or visit teradata.com.

Teradata, the Teradata Logo, Aster, and SQL-MapReduce are registered trademarks and nPath, SNAP Framework are trademarks of Teradata Corporation and/or its affiliates in the U.S. and worldwide. All other trademarks shown are the property of their respective owners. Teradata continually improves products as new technologies and components become available. Teradata, therefore, reserves the right to change specifications without prior notice. All features, functions, and operations described herein may not be marketed in all parts of the world. Consult your Teradata representative or Teradata.com for more information.

About the Author

Sri Raghavan is a Senior Global Product Marketing Manager at Teradata and is in the big data area with responsibility for the Aster solution and all ecosystem partner integrations with Aster. Sri has more than 17 years of experience in advanced analytics and has had various senior data science and analytics roles in Investment Banking, Finance, Healthcare and Pharmaceutical, Government, and Application Performance Management (APM) practices. He has two Master's degrees in Economics and International Relations respectively from Temple University, PA and completed his Doctoral coursework in Business from the University of Wisconsin-Madison. Sri is passionate about reading fiction and playing music and often likes to infuse his professional work with references to classic rock lyrics, AC/DC excluded.

Copyright © 2014 by Teradata Corporation
All Rights Reserved. Produced in U.S.A.

EB6844 08.14