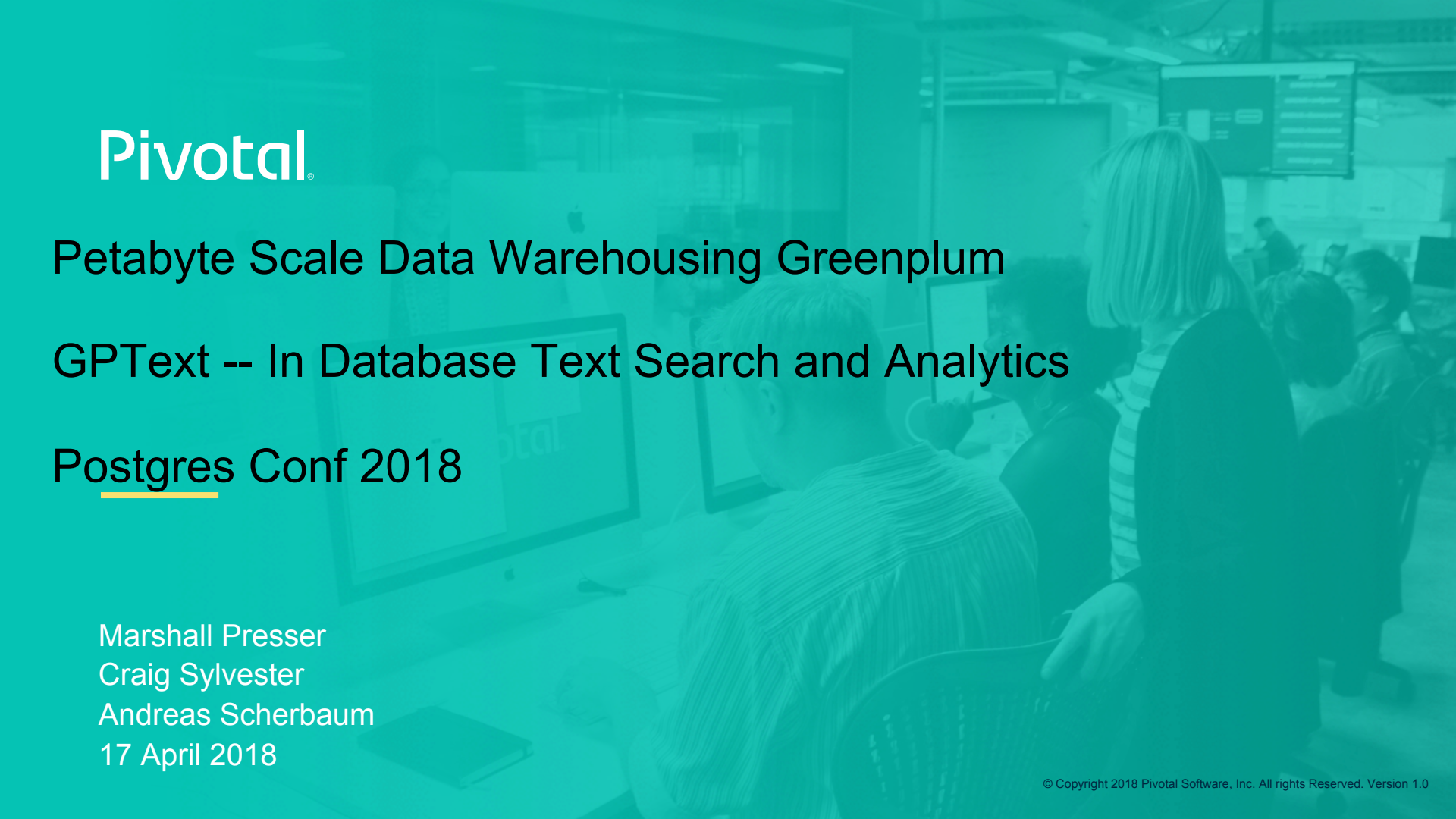


Pivotal

Petabyte Scale Data Warehousing Greenplum

GPText -- In Database Text Search and Analytics

Postgres Conf 2018

Marshall Presser
Craig Sylvester
Andreas Scherbaum
17 April 2018

What is GPText?

- Joins GPDB with Apache Solr enterprise search and MADlib Analytics Library to support large scale text analytics processing
- Supports free-text search and text analysis
- Supports semi-structured and unstructured data
- Uses statistical Machine Learning via MADlib for clustering, sentiment analysis, …

Pivotal

Why GPText

- GPText enables processing mass quantities of raw **text** data (such as social media feeds or e-mail databases) into mission-critical information that guides project and business decisions.
- Some potential uses are: sentiment analysis, forensic analysis (financial, insurance, compliance, security)

Pivotal

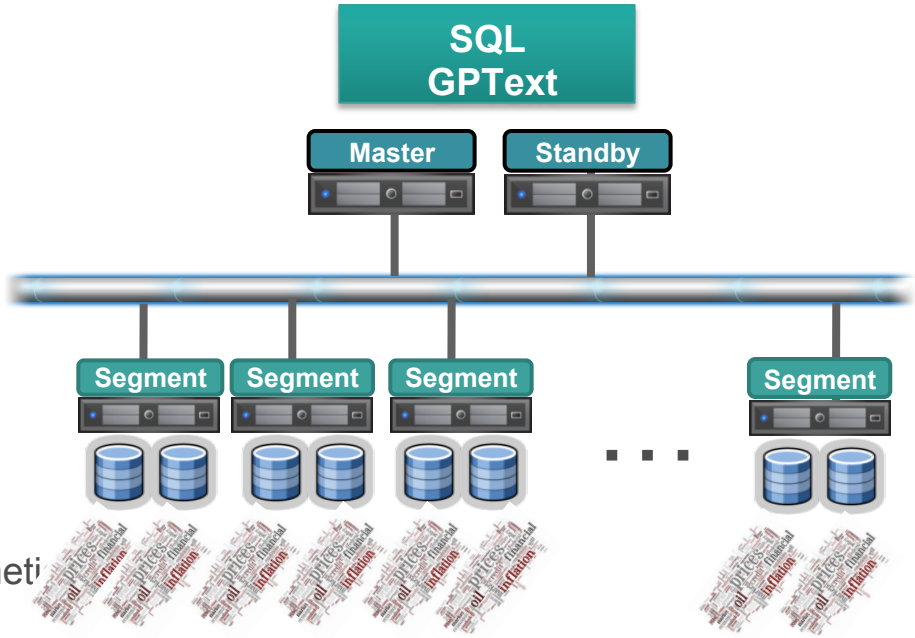
Introduction

Why can't we simply use an index for full-text search?

- Consider the phrase ‘GPDB is an MPP database management system’
- Tokenize - split into array of words: [‘GPDB’, ‘is’, ‘an’, ‘MPP’, ‘database’, ‘management’, ‘system’]::text[]
- Move to lowcase: [‘gpdb’, ‘is’, ‘an’, ‘mpp’, ‘database’, ‘management’, ‘system’]::text[]
- Stem using Porter stemming: [‘gpdb’, ‘is’, ‘an’, ‘mpp’, ‘databas’, ‘manag’, ‘system’]::text[]
- Build binary index

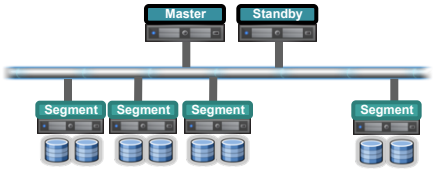
GPText architecture

- **Scalable**
  - One dedicated Solr instance per segment
  - “MPP Text” – can linearly scale
- **High Availability**
  - Replicated
  - Standby Solr instance per segment
- **Database management features**
  - Backup/restore
  - Online expansion
  - Data recovery
  - Performance monitoring
- **Full Text Search**
  - Flexible indexing and search (stemming, phoneti search, multilingual search)
- **Join** structured & text in one query



GPText capabilities

- Index creation
  - Create, load, commit an index



- Real data schema
  - Numeric types, dates, text
  - Define custom analyzer chains for different fields

```
<fields>
  <field name="id"       type="string" indexed="true"  stored="true" required="true"/>
  <field name="type"     type="string" indexed="true"  stored="true"/>
  <field name="artist"   type="text"   indexed="true"  stored="true"/>
  <field name="album"    type="text"   indexed="true"  stored="true"/>
  <field name="track"    type="text"   indexed="true"  stored="true"/>
  <field name="genre"    type="string" indexed="true"  stored="true" multivalued="true"/>
  <field name="tracknr"  type="sint"   indexed="true"  stored="true"/>
  <field name="duration" type="sint"   indexed="true"  stored="true"/>
  <field name="year"     type="sint"   indexed="true"  stored="true"/>
  <field name="location" type="string" indexed="false" stored="true"/>
</fields>
```

- Provides advanced full text search (FTS) capabilities
  - Faceted search and filtering
  - Lucene (default), complex, and surround query parsers

- Vector space representation of document

Term-document matrix

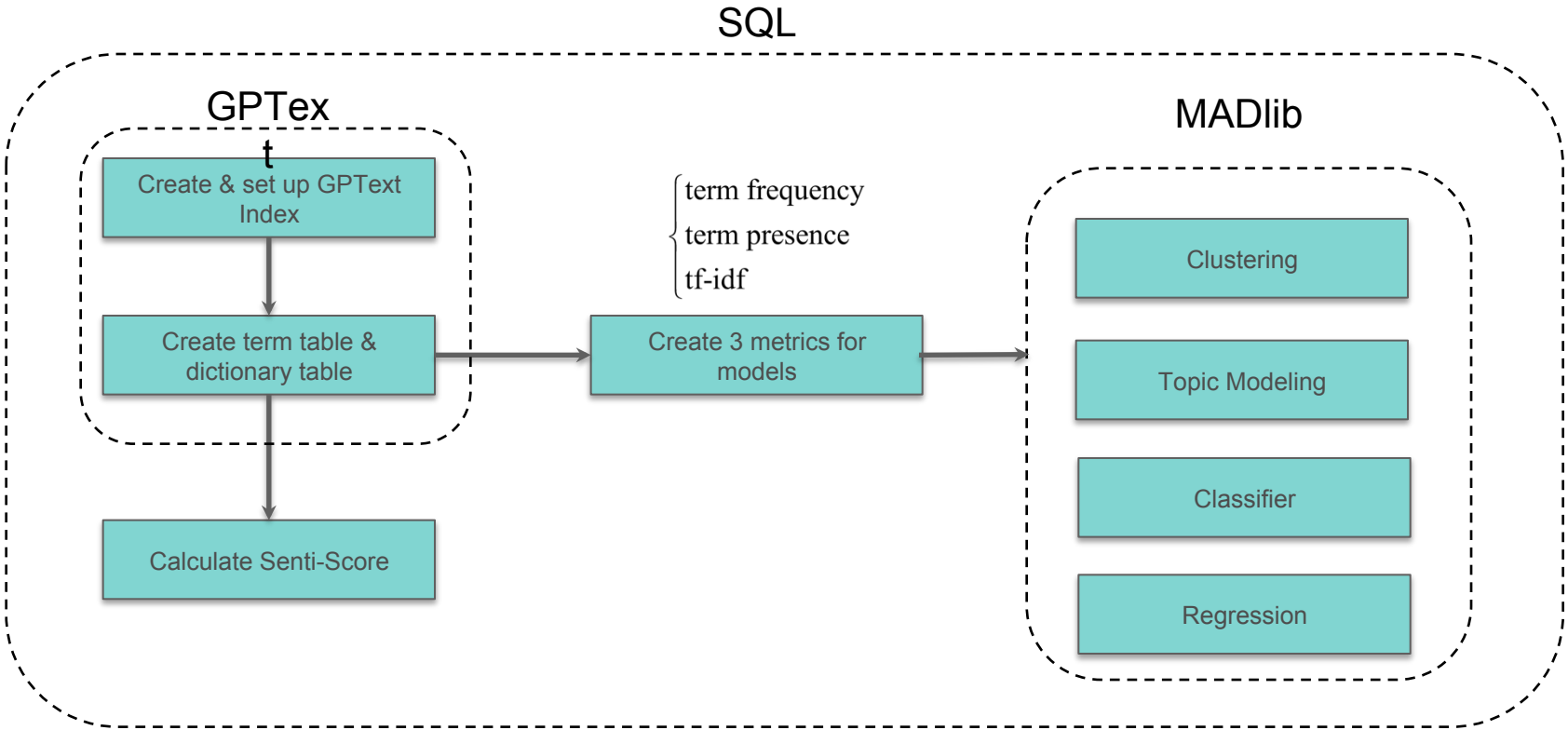
$$\begin{pmatrix} & T_1 & T_2 & \dots & T_t \\ D_1 & w_{11} & w_{21} & \dots & w_{t1} \\ D_2 & w_{12} & w_{22} & \dots & w_{t2} \\ \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & & \vdots \\ D_n & w_{1n} & w_{2n} & \dots & w_{tn} \end{pmatrix}$$

Each w_{tn} is a list containing the positions of term T_t in document D_n

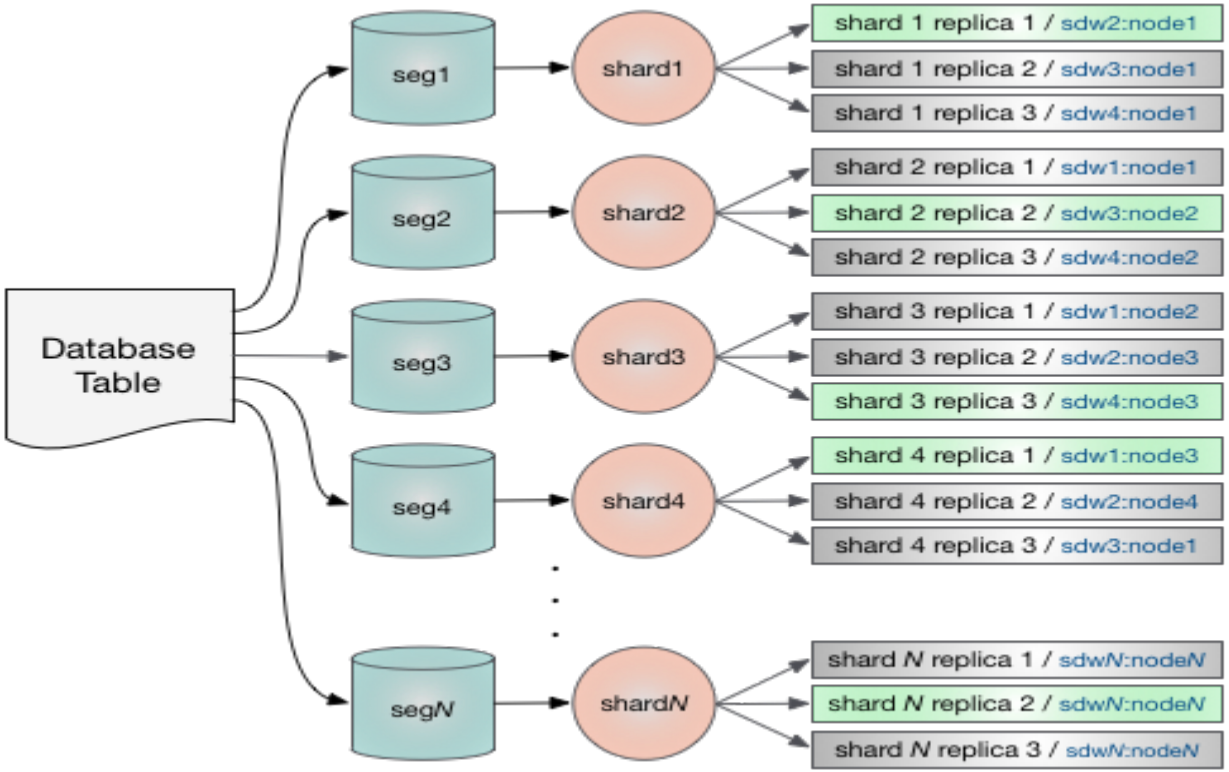
- Text analytics via Madlib

Pivotal

Text Analytics Workflow



General Architecture - GPText 2.0



General Architecture

- GPText consists of
  - Database functions in schema “gptext”
    - Some functions are in pl/pgsql for simplicity
    - Main interface functions are in C
  - Shared library “gptext-2.0.0.so”
    - Contains postgres C functions
    - Uses libcurl.so to query REST API of the Solr
    - Uses libxml2.so to parse XML response from Solr REST API
  - SolrCloud
    - Zookeeper instances manage and monitor SolrCloud cluster
    - Solr core jar libraries
    - Lucene core jar libraries

Main Functions

- Index functions
  - create index, index, drop index, commit index, etc.
- Search functions
  - search, faceted search, etc.
- Terms operations
  - enable terms for index, create terms table for specific query, document vector creation, etc.
- General functions
  - Index statistics, gptext status, gptext version, etc.

Main Functions

- Create Index
  - Metadata operation in Solr, simple REST API call
- Index Data
  - Function executes on the segment level
  - Function accepts stream of table tuples and passes them to the Solr instance
  - Solr performs analysis of passed in text and adds them to the index
- Commit Index
  - Metadata operation in Solr, simple REST API call

Main Functions

- Search
  - Simple REST API call performed by each of the segments
  - Each of the segments returns data separately and does not know about results returned by other segments
- Drop index
  - Metadata operation in Solr that causes index files to drop, simple REST API call

Examples - Setup

```
create schema test;
create table test.test_data  (id bigint, data varchar);
insert into test.test_data (id, data) values
(1, 'example text #1, not much information'),
(2, 'example text #2, a bit more information, but still not enough'),
(3, 'A.Grishchenko is reading this course'),
(4, 'Verint team is participating'),
(5, 'This course is really intensive'),
(6, 'Design of the distributed systems is really important'),
(7, 'The information you get here should be valuable');

select gptext.create_index ('test', 'test_data', 'id', 'data');
select * from gptext.index(TABLE(select * FROM test.test_data), 'test.test.test_data');
select * from gptext.commit_index('test.test.test_data');
```

Examples - Search 1

```
test=# select t.id, t.score, t2.data
test-# from gptext.search(TABLE(SELECT 1 SCATTER BY 1),
test-#      'testdb.test.test_data', 'example', null, null) as t,
test-#      testdb.test_data as t2
test-# where t.id = t2.id;

id  | score  |                                 data
----+--------+-------------------------------------------------------------
  1 | 0.7407 | example text #1, not much information
  2 | 0.4392 | example text #2, a bit more information, but still not enough
```

Examples - Search 2

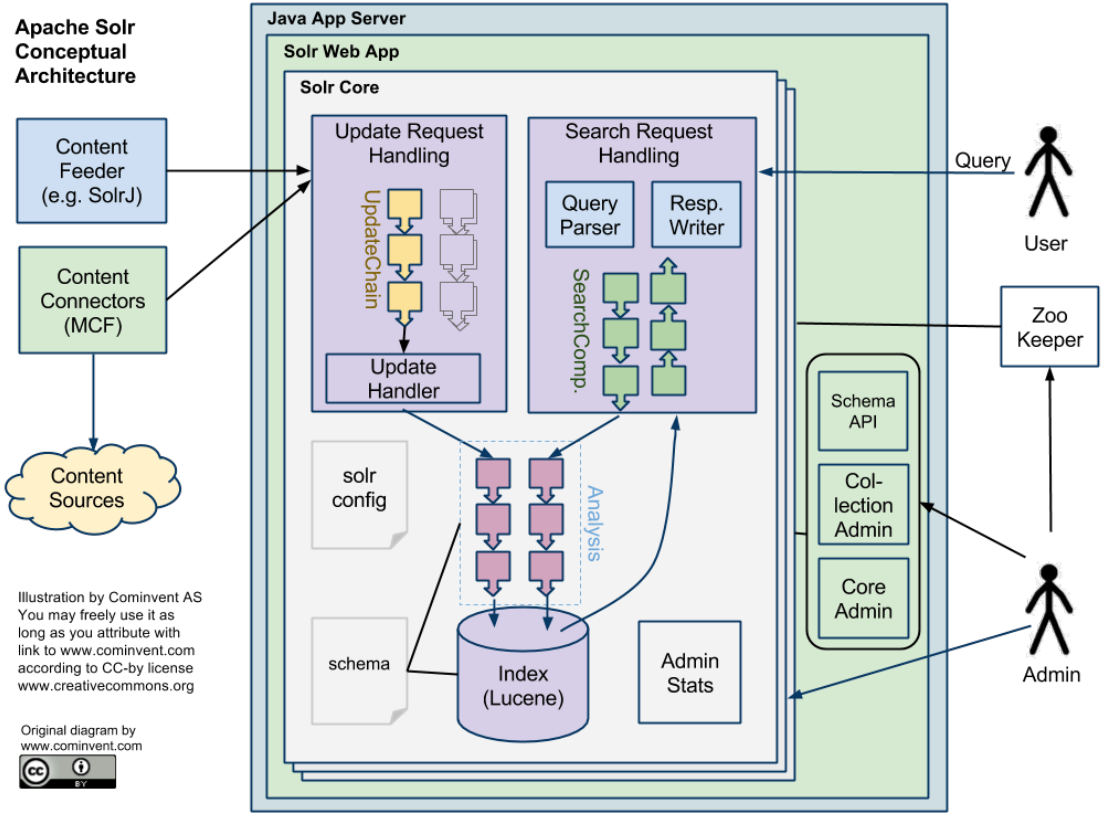
```
test=# select t.id, t.score, t2.data
test-# from gptext.search(TABLE(SELECT 1 SCATTER BY 1),
test-#      'testdb.test.test_data', 'valuable information', null, null) as t,
test-#      testdb.test_data as t2
test-# where t.id = t2.id;

 id | score  |                                 data
----+--------+-----------------------------------------------------------------
  2 | 0.1221 | example text #2, a bit more information, but still not enough
  1 | 0.1705 | example text #1, not much information
  7 | 0.7976 | The information you get here should be valuable
```

Examples - Search 3

```
test=# select t.id, t.score, t2.data
test-# from gptext.search(TABLE(SELECT 1 SCATTER BY 1),
test-#      'testdb.test.test_data', 'course information', null, null) as t,
test-#      testdb.test_data as t2
test-# where t.id = t2.id;

 id | score  |                              data
----+--------+-----------------------------------------------------------------
  1 | 0.1991 | example text #1, not much information
  3 | 0.2276 | A.Grishchenko is reading this course
  5 | 0.2276 | This course is really intensive
  7 | 0.1707 | The information you get here should be valuable
  2 | 0.1221 | example text #2, a bit more information, but still not enough
```

Solr Introduction

Conceptual Architecture

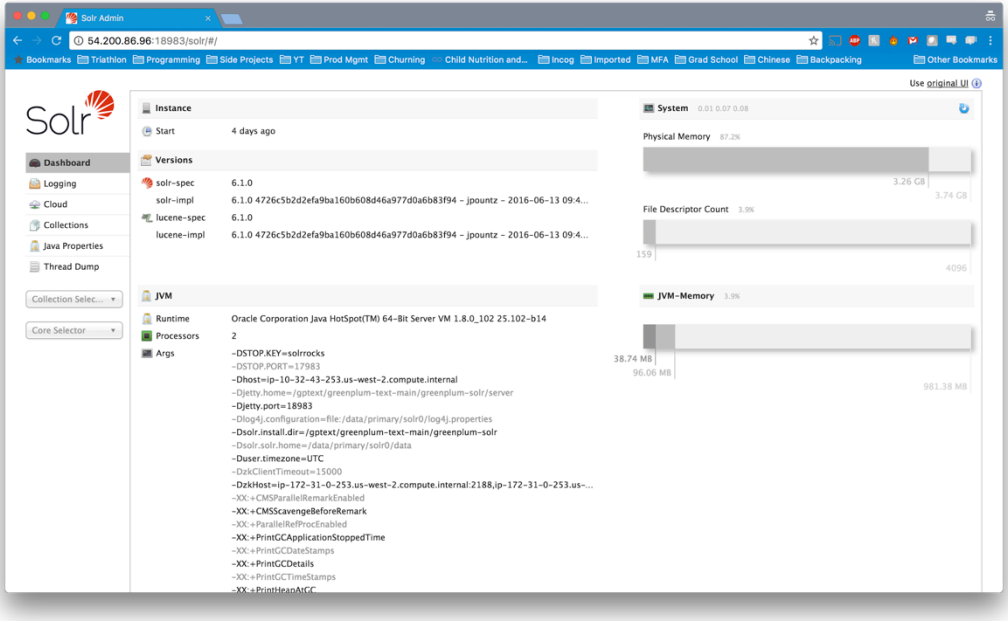


Conceptual Architecture

- Solr consists of
  - Jetty Servlet Container
  - Solr Web Application
  - Solr Core Libraries
  - Lucene Indexing Core
- Jetty Servlet Container
  - Basically is HTTP server for Java applications
  - Hosts Solr web application “solr.war”

Conceptual Architecture

- Solr Web Application
  - Provides WebUI for system status and performance monitoring



Conceptual Architecture

- Solr Web Application
  - Provides REST API for applications (querying, data loading, maintenance, etc.)

Conceptual Architecture

- Solr Core Libraries
  - Handling multiple independent indexes
  - Allowing specification of data/query processing chains using schema.xml/managed-schema
  - Providing replication capabilities
  - Tokenizers, Analyzers, Filters
- Lucene Indexing Core
  - Index storage, caching
  - Transactional updates of the index
  - Querying the index, ranking the results

Differences?

GPText vs. Solr

- Distributed Solr - Highly scalable
  - Leverages GPDB MPP architecture to distribute nodes and shards across multiple hosts
- SQL Interface
  - All Solr calls and functionality are exposed through SQL UDFs for a consistent interface through psql/pgadmin/etc
- Ease of Configurability
  - GUC Configuration management
  - gptext-config
- Integrated with GPDB and included in PDS (Pivotal Data Suite)!

GPText vs. Solr (cont.)

- International/Social Media Analyzer Chains
  - Additional Analyzer Chains with custom Tokenizers/Filters to be able to accurately parse out International Text (including CJK characters) and Social Media (hashtags, emoticons, links, etc.)
- Unified Query Parser
  - Combines Solr and Lucene Query parsers into a single unified interface
- Enables term vector support for text analytics with MADLib
  - LDA, K-Means Clustering, SVM

Searching with GPText

Anatomy of a Search Query

```
SELECT * FROM gptext.search(
  TABLE( SELECT 1 SCATTER BY 1),
  idx_name,
  search_query,
  filter_queries,
  options
);
```

Searches text index for <search_query> and returns id's and relevance scores of matching documents

- idx_name - The name of the index to search.
- search_query - Use enhanced Solr/Lucene query parser syntax
- filter_queries - Array to filter out search terms by other columns in the index.
- options - Solr options to limit data, order, highlight, or return full results.

Basic Search Query

Basic lucene query syntax:

- Term Match: “test” or “hello”
- Field Specification: title:”The Right Way” AND text:”some text”
- Wildcard Searches: te?t OR test*
- Fuzzy Searches (Levenshtein Distance): roam~ OR roam~0.8
- Proximity Searches: “jakarta apache”~10
- Range Searches: id:[5 TO 10] OR title:{Aida TO Carmen}
- Term Boosting: jakarta^4 apache
- Boolean Searching: AND, OR, NOT, +, -, etc..
- Grouping: (jakarta OR apache) AND website

Basic Search Query

Enhanced solr query syntax:

- Wild cards may be used in ranges: field:[* TO 100]
- Pure negative queries: -inStock:false
- Nested query parser support: _query_:”{!dismax}how now brown cow”

Basic Search Query

Greenplum Unified Query Parser:

- Combines functionality from common community query parsers
  - Ordered Proximity Searches: 5n(dog cat) OR 5w(cat dog)
  - Nested Proximity Searches: 5n(dog AND 3n(cat mouse))
  - Wildcards in Phrases: “how now * brown cow”

Basic Search Query - Example

```
demo=# select * from gptext.search(table(select 1 scatter by 1), 'demo.public.enron', 'content:"
Phillips *Petroleum" AND date:["2000-05-22T00:00:00Z" TO NOW]', NULL);
   id   |   score
--------+-----------
 709244 | 11.777072
  12818 | 10.025828
 301590 | 2.6093144
 175041 | 1.8506925
 266880 | 1.8506925
  87189 | 1.7560055
```

Search for email which contains ‘Phillips Petroleum’ after May 22nd,

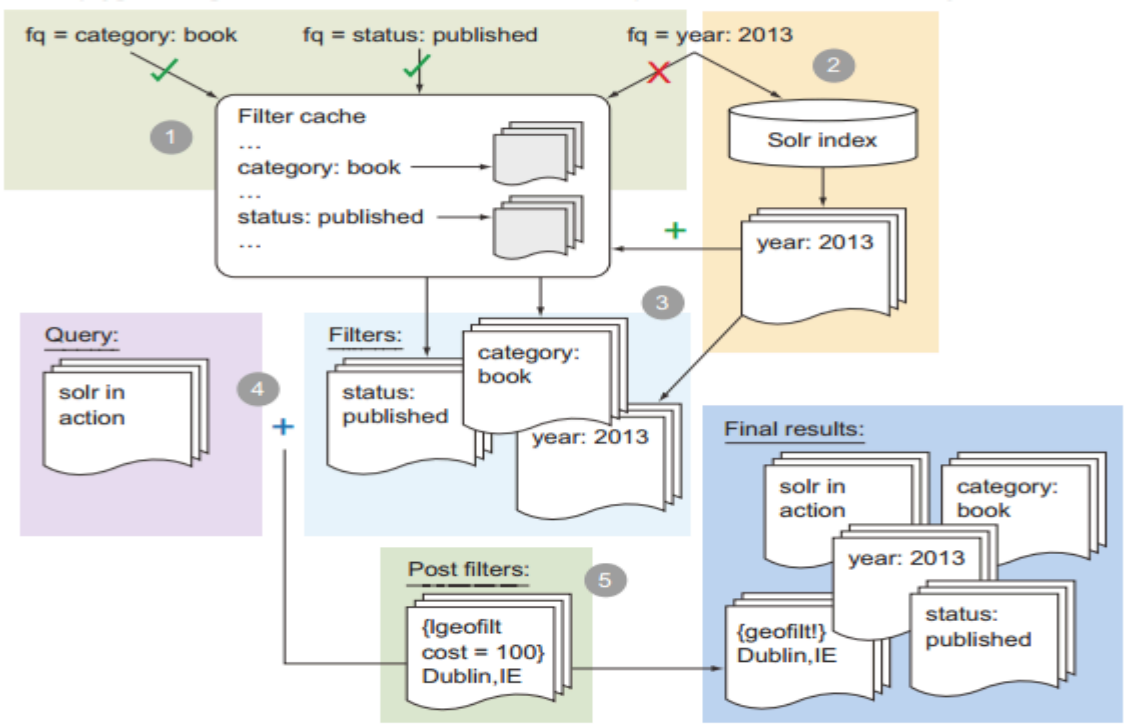
Returns both document id and score. Higher score is, more relevant document is.

Search Query with Filter Query

```
SELECT * FROM gptext.search(
  TABLE( SELECT 1 SCATTER BY 1),
  idx_name,
  search_query,
  filter_queries,
  options
  );
```

- Use filter queries to speed up search by filter cache.
- gptext-config -i <index_name> -f solrconfig.xml to choose your own filter cache in <filterCache> section of solrconfig.xml
- Put often occuring parts in search_query to filter_queries.

Search Query with Filter Query



Search Query with Filter Query

Search for email including ‘Phillips Petroleum’ in year 2000.

```
SELECT * FROM gptext.search(
  TABLE( SELECT 1 SCATTER BY 1),
  'demo.public.enron',
  'content:"Phillips Petroleum"',
  array['date:["2000-01-01T00:00:00Z" TO "2001-01-01T00:00:00Z"]'],
  options
  );
```

Search Query with Filter Query - Multi Filters

```
demo=# select * from gptext.search(table(select 1 scatter by 1), 'demo.public.enron', 'content:Phillips Petr
oleum', array['from:"Eric Bass"', 'to:"Bryan Hull"']);
   id   |   score
--------+-----------
  92546 | 4.8615003
 101725 |  5.698272
  93116 |  4.389782
  96911 |  3.607851
  95748 |  4.198564
  98404 | 4.1117644
 100644 | 4.5973567
```

- Search query with multiple filters
- Search ‘Phillips Petroleum’ related emails sent from ‘Eric Bass’ and to ‘Bryan Hull’

Search Query with Options

```
SELECT * FROM gptext.search(
  TABLE( SELECT 1 SCATTER BY 1),
  idx_name,
  search_query,
  filter_queries,
  options
  );
```

Many solr options you could set in options separated by '&'

- rows controls how many records returned for each segment
- hl and hl.fl controls highlighting related search
- fl returns extra fields including id and score
- ...

Search Query with Solr Limit

```
demo=# select count(*) from gptext.search(table(select 1 scatter by 1), 'demo.public.enron', 'content:Phillips Petroleum', NULL, 'rows=1');
 count
-------
    30
(1 row)

demo=# select count(*) from gp_segment_configuration where content >= 0;
 count
-------
    30
(1 row)
```

- Returns only 1 row for each shard (segment)
- There are 30 segments, therefore returns 30 records

Search Query

Joining results with original table

```
SELECT t.id t.content, q.score
  FROM
     mytable t,
     SELECT * FROM gptext.search(
           TABLE( SELECT 1 SCATTER BY 1),
           myschema.mytable,
           "cat AND dog",
           NULL,
           NULL
     ) q
   WHERE t.id = q.id
);
```

- Allows you to return more than just id and relevancy score
- Query can have arbitrary complexity - join with additional tables or functions

Search Query with Highlighting

```
demo=# select gptext.highlight(content, 'content', hs) from gptext.search(table(select 1 scatter by 1), 'dem
o.public.enron', 'content:Phillips Petroleum', NULL, 'hl=true&hl.fl=content') l, enron r where l.id=r.id lim
it 10;
-[ RECORD 1 ]--------------------------------------------------------------------------------
highlight | how was the well this weekend?
          |
          |
          |
          |         Enron North America Corp.
          |
          |         From:  Timothy Blanchard @ EES                          03/20/2000 08:52 AM
          |
          |
          | To: Eric Bass/HOU/ECT@ECT, Chad Landry/HOU/ECT@ECT, Matthew
          | Lenhart/HOU/ECT@ECT, <em>Phillip</em> M Love/HOU/ECT@ECT, Bryan Hull/HOU/ECT@ECT,
```

- Enable highlighting:
  - select gptext.enable_terms(<index_name>, <field_name>)
- Join with original table to get highlight rendering.
- Change highlight tag by GUC gptext.pre_hl_tag and gptext.post_hl_tag

Search Query with Returning Full Fields

```
SELECT *
  FROM gptext.search(
    TABLE( SELECT 1 SCATTER BY 1),
    idx_name,
    search_query,
    filter_queries,
    'fl=...'
  );
```

- Extra fields could be returned by fl option
- gptext-config -i <index_name> -f managed-schema to change store as True
- The result is buffer format, which looks like repeated: column_value { name: ‘<field_name>’ value: ‘<field_value>’ }
- Use gptext_retrieve_field, gptext_retrieve_field_int8, gptext_retrieve_field_float8 to retrieve values.

Search Query with Returning Full Fields

```
<field name="folder" type="text_intl" indexed="true" stored="false"/>
<field name="from" type="string" indexed="true" stored="true"/>
<field name="id" type="long" indexed="true" stored="true"/>
<field name="ignorecode" type="int" indexed="true" stored="false"/>
<field name="subject" type="text_intl" indexed="true" stored="false"/>
<field name="to" type="string" indexed="true" stored="true"/>
```

```
demo=# select * from gptext.search(table(select 1 scatter by 1), 'demo.public.enron', 'content:Phillips Petr
oleum', NULL, 'fl=from,to');
-[ RECORD 1 ]----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
id    | 253757
score | 7.4884057
hs    |
rf    | column_value { name: "from" value: "Phillip K Allen" } column_value { name: "to" value: "Matt Smith
<Matt.Smith@enron.com>" }
-[ RECORD 2 ]----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
id    | 253140
score | 7.428556
hs    |
rf    | column_value { name: "from" value: "Ratcliff  Renee <Renee.Ratcliff@ENRON.com>" } column_value { nam
e: "to" value: "Allen  Phillip K. <Phillip.K.Allen@ENRON.com>" }
```

Using gptext.gptext_retrieve_field

```
demo=# select gptext.gptext_retrieve_field(rf, 'from') "from", gptext.gptext_retrieve_field(rf, 'to') "to" f
rom gptext.search(table(select 1 scatter by 1), 'demo.public.enron', 'content:Phillips Petroleum', NULL, 'fl
=from,to');
-[ RECORD 1 ]----------------------------------------------------------------------------------------------
--------------------------------------------------------------------------
from | Gerald Nemec
to   | Mark Whitt
-[ RECORD 2 ]----------------------------------------------------------------------------------------------
--------------------------------------------------------------------------
from | Gerald Nemec
to   | Mark Whitt
-[ RECORD 3 ]----------------------------------------------------------------------------------------------
--------------------------------------------------------------------------
from | Susan Scott
to   | Christine Stokes
```

- Use gptext_retrieve_field to fetch text value of returned fields.
- gptext_retrieve_field_int8 and gptext_retrieve_field_float8

STOP HERE

Stop here

Search engine index

- Stores data to enable fast and accurate information retrieval
- Useful to search large volumes of documents and retrieve relevant ones based on request
- Google and Bing have indexes of the web
- Indexing breaks up each document so that we know what word appears where in which document

Example:

- Index the articles in the table
- Provide full text search over the ‘content’ column

Question: Which document contains the word ‘send’?

Answer: Document with id = 2

id	creation_date	author	content
1	2012-02-15	John Smith	What a great ….
2	2013-12-31	Jane Doe	Please send the invoice asap…
..	…	…	…
1000000	2012-06-20	Frank Harris	When are the orders coming in…

Pivotal

Creating an index

- Load table to be indexed into database

```
CREATE TABLE articles
(
id bigint,
creation_date date,
author text,
content text,
)
distributed by (id);
```

id	creation_date	author	content
1	2012-02-15	John Smith	What a great ….
2	2013-12-31	Jane Doe	Please send the invoice asap…
..	…	…	…
1000000	2012-06-20	Frank Harris	When are the orders coming in…

- In practice, the table to be indexed can contain many more columns

Pivotal

Creating an empty index

- To create an empty GPText index

```
SELECT * FROM gptext.create_index(
              schema_name,       -- the schema of the table
              table_name,        -- the name of the table
              id_col_name,       -- the unique ID in the table
              default_search_col)-- the default column to index
```

- Example:

```
SELECT * FROM gptext.create_index(
              'public', 'articles', 'id', 'content');
```

This creates an index named <database_name>.<schema_name>.<table_name> on each segment

Pivotal

Enable terms

- Enabling terms provides access to document level statistics of term occurrences

```
SELECT * FROM gptext.create_index(
              'public', 'articles', 'id', 'content');

SELECT * FROM gptext.enable_terms(
                        'database.public.articles',
                        'content');

SELECT * FROM gptext.index(
              TABLE(SELECT * FROM articles),
              'database.public.articles');

SELECT * FROM gptext.commit_index('database.public.articles');
```

Enable terms for a specified field after index creation or before re-indexing

Pivotal

Enable terms

- Let's enable the terms capability

```
-- 1.2 Enable terms table
-- gptext.terms(src_table, index_name, field_name, search_query, filter_queries, options)
-- remember to substitute your two digit student ID for the XX in the index name

select
gptext.enable_terms('gptextdemo.twenty_news_groups','studentXX.gptextdemo.twenty_news_grou
ps','contents');
```

Enable terms for the 'content' field after index creation or before re-indexing. For this example, there is no 'search query' , 'filter queries', or 'options'

Pivotal

Populating an index

- To populate an empty GPText index

```
SELECT * FROM gptext.index(
             TABLE(SELECT * FROM table_name), -- table-valued expression
             index_name)                                  -- the name of
the index
```

- Note:
  - Arguments to gptext.index() must be expressions
  - `TABLE(SELECT * FROM table_name)` creates a “table-valued expression” from the articles table, using the table function TABLE.
  - Selective index/updates by changing the inner select list in the query
- Example:

```
SELECT * FROM gptext.index(
             TABLE(SELECT * FROM articles),
             ‘database.public.articles’);
```

Pivotal

Committing an index

- Make changes to an index persistent
- To commit a populated GPText index

```
SELECT * FROM gptext.commit_index(index_name)     -- the name of the index
```

- Note:
  - The index picks up any new data added since the last index commit when calling this function
  - Without committing, changes are not visible
- Example:

```
SELECT * FROM gptext.commit_index(
                        'database.public.articles);
```

Pivotal

GPText index functions

- Indexing
  - gptext.create_index()
  - gptext.enable_terms()
  - gptext.index()
  - gptext.commit_index()
- Modifying or deleting an index
  - gptext.add_field()
  - gptext.delete()
  - gptext.drop_field()
  - gptext.drop_index()
  - gptext.rollback_index()
- Configuration and monitoring
  - gptext.optimize_index()
  - gptext.reload_index()
  - gptext.index_statistics()

Pivotal

Searching a GPText index

- Retrieve relevancy ranked documents from an index

  Relevance score is determined by:

  - Term frequency – the frequency with which a term appears in a document
  - Inverse document frequency – the rarer a term is across all documents, the higher it's contribution
  - Coordination factor – the more query terms are found in a document, the higher it's score
  - Field length – penalizes longer documents
  - Boosts – assign more weight to specific terms

  More details about scoring can be found in the SOLR documentation: https://wiki.apache.org/solr/SolrRelevancyFAQ

- GPText uses the Unified Query Parser (QParserPlugin)
  - General purpose query parser with broad capabilities, but does not support span queries
  - GPText also supports: surround queries, complex phrase queries, and dismax queries (operator precedence)

Pivotal

Simple search query

```
SELECT c.id, c.author, c.content, q.score
FROM articles c,
     gptext.search(
                TABLE(SELECT * FROM articles),-- table-valued expression
                ‘database.public.articles’,  -- the name of the index
                ‘invoice OR orders’,         -- the query
                null                         -- no filter options
                )q
      where c.id = q.id;
```

Basic search term with boolean clauses

```
id       |  author     | content                    | score
---------+-------------+----------------------------+-------
 2       | Jane Doe    |Please send the invoice...  | 0.3256
 100000  | Frank Harris|When are the orders ...     | 0.2986
 5937    | Hank Brown  |Yesterday's invoice arr...  | 0.2832
 ...     |  ...        |     ...                    | ...
```

Pivotal

Surround query

```
SELECT c.id, c.author, q.score
FROM articles c,
     gptext.search(
                  TABLE(SELECT 1 SCATTER BY 1),     -- table-valued expression
                  'database.public.articles',       -- the name of the index
                  'mobi N web',                          -- the query
                  null                              -- no filter options
                  'defType=surround'                          -- change the
query parser
                 )q
      where c.id = q.id;
```

N represents a distance. You can prefix it with a number (i.e. 2N, 3N) to increase the distance between words. This looks for the two words at most a particular distance from one another. If you use a **W**, then it means that the left word must precede the right word by **W** distance.

Pivotal

Complex query

```
SELECT c.id, c.author, q.score
FROM articles c,
     gptext.search(
                  TABLE(SELECT 1 SCATTER BY 1),     -- table-valued expression
                  'database.public.articles',       -- the name of the index
                  '"mobi* web"',                         -- the query
                  null,                             -- no filter options
                  'defType=complex'                          -- change the
query parser
                 )q
      where c.id = q.id;
```

This is a wild card search. Looks for any regex pattern matching mobi* and web (i.e. mobile web, mobility web)

Pivotal

More complex query examples

- Boolean, wildcards, and distance queries
- Substitute the following examples for *the query* in the gptext_search function
    - ‘questions 5W answers 3W opinions’
    - ‘(post 5N modernism) AND (evil 2W good)’
    - ‘low W press*’
    - ‘5n(temp*at*, (matt* OR (strike* 3n price*)))’

Pivotal

Faceted search queries

- Faceting breaks up a search result into multiple categories and shows counts for each category.

  Searches can be faceted as follows:
  - faceted_field_search() facets by field
  - faceted_query_search() facets by query
  - faceted_range_search() facets by range

Facets by field query

```
SELECT *
FROM gptext.faceted_field_search(
                    'database.public.articles',     -- the name of the index
                    '*:*',                          -- the query, *:* matches all
                    null,                           -- no filter options
                    '{author}', -1, 5);
```

In this example, the query searches all articles in the data set by the *authors* field and retrieves only articles for authors who have created at least 5 articles.

```
 field_name | field_value | value_count
------------+-------------+-------------
 id         | Frank Harris|       52
 id         | Hank Brown  |       14
 id         | Jane Doe    |       6
 ...        | ...         |       ...
```

Pivotal

Highlighting search terms

- gptext.highlight() highlights search terms by inserting markup tags before and after each occurrence of the term in results returned by gptext.search()
- For example, if the search term is “iphone”, each occurrence of “iphone” in the data is marked up:

  iphone
- Useful when displaying search results on a webpage

```
SELECT  c.id,
        gptext.highlight(c.articles, ‘articles’, q.hs)
FROM articles c,
     gptext.search(
                  TABLE(SELECT 1 SCATTER BY 1),-- table-valued expression
                  index_name,                  -- the name of the index
                  ’iphone’,                    -- the query
                  null                         -- no filter options
                  ’hl=true&hl.fl=content’ -- SOLR options for highlighting
                  )q
      where c.id = q.id;
```

Pivotal

Lab: Searching a GPText Index

Lab: Searching a GPText Index

- Try a simple search query

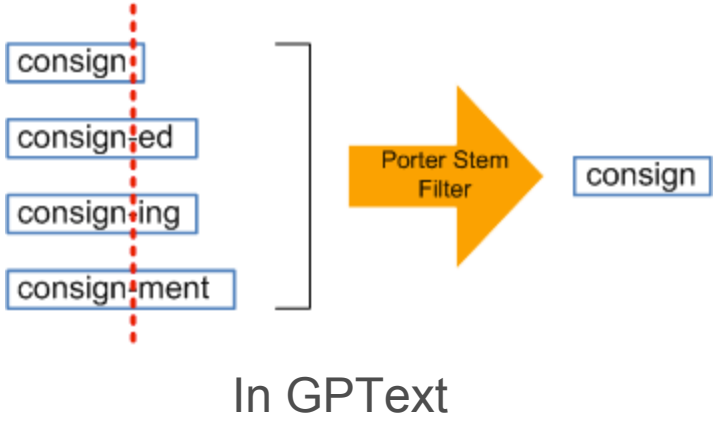
```
-- 1.5 Performing a simple query

-- gptext.search(src_tbl, index_name, search_query, filter_queries, options)

-- remember to substitute your two digit student id for the XX in the index name

select * from gptext.search(TABLE(select * from gptextdemo.twenty_news_groups),

'studentXX.gptextdemo.twenty_news_groups',

'mac',

null

) limit 10;
```

Pivotal

GPText Tokenizers and Analyzers

GPText analyzers – Getting ready for indexing

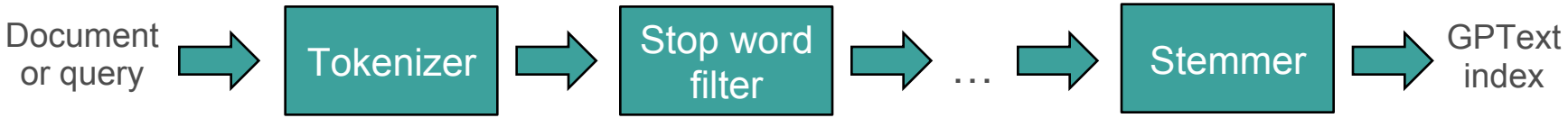
- Normalize words (e.g., convert to lower case)
  LOVE -> love
- Analyze numbers and punctuations for relevancy
  0-9, $#%@! ...
- Remove stop words
  you, me, we, us, in, on, to, here, there…
- Stemming
  Convert “running”, “runs” into “run”



In GPText

Analyzer chains

- An analyzer chain defines preprocessing sequences prior to indexing



- Two main analyzer chains
  - text_intl – For use with international text
    - WorldLexer Improvements
  - text_sm – For use with social media (e.g., Twitter, Facebook)
    - Support for emoticons

Note: Search terms in queries are processed by the analyzer chain defined for an index

Pivotal

Analyzer chain – text_intl

- WLT – World lexer tokenizer
  - Separates CJK characters from others and identifies symbols
- CJKWF – Chinese/Japanese/Korean word filter
  - Half/Full width characters
- LCF – Lower case filter
- WLBF – Word lexer bigram filter
  - Generates Bigrams for any CJK characters
    - Han, hiragana, katakana, hangul
- SF – Stop word filter
- KMF – Keyword marker filter
  - Marks words to prevent from being stemmed (protwords.txt)
- PSF – Porter stemmer filter

Pivotal

Analyzer chain – text_sm

- WT – Whitespace tokenizer
- SF – Stop word filter
- LCF – Lower case filter
- ECTF – Emoticons classifier filter
  - Uses the emoticons.txt file
- TTF – Twitter tokenizer
  - Extends whitespace tokenizer
  - Emoticons, hyperlinks, #hashtags, @users, etc.

Pivotal

Configuring GPText with gptext-config

- Allows editing the index configuration files:
  - schema.xml
  - stopwords.txt
  - protwords.txt
  - synonyms.txt
  - elevate.xml
  - emoticons.txt
- Distributes the specified configuration files to the specified directory. (-d parameter)
- Allows distribution of jar files. (-j parameter)
- Modifies jvm_options. (-o parameter)
- Appends a file to one of the configuration files. (-a parameter)
- Sets properties in the solr.xml configuration file (-k parameter)

Pivotal

Configuring analyzer chains with gptext-config

- Edit the schema.xml with gptext-config:
  - Use command: `gptext-config –f schema.xml –i <index_name> -e`
  - Find the XML line corresponding to indexed column of the database table (e.g., *content*)

```
<field name=”<text_search_col>" indexed="true" stored="false" type="text_intl"/>
```

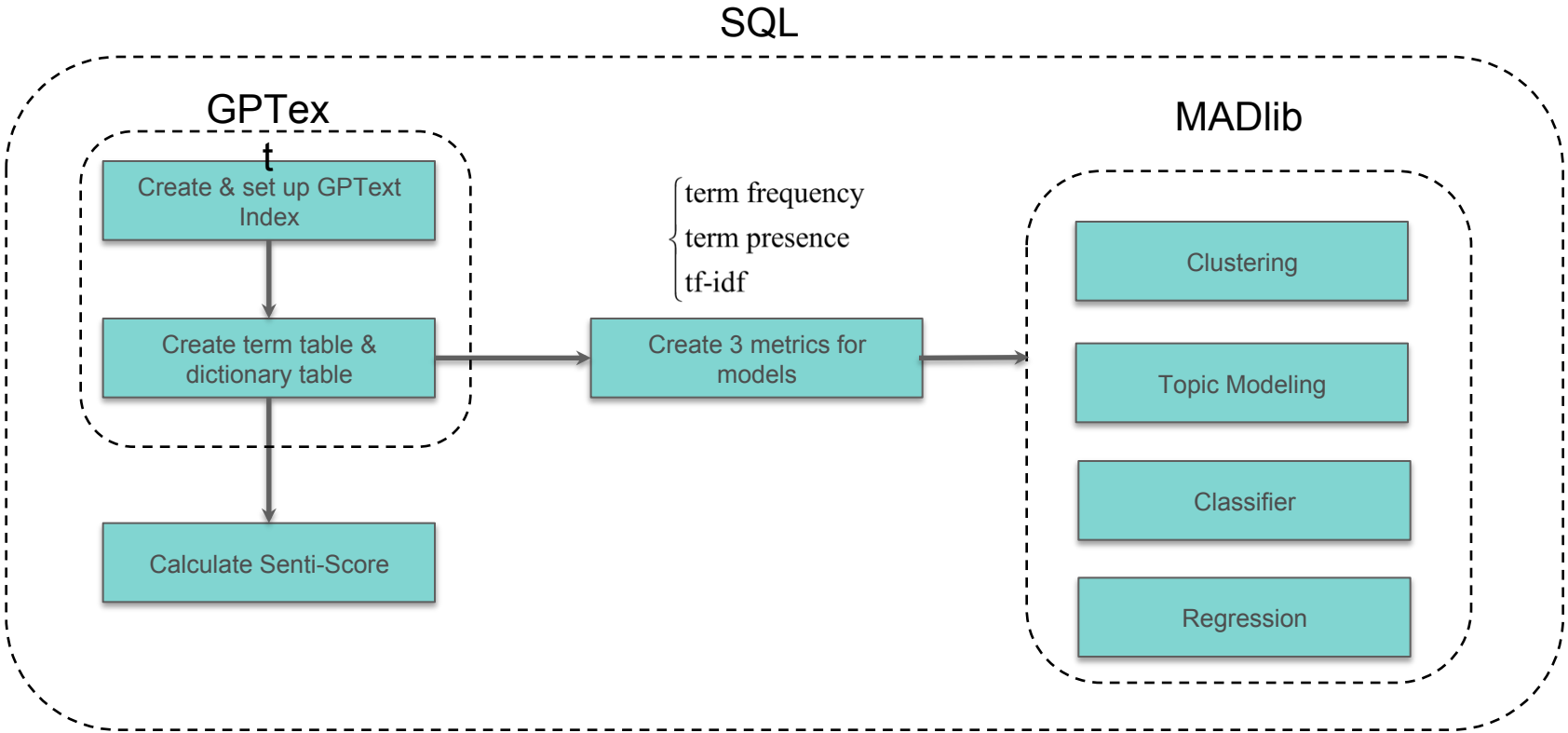
Edit type to either “text_intl” or “text_sm”

- Gptext-config syntax:
  - `gptext-config -f <file_name> -i <index_name> [-r] [-e] [-b]`
  - `gptext-config -d <path/file_name> -i <index_name>`
  - `gptext-config -j <path/jar_file>`
  - `gptext-config -o <jvm_options>`
  - `gptext-config -f <file_name> -i <index_name> -a <file_to_append>`
  - `gptext-config -k <solr_key> -s <solr_value>`

Pivotal

3 metrics to enable Analytics

Text Analytics Workflow



Enable terms (repeat for context)

- Enabling terms provides access to document level statistics of term occurrences

```
SELECT * FROM gptext.create_index(
              'public', 'articles', 'id', 'content');

SELECT * FROM gptext.enable_terms(
                        'database.public.articles',
                        'content');

SELECT * FROM gptext.index(
              TABLE(SELECT * FROM articles),
              'database.public.articles');

SELECT * FROM gptext.commit_index('database.public.articles');
```

Enable terms for a specified field after index creation or before re-indexing

Creating a terms table in the database

```
CREATE TABLE terms_table AS
    SELECT *
    FROM gptext.terms(TABLE(SELECT 1 SCATTER BY 1),
                      'database.public.articles',  -- the name of the index
                      'content',                   -- the field for which to extract
                                                     the terms (must have terms enabled)
                      '*',                         -- a search query
                      null);                       -- no filter query
```

id	term	positions
222	home	{16,20,22}
544	work	{26,39,58,236,246}
357	advanc	{26}
855	regularli	{167}
298	orlean	{168}
989	univ	{9,376}
609	perman	{209}
595	arithmet	{10}
742	amount	{113,137,247,365,457}
455	relat	{130}

Example terms table

- id … the unique id of a document
- term … a term appearing in the document
- positions … the token positions where the term appeared in the document

Pivotal

Data transformation – create corpus matrices

Term Frequency Matrix

	term1	term2	term3	term4	term5	term6	term7	term8	term9	term10	term11	term12	term13	term14
Doc1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Doc2	1	0	0	0	0	0	0	0	0	0	0	0	0	1
Doc3	0	2	0	0	1	0	1	0	0	0	0	0	0	0
Doc4	0	0	1	0	0	0	0	0	0	3	5	0	2	0
Doc5	0	0	0	2	0	0	2	0	0	0	0	0	0	0
Doc6	0	0	3	0	0	0	0	0	4	0	0	0	0	0
Doc7	0	0	0	0	0	0	0	0	0	0	0	2	0	0
Doc8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Doc9	0	0	0	0	0	0	0	0	0	0	0	0	0	3
Doc10	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The number of times term 13 appears in document 4

Term Presence Matrix

	term1	term2	term3	term4	term5	term6	term7	term8	term9	term10	term11	term12	term13	term14
Doc1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Doc2	1	0	0	0	0	0	0	0	0	0	0	0	0	1
Doc3	0	1	0	0	1	0	1	0	0	0	0	0	0	0
Doc4	0	0	1	0	0	0	0	0	0	1	1	0	1	0
Doc5	0	0	0	1	0	0	1	0	0	0	0	0	0	0
Doc6	0	0	1	0	0	0	0	0	1	0	0	0	0	0
Doc7	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Doc8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Doc9	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Doc10	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1: presence
0: absence

Pivotal

Data transformation – TF-IDF matrix

tf–idf = term frequency x inverse document frequency

	term1	term2	term3	term4	term5	term6	term7	term8	term9	term10	term11	term12	term13	term14
Doc1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Doc2	0.32	0	0	0	0	0	0	0	0	0	0	0	0	1
Doc3	0	0.53	0	0	0.21	0	0.24	0	0	0	0	0	0	0
Doc4	0	0	0.4	0	0	0	0	0	0	0.61	0.71	0	2.2	0
Doc5	0	0	0	3.43	0	0	0.32	0	0	0	0	0	0	0
Doc6	0	0	0.67	0	0	0	0	0	0.48	0	0	0	0	0
Doc7	0	0	0	0	0	0	0	0	0	0	0	1.1	0	0
Doc8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Doc9	0	0	0	0	0	0	0	0	0	0	0	0	0	0.21
Doc10	0	0	0	0	0	0	0	0	0	0	0	0	0	0

$$\mathrm{idf}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

total number of documents

the number of documents containing the term

- A high weight in tf–idf is reached by a high term frequency (in the given document) and a low document frequency of the term in the whole collection of documents
- A numerical statistic which reflects how important a word is to a document in a collection or corpus

Pivotal

Lab: 3 metrics

Pivotal

Lab: Creating the Terms table

- Create the terms table

```
-- 1.6 Create Terms Table
-- remember to substitute your two digit student id for the XX in the index name

drop table if exists gptextdemo.twenty_news_groups_terms cascade;
create table gptextdemo.twenty_news_groups_terms as
(
    select *
    -- gptext.terms(src_table, index_name, field_name, search_query, filter_queries, options)
    from gptext.terms(TABLE
    (
        select * from gptextdemo.twenty_news_groups),
        'studentXX.gptextdemo.twenty_news_groups','contents','*:*',null
    )
) distributed by (id);

-- now we should show the table so we can see the result
select * from gptextdemo.twenty_news_groups_terms limit 2;
```

Pivotal

Lab: Creating the term dictionary

- Create the term dictionary

```
-- 1.7 Create term dictionary
drop table if exists gptextdemo.twenty_news_groups_terms_dict cascade;
create table gptextdemo.twenty_news_groups_terms_dict as
(
    select row_number() over(order by term asc) as idx,
           term,
           term_freq
    from
    ( select term,
             count (*) as term_freq
        from gptextdemo.twenty_news_groups_terms
        where term is not null
        group by term
    ) q where term_freq > 50 -- disregard terms which occur less than 50 times at least (long tail).
) distributed by (idx);

select * from gptextdemo.twenty_news_groups_terms_dict limit 10;
```

Pivotal

Lab: Create the sparse vectors

- Create a sparse vector with td-idf scores for the corpus

```
-- 1.8 Create sparse vectors representation using tf-idf scores for the corpus
drop table if exists gptextdemo.twenty_news_groups_corpus cascade;
create table gptextdemo.twenty_news_groups_corpus as (
    select terms.id,
           gptext.gen_float8_vec
           -- gptext.gen_float8_vec(index_array, count_array, vector_size)
           (
               array_agg(dict.idx),                          -- Array of indices
               array_agg(array_upper(terms.positions, 1)), -- Array of values at indices
               -- gptext.count_t(table_name)
               gptext.count_t('gptextdemo.twenty_news_groups_terms_dict') --# terms in dictionary
           )::madlib.svec sfv
    from   gptextdemo.twenty_news_groups_terms terms, gptextdemo.twenty_news_groups_terms_dict dict
    where terms.term = dict.term
    group by terms.id
) distributed by (id);
```

Lab: Create the tf-idf feature vector

- Create the tf-idf feature vector table

```
-- 1.9 TF-IDF feature vector
drop table if exists gptextdemo.twenty_news_groups_fvect_tfidf;
create table gptextdemo.twenty_news_groups_fvect_tfidf as
            (select
                        id as doc_id,
                        madlib.svec_mult( sfv, logidf ) as tf_idf
                        from gptextdemo.twenty_news_groups_corpus,
                        (select madlib.svec_log
                                    (madlib.svec_div
                                                (count(sfv)::madlib.svec,
                                                madlib.svec_count_nonzero(sfv)
                                                )
                                    ) as logidf
                                    from gptextdemo.twenty_news_groups_corpus
                        ) foo
                        order by doc_id
            ) distributed by (doc_id);

\x
select * from gptextdemo.twenty_news_groups_fvect_tfidf limit 1;
```

NLP: Practical Examples

Common NLP tasks

NLP Task	Example
Sentence Segmentation *Identify sentence boundaries*	He said: “Hi! What’s up - Mr. President?”
Tokenization *Identify word/token boundaries*	My phone tries to change 'eating' to 'dating’. #ihateautocorrect
Part-of-Speech Tagging	If you build it , he will come . IN PRP VBP PRP , PRP MD VB .
Stemming / lemmatization	eating = eat / ate = eat
Parsing	(S (NP (NP John) and (NP Frank)) (VP went (PP into (NP a bar))))
Named Entity Recognition	Let’s meet John in DC at 6pm.
Co-reference resolution	John drank a beer. He thought It was warm.

Pivotal

Regular expressions (Regex)

- What are regular expressions?
  - Wikipedia: “Provide means for matching strings of text”
  - CS lecture: Specify languages that are accepted by finite state machines
- In NLP regex are often used to derive features
  - Word is all caps
  - First letter of word is lower/upper case
  - Token contains numbers and letters (e.g., biological entities SNP76325)
  - Token contains special characters
- Examples:
  - Syntactically correct email addresses
  - Convert 9 digits into format “(123) 456-7890”

Pivotal

Regex in GPDB

- Operators (~, ~*, and negated versions !~, !~*):

Operator	Description	Example
~	case-sensitive substring	'Greenplum' ~ '^Green'
~*	case-insensitive substring	'Greenplum' ~ 'ee+'

- Functions
  - substring(*string* [from *pattern*] [for *escape*])
  - regexp_matches(*string*, *pattern*, [*flags*])
  - regexp_replace(*string*, *pattern*, *repl*, [*flags*])
  - regexp_split_to_{array|table}

- Regular expression functionality is also available via custom PL functions
  - Python module '`re`'
  - Java module '`util.regex`'

Pivotal

Regular expression syntax

- Basic meta characters

Expression	Matches
.	Arbitrary character
^ and $	Virtual characters for beginning and end
*	Preceding item zero or more times
+	Preceding item one or more times
?	Preceding item is optional
{*n*}	Preceding item *n* times
a|*b*	Item *a* or *b*
…	…

- See PostgreSQL website

Pivotal

N-grams

- Often used in Markov models
- Higher order n-grams preserve context in which words appear
- Example:

"The quick brown fox jumped over the lazy dog."

Unigram:	**(The), (quick), (brown), ...**
Bigram:	(The, quick), (quick, brown), (brown, fox), …
Trigram:	(The, quick, brown), (quick, brown, fox), (brown, fox, jumps), …
4-gram:	(The, quick, brown, fox), (quick, brown, fox, jumps), (brown, fox, jumps, over), …

Pivotal

PL/Python to extract n-grams

```
/* Extract n-grams from a text field */

CREATE OR REPLACE FUNCTION extract_ngrams(textfield text, min_n integer, max_n integer)
  RETURNS text[] AS  $BODY$
if textfield == None:
                    return None
tokens = textfield.split(" ") # tokenize by whitespace
list = []
n_tokens = len(tokens)
for i in xrange(n_tokens):
                    for j in xrange(i+min_n, min(n_tokens, i+max_n)+1):
                                        list.append(' '.join(tokens[i:j]))
return list
$BODY$ LANGUAGE plpythonu IMMUTABLE;
```

- Example:

```
SELECT extract_ngrams('The quick brown fox jumped over the lazy dog', 2, 2);
```

Result: {"The quick","quick brown","brown fox","fox jumped","jumped over","over the","the lazy","lazy dog"}

Pivotal

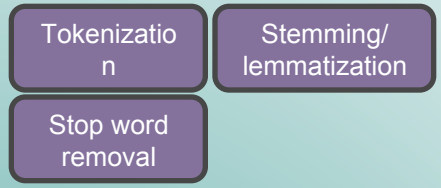
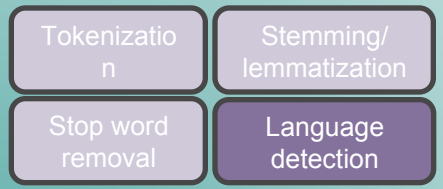
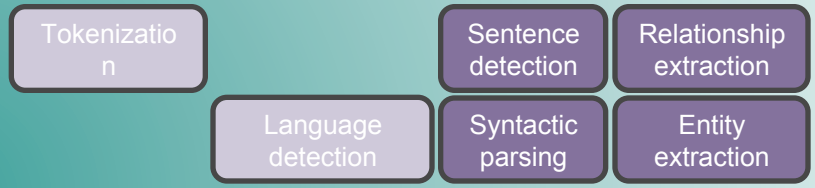
Utilizing NLP packages via PL languages

- Languages
  - Python
  - R
  - Java
  - PGSQL

- Extend database functions with open source packages
  - NLTK (Python)
  - TM (R)
  - OpenNLP (Java)
- Many NLP tasks are *embarrassingly* parallel
  - PoS tagging
  - Parsing

Pivotal

Open source tools for common NLP tasks

RELEVANT NLP TOOLS	OPEN SOURCE SOFTWARE
WORD CLOUDS 	• GPText • Apache UIMA • OpenNLP (Java)
TOPIC MODELING/TEXT CLASSIFICATION 	• NLTK (Python) • WordNet • Madlib (PLDA) • Pytagcloud • gensim (LSA & LDA package for python) • https://code.google.com/p/language-detection/
INFORMATION EXTRACTION 	• GPText and Madlib • OpenNLP • NLTK • Stanford CoreNLP (incl. POS tagger, NER, parser, etc.)

NLP: Practical examples with NLTK

Lemmatization using NLTK

- Identify the lemma (citation form) of a word
- Why:
  - Stemmers only stem
    eating = eat (rule: remove 'ing')
  - They do not catch: ate = eat
  - Or better
- Example:
  - ate = eats = eating → eat
  - good = better → good
- Prerequisites
  - Access to WordNet dictionary files

```
/* Lemmatize, but only look for
 * lemmas with the given Part-of-Speech */

CREATE OR REPLACE FUNCTION lemmatize(word text, pos text)
  RETURNS text AS
$BODY$
  import nltk
  from nltk.stem.wordnet import WordNetLemmatizer

  lemmatizer = WordNetLemmatizer()
  return lemmatizer.lemmatize(word, pos)
$BODY$
  LANGUAGE plpythonu;
```

- Example:

```
SELECT lemmatize('meeting', 'n')
```

Result: 'meeting'

```
SELECT lemmatize('meeting', 'v')
```

Result: 'meet'

Pivotal

Part-of-Speech tagging

- Assign each word with its part of speech, e.g., noun, adjective, verb, …
- Can be used as features, to distinguish homonyms, or lemmatization

- Example:

```
SELECT pos_tag('We were meeting at the meeting.', 'n')
```

Result:

```
[('We', 'PRP'),
 ('were', 'VBD'),
 ('meeting', 'VBG'),
 ('at', 'IN'),
 ('the', 'DT'),
 ('meeting', 'NN'),
 ('.', '.')]
```

```
/**
 * Part-of-speech tag the text in the textfield
 */
CREATE OR REPLACE FUNCTION pos_tag(textfield text)
  RETURNS text[] AS
$BODY$
  import nltk

  tokens = nltk.word_tokenize(textfield)
  return nltk.pos_tag(tokens)
$BODY$
  LANGUAGE plpythonu;
```

Pivotal

Named entity recognition

- Recognize entities in text and label them
  - LOCATION, ORGANIZATION, PERSON, DURATION, DATE, CARDINAL, PERCENT, MONEY, and MEASURE

- Example:

```
SELECT ner(’ John will arrive in New York at 9pm.’)
```

Result: Tree('S', [Tree('PERSON', [('John', 'NNP')]), ('will', 'MD'), ('arrive', 'VB'), ('in', 'IN'), Tree('GPE', [('New', 'NNP'), ('York', 'NNP')]), ('at', 'IN'), ('9pm', 'CD'), ('.', '.')])

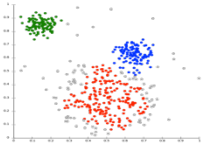
```
/**
 * Recognize NEs in the given textfield
 */
CREATE OR REPLACE FUNCTION ner(textfield text)
  RETURNS text AS
$BODY$
  import nltk

  tokens = nltk.word_tokenize(textfield)
  pos_tags = nltk.pos_tag(tokens)
  return nltk.ne_chuck(pos_tags)
$BODY$
  LANGUAGE plpythonu;
```

Putting it all together

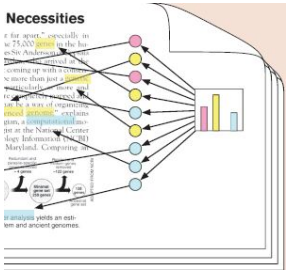
Pivotal

What's possible with text analytics in GPDB

- Document clustering

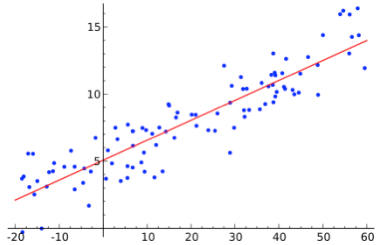


- Topic modeling



- Document classification

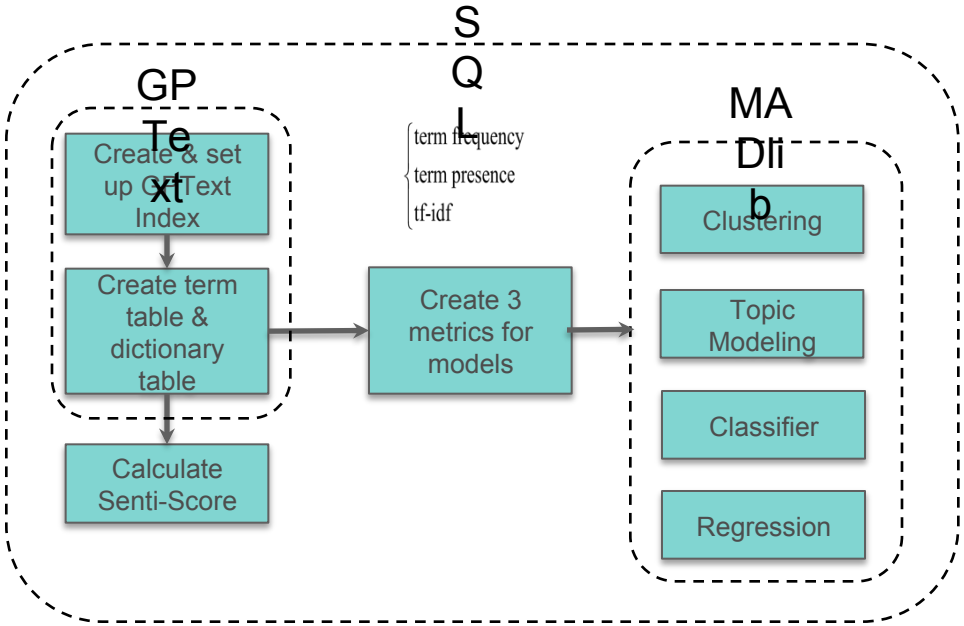
- Sentiment analysis



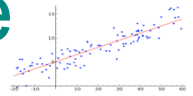
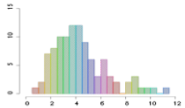
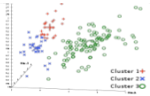
Pivotal

GPText integration with MADlib

- Vector space representation of documents in GPText lends itself to machine learning analysis with MADlib
- Algorithms in MADlib operating on vectors
  - Support vector machines
  - Kmeans clustering
  - Conditional random fields
  - Parallel latent dirichlet allocation
- Supporting data types in MADlib
  - Sparse vectors – are useful in representing word level features



MADlib In-Database Functions

Predictive Modeling Library

Generalized Linear Models
- Linear Regression
- Logistic Regression
- Multinomial Logistic Regression
- Cox Proportional Hazards
- Regression
- Elastic Net Regularization
- Sandwich Estimators (Huber white, clustered, marginal effects)

Machine Learning Algorithms
- Principal Component Analysis (PCA)
- Association Rules (Affinity Analysis, Market Basket)
- Topic Modeling (Parallel LDA)
- Decision Trees
- Ensemble Learners (Random Forests)
- Support Vector Machines
- Conditional Random Field (CRF)
- Clustering (K-means)
- Cross Validation

Matrix Factorization
- Single Value Decomposition (SVD)
- Low-Rank

Linear Systems
- Sparse and Dense Solvers

Descriptive Statistics

Sketch-based Estimators
- CountMin (Cormode-Muthukrishnan)
- FM (Flajolet-Martin)
- MFV (Most Frequent Values)

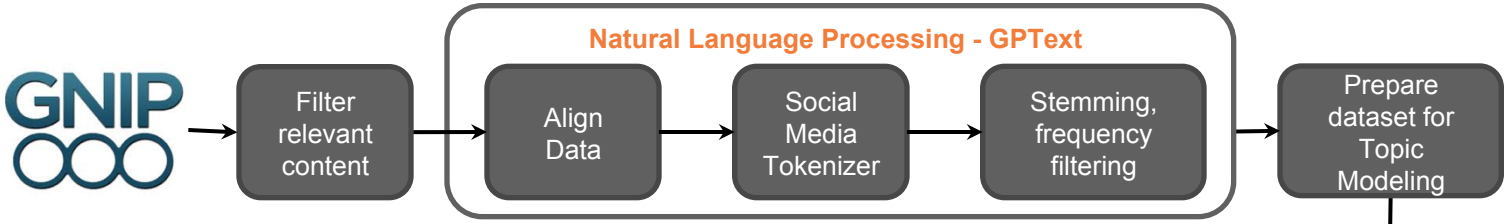
Correlation
Summary

Support Modules

Array Operations
Sparse Vectors
Random Sampling
Probability Functions

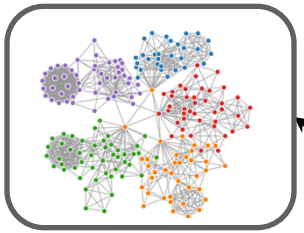
Pivotal

Topic analysis – MADlib pLDA



id integer	contents bigint[]
103	{12858,45693,40752,36663,13146,488,11354,20102,14789,
175	{29944,6810,40842,40842,4831,32610,13146,18631,41707,
625	{13763,13763,13763,42635,30637,24431,24431,9745,9745,

Topic Graph



Topic Clouds

Topic composition

id integer	topic_probabilities numeric[]	label integer
120103	{0.00,0.00,0.75,0.20,0.05}	-1
239097	{0.06,0.26,0.00,0.59,0.09}	1
278079	{0.04,0.13,0.05,0.21,0.57}	1
327383	{0.00,0.05,0.00,0.95,0.00}	-1

MADlib Topic Model

Pivotal

Predicting Commodity Futures through Twitter

Customer

A major a agri-business cooperative

Business Problem

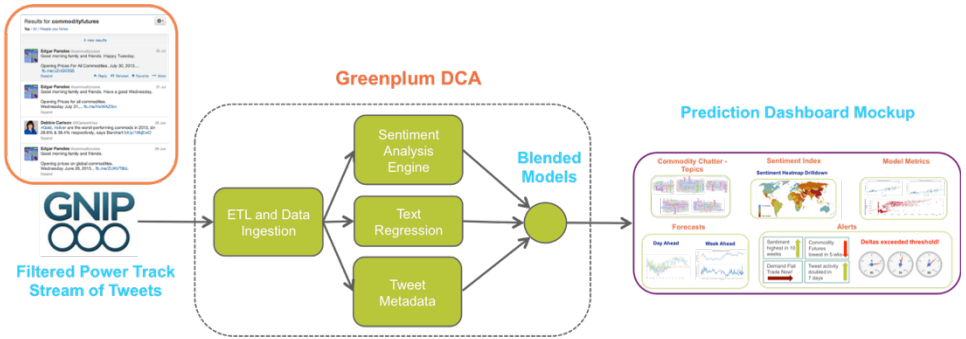
Predict price of commodity futures through Twitter

Challenges

- Language on Twitter does not adhere to rules of grammar and has poor structure
- No domain specific label corpus of tweet sentiment – problem is semi-supervised

Solution

- Built Sentiment Analysis and Text Regression algorithms to predict commodity futures from Tweets
- Established the foundation for blending the structured data (market fundamentals) with unstructured data (tweets)



Pivotal

Text Analytics for Churn Prediction

Customer

A major telecom company

Business Problem

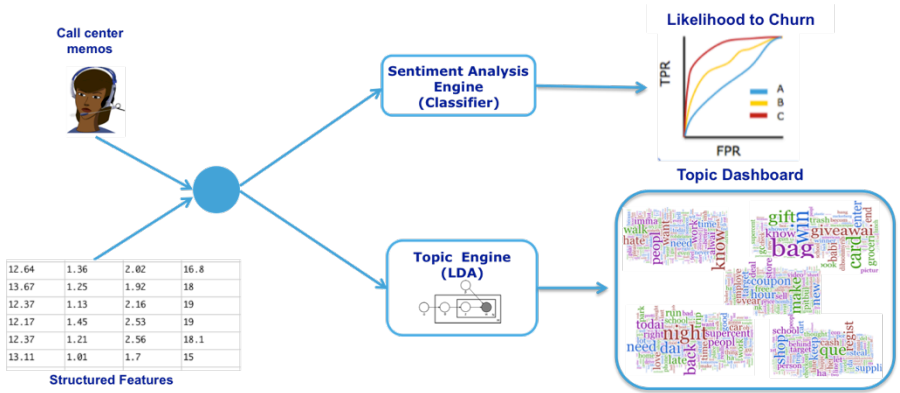
Reducing churn through more accurate models

Challenges

- Existing models only used structured features
- Call center memos had poor structure and had lots of typos

Solution

- Built sentiment analysis models to predict churn and topic models to understand topics of conversation in call center memos
- Achieved 16% improve in ROC curve for Churn Prediction



Pivotal

Website Classification

Customer

An internet domain name service provider

Business Problem

Create a multilevel website classification that groups websites by function rather than topic

Challenges

- Complex unstructured data format required several transformations
- Model needed to be language independent, so classic language features could not be used

Solution

- New hierarchical model resulted in reducing the number of previously ‘unclassified’ websites by ~75%
- Created an in-database analytics framework for unsupervised learning models and enabled real-time validation of current production model

Map of Domains

Lab: Analytics with MADlib

Pivotal

Lab: Create a complete feature vector

- Now that we have the basic GPText structures in place, we need to create a training data set and a test data set. First we have to create a table to hold all the feature vectors

```
-- 2.0 Create Train & Test sets, to build a predictive model using SVM

drop table if exists gptextdemo.twenty_news_groups_fvect_tfidf_all cascade;
create table gptextdemo.twenty_news_groups_fvect_tfidf_all as
  (select t1.doc_id as id,
    madlib.svec_return_array(t1.tf_idf) as ind,random() as rand_num,
    -- We will detect if a news group discussion is about computers or not
    case when t2.label in
('comp.graphics','comp.os.mswindows.misc','comp.sys.ibm.pc.hardware','comp.windows.x')
    then 1 else -1
    end as label
  from gptextdemo.twenty_news_groups_fvect_tfidf t1, gptextdemo.twenty_news_groups t2
  where t1.doc_id = t2.doc_id
  ) distributed by (id);
\x
select * from gptextdemo.twenty_news_groups_fvect_tfidf_all limit 1;
```

Pivotal

Lab: Create a training data set

- Now let us create the training data set

```
-- 2.1 Training set
-- 2445 CS related, 9530 non-CS related (Roughly 20% are CS related)
drop table if exists gptextdemo.twenty_news_groups_fvect_tfidf_train cascade;
create table gptextdemo.twenty_news_groups_fvect_tfidf_train as
            (
                        select * from gptextdemo.twenty_news_groups_fvect_tfidf_all
                        where rand_num < 0.60
            ) distributed by (id);
```

Pivotal

Lab: Create a test data set

- Now let us create the test data set

```
-- 2.2 Test set
-- 1555 CS related, 6467 non-CS related
drop table if exists gptextdemo.twenty_news_groups_fvect_tfidf_test cascade;
create table gptextdemo.twenty_news_groups_fvect_tfidf_test as
            (
                        select * from gptextdemo.twenty_news_groups_fvect_tfidf_all
                        where rand_num > 0.60
            ) distributed by (id);
```

Pivotal

Lab: Now we have to actually train the model

- Now train the model using MADlib's lsvm_classification algorithm
- Support Vector Machines - Classification Learning Function

```
-- 2.3 Linear SVM Classifier training
drop table if exists gptextdemo.svm_mdl_newsgrps cascade ;
drop table if exists gptextdemo.svm_mdl_newsgrps_param cascade ;
select madlib.lsvm_classification('gptextdemo.twenty_news_groups_fvect_tfidf_train',
'gptextdemo.svm_mdl_newsgrps',
false
);
```

Pivotal

Lab: Now let us do some prediction

- Now let's do some prediction using the model and MADlib's lsvm_predict_batch algorithm
- Support Vector Machines - Prediction Function

```
-- 2.4 Linear SVM prediction
drop table if exists gptextdemo.twenty_news_groups_svm_predictions cascade;
select madlib.lsvm_predict_batch(
'gptextdemo.twenty_news_groups_fvect_tfidf_test',
'ind',
'id',
'gptextdemo.svm_mdl_newsgrps',
'gptextdemo.twenty_news_groups_svm_predictions',
false
);
```

Pivotal

Lab: Now let us show the prediction accuracy

- Now let's show the accuracy of the model (should be around 82% - 85%, which is fairly decent)

```
-- Show SVM prediction accuracy
-- 82% accuracy
select 'svm' as model, sum(is_correct)*100.0/sum(dummy) as precision
from
( -- Verify if the predicted label is equal to the actual label
  select id, case when actual_label = predicted_label then 1 else 0 end as is_correct, 1 as dummy
  from
  (
  -- Show id, actual label & predicted label
    select t1.id, t1.label as actual_label,
    case when t2.prediction > 0 then 1 else -1 end as predicted_label
    from gptextdemo.twenty_news_groups_fvect_tfidf_test t1, gptextdemo.twenty_news_groups_svm_predictions t2
    where t1.id = t2.id
  ) q1
) q2;
```

Lab 5

Search Queries

Faceted Search Query

Faceted Field Search:

```
SELECT * FROM gptext.faceted_field_search(
  idx_name,
  search_query,
  filter_queries,
  facet_fields,
  facet_limit,
  minimum
  );
```

idx_name, search_query and filter_queries are as same as that in search.

- facet_fields - what fields faceted results are grouped by
- facet_limit - the number of results to be returned
- minimum - only results with count number >= minimum to be returned

Faceted Field Search Example

```
demo=# select * from gptext.faceted_field_search('demo.public.enron', 'content:Phillips Petroleu
m', NULL, '{from, to}', 4, 0);
-[ RECORD 1 ]----------
field_name  | from
field_value | phillip
value_count | 190
-[ RECORD 2 ]----------
field_name  | from
field_value | enron
value_count | 120
-[ RECORD 3 ]----------
field_name  | from
field_value | allen
value_count | 118
-[ RECORD 4 ]----------
field_name  | from
field_value | k
value_count | 118
-[ RECORD 5 ]----------
field_name  | to
field_value | enron.com
value_count | 115
```

- Get the aggregation number of each group of field given by fields array

Faceted Search Query

Faceted Query Search:

```
SELECT * FROM gptext.faceted_query_search(
  idx_name,
  search_query,
  filter_queries,
  facet_queries
  );
```

idx_name, search_query and filter_queries are as same as that in search.

- facet_queries - what queries faceted results are grouped by

Faceted Query Search

```
demo=# select * from gptext.faceted_query_search('demo.public.enron', '*:*', NULL, array['date:[
2000-05-22T00:00:00Z TO NOW]', 'from:"Kaminski, Vince"']);
-[ RECORD 1 ]---------------------------------
query_name  | date:[2000-05-22T00:00:00Z TO NOW]
value_count | 43858
-[ RECORD 2 ]---------------------------------
query_name  | from:"Kaminski, Vince"
value_count | 259
```

- Specify multi queries in queries array to get multi aggregation

Faceted Search Query

Faceted Query Search:

```
SELECT * FROM gptext.faceted_range_search(
  idx_name,
  search_query,
  filter_queries,
  range_start,
  range_end,
  range_gap
  );
```

idx_name, search_query and filter_queries are as same as that in search.

- range_start, range_end, range_gap - used to generated range intervals by which facted results are grouped

Faceted Range Search

```
demo=# select * from gptext.faceted_range_search('demo.public.enron', '*:*', NULL, 'date', '2000
-05-22T00:00:00Z', '2012-05-22T00:00:00Z', '+1YEAR');
-[ RECORD 1 ]--------------------
field_name  | date
range_value | 2000-05-22T00:00:00Z
value_count | 31486
-[ RECORD 2 ]--------------------
field_name  | date
range_value | 2001-05-22T00:00:00Z
value_count | 12275
-[ RECORD 3 ]--------------------
field_name  | date
range_value | 2002-05-22T00:00:00Z
value_count | 94
-[ RECORD 4 ]--------------------
```

- Get the aggregation number of each group generated by start, end and gap.

Search Count

```
SELECT * FROM gptext.search_count(
  TABLE( SELECT 1 SCATTER BY 1),
  idx_name,
  search_query,
  filter_queries
  );
```

- Lightweight search query
- Used when you only care about the number of documents that match
- Useful to gauge approximate return set of a given query
- Much faster than a search because no results are returned from the indexes

GPText + MADLib

Madlib - GPDB In-database Analytics

MADlib Functions

Predictive Analytics Library

Supervised Learning

Regression Models
- Cox Proportional Hazards Regression
- Elastic Net Regularization
- Generalized Linear Models
- Linear Regression
- Logistic Regression
- Marginal Effects
- Multinomial Regression
- Ordinal Regression
- Robust Variance, Clustered Variance
- Support Vector Machines

Tree Methods
- Decision Tree
- Random Forest

Other Methods
- Conditional Random Field
- Naïve Bayes

Unsupervised Learning
- Association Rules (Apriori)
- Clustering (K-means)
- Topic Modeling (LDA)

Time Series
- ARIMA

Model Evaluation
- Cross Validation

Other Modules
- Conjugate Gradient
- Linear Solvers
- PMML Export
- Random Sampling
- Term Frequency for Text

Data Types and Transformations
- Array Operations
- Dimensionality Reduction (PCA)
- Encoding Categorical Variables
- Matrix Operations
- Matrix Factorization (SVD, Low Rank)
- Norms and Distance Functions
- Sparse Vectors

Statistics

Descriptive
- Cardinality Estimators
- Correlation
- Summary

Inferential
- Hypothesis Tests

Other Statistics
- Probability Functions

Pivotal

Term Vectors

Populating Term Vectors in Database:

```
SELECT * FROM gptext.terms(
  TABLE(SELECT 1 SCATTER BY 1),
  idx_name,
  field_name,
  search_query,
  filter_queries,
  options
  );
```

- Extract term vectors from Solr
- Terms switch on by gptext.enable_terms(idx_name, field_name) before indexing
- Preserve all works done during index time analysis
- Term vectors used for further in-database analysis
- Term vectors can be selectively extracted by using <search_query> and <filter_queries>

Term Vectors

Term extract example:

Original Tweet: Finally got an iPhone 4S

```
select * from gptext.terms( TABLE( select 1 scatter by 1), 'demo.twitter.message', 'message_text',
'id:199206', null, null );
   id    | term   | positions
---------+--------+-----------
 199206 | 4s          | {4}
 199206 | final | {0}
 199206 | got    | {1}
 199206 | iphon | {3}
(4 rows)
```

- Term Vectors created in Solr at index time
- Selectively extract the terms through the database at any point
- Stop words, protected words, stemming, etc. all applied

Term Vectors

Creating a terms table:

```
CREATE TABLE myterms AS
SELECT id AS doc_id, term, positions FROM gptext.terms(
  TABLE(SELECT 1 SCATTER BY 1),
  <my_schema>.<my_table>,
  <my_field>,
  '*:*',
  'lang:eng',
  'rows=50'
  );
```

Term Vectors

Creating a dictionary from term vectors:

```
CREATE TABLE dict AS
SELECT
    row_number() OVER( ORDER BY term ASC ) term_id,
        term,
FROM <mySchema.terms>
WHERE term IS NOT NULL
GROUP BY term
```

Madlib K-means

Creating a tf-idf table from dict:

```
SELECT madlib.gen_doc_svecs(
    'tfidf', -- output table
    'dict', 'term_id', 'term', -- term dictionary
    'myterms', 'doc_id', 'term', 'positions' -- document terms);
```

K-means algorithm:

```
SELECT madlib.kmeanspp(
    'tfidf', 'sparse_vector',
    <k_number>,
    <madlib_distance_norm>,
    <madlib_average_func>,
    ....);
```