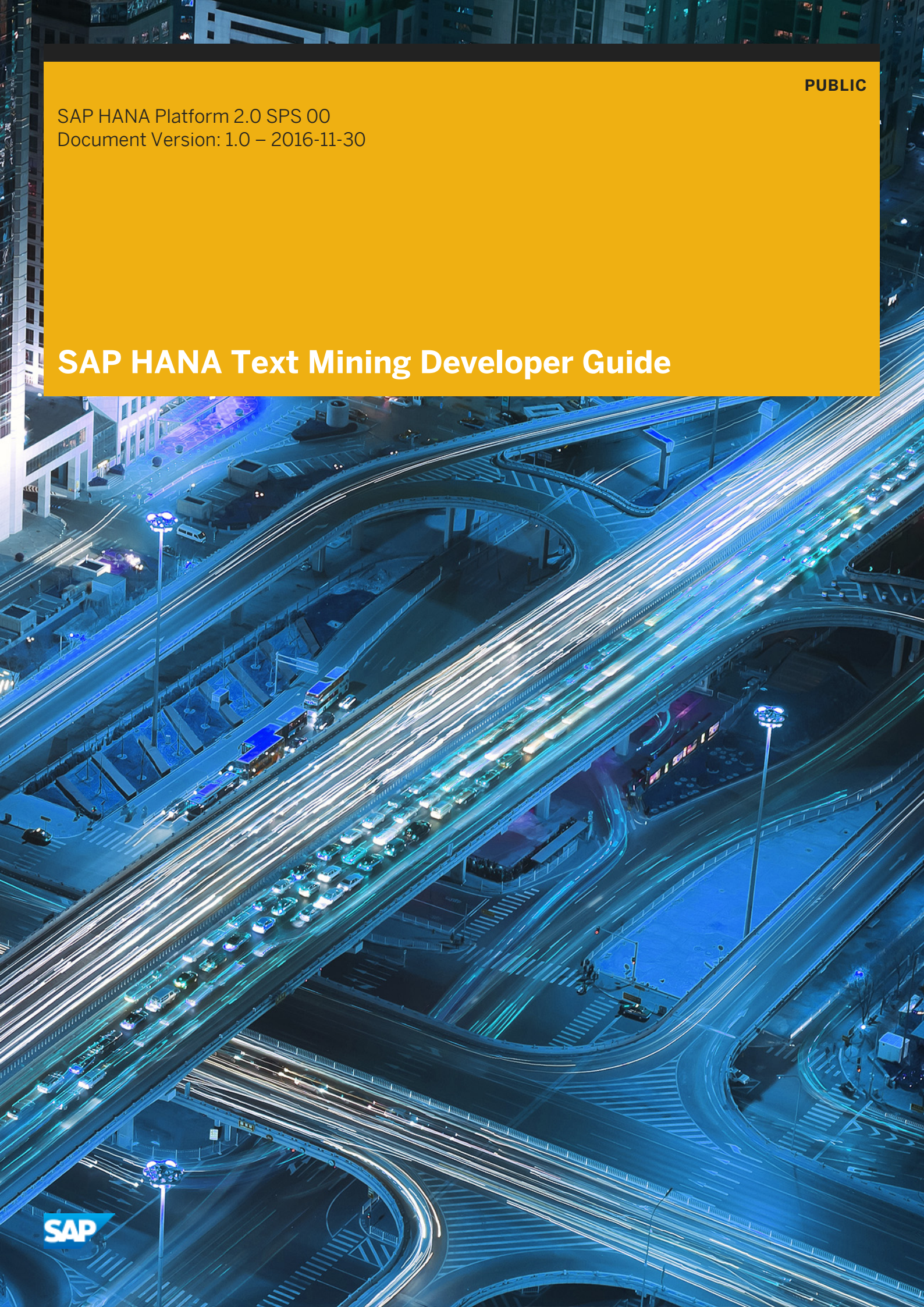
PUBLIC

SAP HANA Platform 2.0 SPS 00
Document Version: 1.0 – 2016-11-30

SAP HANA Text Mining Developer Guide



Content

1 What is Text Mining?. 3

2 Terms and Definitions. 4

2.1 Bag of Terms. 4

2.2 Relative Comparison. 4

2.3 Term Weights. 4

2.4 Term-Document Matrix 5

2.5 Similarity. 5

3 Text Mining Functions. 6

3.1 Initialization. 6

3.2 Term-Document Functions. 7

3.3 Categorization. 7

3.4 Statistical Analysis. 8

4 Configuring Text Mining 9

4.1 Configuration File Syntax. 9

4.2 Configuration Properties. 10

5 Using Text Mining. 13

5.1 General Requirements. 13

5.2 Using Different User Credentials. 14

5.3 Text Mining XS Classic API Example Application. 15

5.4 Text Mining XS Advanced API Example Application. 17

5.5 Text Mining SQL API Example Application. 19

5.6 The Public Text Mining Data Table Interface. 21

1 What is Text Mining?

Text mining provides functionality that can compare documents by examining the terms used within them.

Text mining supports functions to determine the top-ranked related and relevant documents and terms relative to a given reference document set and perform statistical analysis. It also supports methods for categorizing (classifying) documents, given a set of reference documents with predefined categorizations.

Text mining works at the document level – it makes semantic determinations about the overall content of documents relative to other documents. This is in contrast to text analysis, which does linguistic analysis and extracts information embedded within each document. They are complementary. Text mining benefits by using the information determined by text analysis.

Text mining can be used to analyze data at will in various selected database tables after it has been entered.

The principal envisioned use cases are for categorization of a stream of input documents, e.g. news articles or customer support incidents. The other term-document functions and statistical results will also be needed for further analysis to tune accuracy and maintain the set of reference documents.

Other use cases may be search applications that wish to find related documents and terms or suggest search terms based on an initial substring.

2 Terms and Definitions

2.1 Bag of Terms

Text mining compares documents using a bag-of-terms model.

This means that only the number of occurrences of terms is taken into account, not the order in which they appear. The nature of a document's contents is defined by the relative frequencies (counts) of the terms it uses.

A term is more than just a word. Text mining benefits from text analysis to extend the definition of what a term is. Text analysis preprocessing allows terms to be based on the normalized stems of the original surface tokens, coupled with their parts of speech. Phrases containing multiple words are also used as terms.

2.2 Relative Comparison

Comparison is relative to a given set of documents.

A document is not evaluated by itself in a stand-alone manner. It is compared to other documents in a given reference set . This allows text mining to automatically determine how relevant/salient each term is for differentiating that document from the others. The user does not need to develop a list of key terms manually. This means that document similarity and other text mining results depend on the other documents present in the set rather than being absolute.

The reference set of documents is typically an entire table, but it can also be a selected part of a table. The input document may itself be one of the reference documents or not, and it may be separate from the table.

2.3 Term Weights

Text mining assigns a weight to each term in a document to represent its importance. It can consider the number of occurrences in the document and also in the other reference documents. This allows it to discount terms that occur in many of the other documents.

The following term weights are commonly used in the industry:

Table 1:

Term Weight	Description
TF (term frequency)	Number of occurrences of term in document
IDF (inverse document frequency)	`log(D / DF)`, where `D` = total number of documents and `DF` (document frequency) is the number of documents in which term occurs
TF-IDF (term frequency × inverse document frequency)	`TF × IDF`
LTF-IDF (log term frequency × inverse document frequency)	`log(TF + 1) × IDF`

Text mining supports a generalized approach `tfWeighting(TF) × idfWeighting(DF,D)`, where `tfWeighting` and `idfWeighting` are selectable from a defined set of functions. This allows the above term weights plus some additional possibilities. The default is TF-IDF, the most commonly used term weight.

2.4 Term-Document Matrix

Text mining maintains a Term-Document matrix for the set of documents it has analyzed, where each row is a document, each column is one of the N terms, and each cell represents a term weight.

2.5 Similarity

Text mining compares the similarity of two documents by comparing the relative weights of the terms they contain.

The comparison is done using an N-dimensional vector space model, where each of the N terms in the term vocabulary (the union of all documents) is another orthogonal axis. A given document can be understood as the vector formed in this space where the length along each term's axis is the term weight for that term in the document.

Text mining supports several standard similarity measures. The most commonly used one is cosine similarity. The cosine similarity of two documents is related to how much their vectors point in the same direction. More precisely, it is the cosine of the angle between the vectors, since that is the scalar projection or component of one's unit vector (length normalized to 1) that goes in the direction of the other.

Text mining also supports three other standard similarity measures, which differ in the treatment of matching vs. differing term weights. They are Jaccard, Dice (also known as Sørensen-Dice), and Overlap.

3 Text Mining Functions

Text mining provides several types of functions you can use.

- Initialization
- Term-Document functions
- Categorization
- Statistical analysis

The text mining functions are offered via the SAP HANA Text Mining XS classic and advanced APIs and via SQL.

For information on how to access text mining functions via SQL, see the *SAP HANA SQL and System Views Reference*.

For information on how to access text mining functions via the SAP HANA XS classic model JavaScript API, see the *SAP HANA Text Mining XS JavaScript API Reference*.

For information on how to access text mining functions via the SAP HANA XS advanced model Node.js API, see the *SAP HANA Developer Guide for XS Advanced Model*.

Related Information

SAP HANA SQL and System Views Reference
SAP HANA Text Mining XS JavaScript API Reference
SAP HANA Developer Guide for XS Advanced Model

3.1 Initialization

When text mining is initialized on the reference document content in a given table and column, it creates and retains the term-document matrix and other configuration context needed for text mining functions.

Text mining initialization of reference data is normally done in advance when the full-text index is created for the given table and column. An explicit initialization call is also provided for advanced use. This can be done using either the `ALTER FULLTEXT INDEX` command in SQL or the `XS API initialize()` function. See the above document references for further information. Input documents are typically separate from the reference data and are automatically processed by text mining at run time as needed, so initialization does not need to be invoked for them.

3.2 Term-Document Functions

Text Mining provides the following functions to determine top-ranked related and relevant documents and terms.

Table 2:

Name of Function	Input	Output	Returned Content
`RELATED TERMS`	Term	Terms	top-ranked related terms based on co-occurrence
`RELEVANT TERMS`	Document	Terms	top-ranked relevant terms (keyphrases) that describe document
`SUGGESTED TERMS`	Term Substring	Terms	top-ranked terms matching initial substring
`RELATED DOCUMENTS`	Document	Documents	top-ranked documents related to document
`RELEVANT DOCUMENTS`	Term	Documents	top-ranked documents relevant to term

3.3 Categorization

Text mining supports categorization functionality.

Categorization or classification is based on comparing an input document with a reference set of already-classified documents. The reference set documents must contain a suitably complete and representative vocabulary and be sufficiently numerous for each categorization choice.

The simplest case is to have a single categorization column that can have a single value. Text mining can also support multiple independent categorization columns of this type. For example, each document could have a subject and a geographical region.

Text mining supports categorization using the KNN Classifier. No training (up-front preparation) is required – it uses the reference set documents directly. The KNN Classifier determines the K Nearest Neighbors (most-similar documents) from the reference set and then sums and normalizes their similarities per category value to determine the winning category value. If there are multiple categorization columns, no recomputation is necessary – the same K Nearest Neighbors can determine multiple category columns within the same function call.

For a simple example, let's assume your documents will either be about cars or about cell phones. You would collect a sufficient number of representative documents about each subject and manually label them with a category value that indicates the subject. These are then read into a SAP HANA database table, where the text is put into one column and the assigned subject category value is put into another column. Text analysis is performed on the text, which allows the base forms and parts of speech of words to be used as terms by text mining. Then text mining is initialized on this reference document table to generate the internal Term-Document matrix data that is needed for text mining functions.

At run time, you receive a new test document that you want to classify. The role of the text mining functions at run time is to analyze the input document and determine the K documents from the reference set that are most similar to it based on the terms used. Text mining automatically determines which terms are significant without human assistance when it examines the term usage within the document set. The categorization function determines the top K most-similar reference documents and their similarities, and does a weighted comparison by adding and normalizing the similarities for each category value. You may now see for example that the winning subject is cars, so we classify the new test document as also being about cars and return the ranked list for analysis.

3.4 Statistical Analysis

Text mining supports various kinds of statistical analysis in the related and relevant document and term functions.

The following statistical functions are available:

Table 3:

Statistical Function	Description
Correlation Matrix	Returns a term or document correlation matrix, as per the result type. The matrix contains the correlation values between each of the returned results.
Principal Component Analysis	Does principal component analysis (dimensionality reduction). Returns factor and rotation values for the specified number of principal components, for each of the returned results.
Clustering	Does hierarchical clustering with the method specified: Complete Linkage, Single Linkage, Average Distance Within, Average Distance Between, or Ward's Method. Returns the clustering level, left value, and right value for each of the returned results.

If any statistical functions cannot be calculated (due to insufficient data or due to linear dependencies in the correlation matrix), no results will be returned for them.

4 Configuring Text Mining

Text mining configuration properties are used during initialization and are retained for later use with text mining functions on the same reference data. The configuration context specified during initialization is used later as the defaults for unspecified parameters when text mining functions are invoked on the given reference data.

The configuration is user-defined and may be hosted in the SAP HANA repository (where the default configuration file is named `DEFAULT.textminingconfig`), or stored in the database via the SAP HANA deployment infrastructure or using stored procedures. You can create additional custom configuration files, which can be specified by name when text mining is initialized.

See the *SAP HANA Text Analysis Developer Guide* for more details on storing configurations in the repository or database.

Related Information

SAP HANA Text Analysis Developer Guide

4.1 Configuration File Syntax

To create or modify text analysis configurations, you need to understand the XML syntax and parameters described in this topic.

Text mining options are grouped into two sections `<properties>` and `<stopwords>`. The default configuration and function parameters are specified in the `<properties>` section. The `<stopwords>`section is used to specify a list of terms to be ignored by text mining. These are generally common words, which are considered to have no value for document comparison and which would only waste internal space and degrade accuracy.

Stop words in the `<generic>` section apply to all languages. Language-specific sections can be created using the ISO 639-1 two-letter language codes, for example `<en>` or `<de>`.

The following is an example text mining configuration file:

```
<?xml version="1.0" encoding="utf-8" ?>
<textmining-configuration xmlns="http://www.sap.com/textmining/config/1.0">
  <properties>
<property name="documentRestriction"></property>
<property name="termTypeRestriction"></property>
<property name="language"></property>
<property name="mimeType"></property>
<property name="tfWeighting">TF</property>
<property name="idfWeighting">IDF</property>
<property name="minDocFrequency">1</property>
```

```
<property name="maxDocFrequency">-100</property>
<property name="minTermFrequency">1</property>
<property name="maxTermFrequency">-100</property>
<property name="top">12</property>
<property name="checkDuplicates">0</property>
<property name="similarityFunction">COSINE</property>
<property name="kNN">15</property>
<property name="threshold">0.0</property>
  </properties>
  <stopwords>
    <generic>
      <stopword>a</stopword>
      <stopword>about</stopword>
      <stopword>above</stopword>
      <stopword>according</stopword>
      <stopword>across</stopword>
      <stopword>actually</stopword>
      <stopword>adj</stopword>
      <stopword>after</stopword>
      <stopword>afterwards</stopword>
      <stopword>again</stopword>
      <stopword>against</stopword>
      <stopword>against</stopword>
      <!-- ... -->
    </generic>
  </stopwords>
</textmining-configuration>
```

Related Information

Configuration Properties [page 10]

4.2 Configuration Properties

The table lists all configuration properties which are available for text mining.

Table 4:

Property Name	Description
documentRestriction	Restricts reference documents used. SQL `WHERE` clause, for example `KEY=100` or `YEAR>2013` (default empty to allow all).
termTypeRestriction	Restricts reference terms used. Comma-separated list of term types to allow, for example `noun,proper*,verb`. A final `*` wildcard is supported, for example `proper*` will allow any term type beginning with `proper` such as `proper name`. Text analysis entities and phrases can also be specified. The default value of empty at initialization time allows `noun`, `proper name`, and all text analysis entities and phrases.

Property Name	Description
language	Language: EN, DE and so on (default empty for unspecified). Used both for processing input text and for filtering document result list. If unspecified, the default language is used for processing input text and documents from all languages are returned. The default language is the first one specified in `CREATE FULLTEXT INDEX LANGUAGE DETECTION` or English if none.
mimeType	MIME type: unknown, text/plain, text/html, application/pdf, etc. (default empty for unspecified).
tfWeighting	TF weighting: DEFAULT, TF, LTF, SQRT, CONSTANT (default TF).
idfWeighting	IDF weighting: DEFAULT, IDF, LINEAR, SQRT, CONSTANT (default IDF).
minDocFrequency	Minimum required document frequency a term must have for use (default 1 for no filtering). If negative, it is understood as the percentage of the total number of reference documents rather than as an absolute number.
maxDocFrequency	Maximum allowed document frequency a term can have for use (default -100 for no filtering). If negative, it is understood as the percentage of the total number of reference documents rather than as an absolute number.
minTermFrequency	Minimum required term frequency (number of distinct terms) a document must have for use (default 1 for no filtering). If negative, it is understood as the percentage of the total number of distinct terms in all reference documents rather than as an absolute number.
maxTermFrequency	Maximum allowed term frequency (number of distinct terms) a document may have for use (default -100 for no filtering). If negative, it is understood as the percentage of the total number of distinct terms in all reference documents rather than as an absolute number.
top	Maximum number of results to return (default 12).
maxTokensPhrase	Filter for maximum token length of phrases (default 4). No effect in SPS12 new text mining implementation.
maxThreads	**Note** ADVANCED USE ONLY Internal maximum number of threads (default 4). No effect in SPS12 new text mining implementation.
checkDuplicates	Detect and discard duplicate reference documents, as well as terms that are subsets of longer phrases. 1 to enable, 0 to disable (default).

Property Name	Description
similarityFunction	Similarity function. COSINE (default), JACCARD, DICE, OVERLAP
kNN	KNN K value. Maximum number of nearest neighbors for KNN categorization for example 15 (default 0).
threshold	Minimum similarity threshold for returned results, in range 0.0 to 1.0 (default 0.0).
stop words... (multiple entries, long list)	Stop list of terms to ignore (default empty to allow all).

5 Using Text Mining

Text mining is supported via the SAP HANA XS classic and advanced APIs and SQL.

To create a SAP HANA XS classic model application, you set up a project in SAP HANA Studio. You use JavaScript to develop your application.

To run your application, you can use any internet browser and open the application URL on the SAP HANA server which is hosting your application.

For information on how to access text mining functions via the SAP HANA XS JavaScript API, see the *SAP HANA Text Mining XS JavaScript API Reference*.

To create and run a SAP HANA XS advanced model application, you can use SAP Web IDE, the XS advanced application development environment. For more information, see the *SAP HANA Developer Guide for XS Advanced Model*.

You can also use SQL functions. For information on how to access text mining functions via SQL, see *SQL Reference for SAP HANA Options - Advanced Data Processing* in the *SAP HANA SQL and System Views Reference*.

To load data into the database tables you use the standard SAP HANA tools.

Related Information

SAP HANA Text Mining XS JavaScript API Reference
SAP HANA SQL and System Views Reference
SAP HANA Developer Guide for XS Advanced Model

5.1 General Requirements

There are several requirements that must be met in order to use text mining.

Text mining requires document content to be of a data type supported by text analysis. Text analysis is a prerequisite of text mining.

Currently, text mining cannot handle pending table changes that have not been merged yet. You must explicitly merge any delta table pending changes into the table using the following command and then (re)initialize text mining to incorporate the changes:

```
MERGE DELTA OF table_name
```

Text mining initialization of reference data is normally done when the full-text index is created for the table. The parameter settings `FAST PREPROCESS OFF` and `SEARCH ONLY OFF` must be in effect. The defaults for these parameters are different for different data types, so it is a good idea to specify them explicitly.

For example, you could use the `CREATE FULLTEXT INDEX` SQL command:

```
CREATE FULLTEXT INDEX index_table_name ON table_name (column_name)
    ASYNC
    FAST PREPROCESS OFF
    SEARCH ONLY OFF
    TEXT ANALYSIS ON
    TEXT MINING ON
    TEXT MINING CONFIGURATION MYTMCONFIGURATION.textminingconfig
    LANGUAGE DETECTION ('EN')
    CONFIGURATION 'LINGANALYSIS_FULL'
```

The `ALTER FULLTEXT INDEX` command can be used to do explicit text mining initialization after the full-text index has been created.

There is also a `TEXT MINING CONFIGURATION OVERLAY` parameter which can be used to override certain parameters in the text mining configuration without changing the configuration itself.

For more information on full-text indexes and the related SQL commands, see the *SAP HANA Search Developer Guide*.

The `TEXT/BINTEXT` data types are different because the full-text index is created automatically. For these types, the parameters must be specified in the `CREATE COLUMN TABLE` command after the document content column. For example:

```
CREATE COLUMN TABLE table_name(
    KEY NVARCHAR(12),
    FILENAME NVARCHAR(200),
    FILECONTENT TEXT FAST PREPROCESS OFF SEARCH ONLY OFF,
    SUBJECT NVARCHAR(64),
    REGION NVARCHAR(64),
    PRIMARY KEY (KEY))
```

The reference documentation content and its intended input documents should all be in the same language. If the language is not the default (English), it should be specified both at full-text index creation and in the text mining function calls.

Related Information

SAP HANA Search Developer Guide

5.2 Using Different User Credentials

For security reasons we recommend to only grant users access to the text mining application and not directly to the database tables.

Your text mining application needs to have the proper credentials to access the reference data and other tables that it uses. By default, text mining uses the credentials of the user running the application. For security reasons we recommend to only grant users access to the application and not directly to the database tables.

To support this, text mining allows you to specify an alternate anonymous/technical user via SQLCC that should be used during a text mining session to determine access rights.

You deploy the text mining application along with a `.xssqlc` file that specifies a technical user with the desired access rights. The application needs to open a connection to the `.xssqlcc` file and specify it when creating the text mining session.

The following example shows how to specify an alternate technical user in the XS classic API:

Sample Code

```
var conn = $.db.getConnection("package_path::tm_technical_user");
var TM = new $.text.mining.Session({
    referenceTable: "SYSTEM.TMDOCUMENTS",
    referenceColumn: "FILECONTENT",
    connection: conn
});
var init = TM.initialize({});
...
```

Related Information

SAP HANA Text Mining XS JS API

5.3 Text Mining XS Classic API Example Application

The example shows an SAP HANA XS classic model JavaScript API application which performs an explicit initialization and exercises each of the text mining functions. The returned result table of each function is displayed.

The example assumes that the reference data table has already been loaded into the table and column specified in the initialize call.

Sample Code

```
var output = "<body>";
output += "<h2>Text Mining function test</h2>";
//---------------------------------------------------------------------------
function displayResults(results) {
    var propName = null;
    var propNames = [];
    var result = null;
    var row = 0;
    var col = 0;
    if (results.length === 0) {
        output += "<p>(empty result table)</p>";
        return;
    }
    output += "<table border='1'>";

    // Get and save property names from first result row and put in the table
header.
```

```
    output += "<tr>";
    for (propName in results[0]) {
        if (results[0].hasOwnProperty(propName)) {
            propNames[propNames.length] = propName;
            output += "<th>" + propName + "</th>";
        }
    }
    output += "</tr>";
    // For each result row, put in the table property values.
    for (row = 0; row < results.length; ++row) {
        result = results[row];
        output += "<tr>";
        for (col = 0; col < propNames.length; ++col) {
            output += "<td>" + result[propNames[col]].slice(0,50) + "</td>";
        }
        output += "</tr>";
    }
    output += "</table>";
}
//
------------------------------------------------------------------------------
try {
    output += "<p>create session</p>";
    var TM = new $.text.mining.Session({
        referenceTable: "SYSTEM.TMDOCUMENTS",
        referenceColumn: "FILECONTENT",
    });

    output += "<p>initialize</p>";
    var init = TM.initialize({    });

    output += "<p>categorizeKNN</p>";
    var categoryResults = TM.categorizeKNN ({
        top: 16,
        inputDocumentText: "animals",
        categorySets: ["SUBJECT", "REGION"],
        kNN: 15,
    });
    displayResults(categoryResults);

    output += "<p>getRelatedTerms</p>";
    var termResults = TM.getRelatedTerms ({
        top: 16,
        inputTermText: "animals",
    });
    displayResults(termResults);

    output += "<p>getRelevantTerms</p>";
    var termResults = TM.getRelevantTerms ({
        top: 16,
        inputDocumentText: "animals",
    });
    displayResults(termResults);

    output += "<p>getSuggestedTerms</p>";
    var termResults = TM.getSuggestedTerms ({
        top: 16,
        inputTermText: "a",
    });
    displayResults(termResults);

    output += "<p>getRelatedDocuments</p>";
    var documentResults = TM.getRelatedDocuments ({
        top: 16,
        inputDocumentText: "animals",
        includeColumns: ["KEY", "FILECONTENT"],
    });
    displayResults(documentResults);
```

```
    output += "<p>getRelevantDocuments</p>";
    var documentResults = TM.getRelevantDocuments ({
        top: 16,
        inputTermText: "animals",
        includeColumns: ["KEY", "FILECONTENT"],
    });
    displayResults(documentResults);
    output += "<p>Done -- no errors detected.</p>";
}
catch(err) {
    output += "<p>" + err + "</p>";
}
//------------------------------------------------------------------------------
output += "</body>";
$.response.contentType = "text/html";
$.response.setBody(output);
```

5.4 Text Mining XS Advanced API Example Application

This example shows an SAP HANA XS advanced model Node.js API application which performs an explicit initialization and exercises each of the text mining functions. The returned results of each function are displayed.

The example assumes that the reference data table has already been loaded into the table and column specified in the initialize call.

Sample Code

```
// To run this sample, you need these two modules in your Node.js development
// environment:
var textmining = require('sap-textmining');
// If you run this sample under the sap-textmining directory, you can replace
// the above line with following:
//var textmining = require('./index');
var hdb = require('hdb');
// Setup HANA database info
var db = {
  "host": "localhost",
  "port": 30015,
  "user": "SYSTEM",
  "password": "manager"
}
// Create hdb connection instance
var client = hdb.createClient({
  host: db.host,
  port: db.port,
  user: db.user,
  password: db.password
});
var config = {
  client : client,
  referenceTable : 'SYSTEM.TMDOCUMENTS',
  referenceColumn : 'FILECONTENT'
}
// Create textmining instance
var tmd = new textmining(config);
client.connect(function(err) {
  if(err) {
```

```
    client.end();
    throw err;
    return;
  }
  // initialize()
  var p_initialize = {
    referenceIndex : "\"SYSTEM.INDEX_TMDOCUMENTS\""
  }
  tmd.initialize(p_initialize, function(err) {
    if(err)
    {
      client.end();
      throw err;
      return;
    }
    // categorizeKNN()
    var pKNN = {
      top: 16,
      inputDocumentText: "animals",
      categorySets: ["SUBJECT", "REGION"],
      kNN: 15
    }
    tmd.categorizeKNN(pKNN, function(err, result) {
      if(err) {
        client.end();
        throw err;
        return;
      }
      console.log("Result of categorizeKNN:");
      console.log(result);
      console.log();
      client.end();
    });
    // getRelatedTerms()
    var pGetRelatedTerm = {
      top: 16,
      inputTermText: "animals"
    }
    tmd.getRelatedTerms(pGetRelatedTerm , function(err, result) {
      if(err) {
        client.end();
        throw err;
        return;
      }
      console.log("Result of getRelatedTerms:");
      console.log(result);
      console.log();
      client.end();
    });
    // getRelevantTerms()
    var pGetRelevantTerm = {
      top: 16,
      inputDocumentText: "animals"
    }
    tmd.getRelevantTerms(pGetRelevantTerm , function(err, result) {
      if(err) {
        client.end();
        throw err;
        return;
      }
      console.log("Result of getRelevantTerms:");
      console.log(result);
      console.log();
      client.end();
    });
    // getSuggestedTerms()
    var pGetSuggestedTerm = {
      top: 16,
```

```
      inputTermText: "a"
    }
    tmd.getSuggestedTerms(pGetSuggestedTerm , function(err, result) {
      if(err) {
        client.end();
        throw err;
        return;
      }
      console.log("Result of getSuggestedTerms:");
      console.log(result);
      console.log();
      client.end();
    });
    // getRelatedDocuments()
    var pGetRelatedDoc = {
      top: 16,
      inputDocumentText: "animals",
      includeColumns: ["REF_DOC_KEYS", "KEY", "FILECONTENT"]
    }
    tmd.getRelatedDocuments(pGetRelatedDoc , function(err, result) {
      if(err) {
        client.end();
        throw err;
        return;
      }
      console.log("Result of getRelatedDocuments:");
      console.log(result);
      console.log();
      client.end();
    });
    // getRelevantDocuments()
    var pGetRelevantDoc = {
      top: 16,
      inputTermText: "animals",
      includeColumns: ["REF_DOC_KEYS", "KEY", "FILECONTENT"]
    }
    tmd.getRelevantDocuments(pGetRelevantDoc , function(err, result) {
      if(err) {
        client.end();
        throw err;
        return;
      }
      console.log("Result of getRelevantDocuments:");
      console.log(result);
      console.log();
      client.end();
    });
  });
});
```

5.5 Text Mining SQL API Example Application

This example shows an SAP HANA SQL API application which exercises each of the text mining functions. The returned results of each function are displayed.

The example assumes that the reference data table has already been loaded into the table and that the full-text index has been created to initialize text mining.

Sample Code

```
select *
  FROM TM_CATEGORIZE_KNN (
    DOCUMENT 'animals'
    SEARCH NEAREST NEIGHBORS 15 "FILECONTENT" FROM "SYSTEM"."TMDOCUMENTS"
    RETURN
      TOP 16
      SUBJECT
      FROM "SYSTEM"."TMDOCUMENTS"
    RETURN
      TOP 16
      REGION
      FROM "SYSTEM"."TMDOCUMENTS"
  ) AS T;
select T.TERM, T.NORMALIZED_TERM, T.TERM_TYPE, T.TERM_FREQUENCY,
T.DOCUMENT_FREQUENCY, T.SCORE
  FROM TM_GET_RELATED_TERMS (
    TERM 'animals'
    SEARCH "FILECONTENT" FROM "SYSTEM"."TMDOCUMENTS"
    RETURN
      TOP 16
  ) AS T;
select T.TERM, T.NORMALIZED_TERM, T.TERM_TYPE, T.TERM_FREQUENCY,
T.DOCUMENT_FREQUENCY, T.SCORE
  FROM TM_GET_RELEVANT_TERMS (
    DOCUMENT 'animals'
    SEARCH "FILECONTENT" FROM "SYSTEM"."TMDOCUMENTS"
    RETURN
      TOP 16
  ) AS T;
select T.TERM, T.NORMALIZED_TERM, T.TERM_TYPE, T.TERM_FREQUENCY,
T.DOCUMENT_FREQUENCY, T.SCORE
  FROM TM_GET_SUGGESTED_TERMS (
    TERM 'a'
    SEARCH "FILECONTENT" FROM "SYSTEM"."TMDOCUMENTS"
    RETURN
      TOP 16
  ) AS T;
select T.KEY, T.FILECONTENT, T.TOTAL_TERM_COUNT, T.SCORE
  FROM TM_GET_RELATED_DOCUMENTS (
    DOCUMENT 'animals'
    SEARCH "FILECONTENT" FROM "SYSTEM"."TMDOCUMENTS"
    RETURN
      TOP 16
      KEY, FILECONTENT
  ) AS T;
select T.KEY, T.FILECONTENT, T.TOTAL_TERM_COUNT, T.SCORE
  FROM TM_GET_RELEVANT_DOCUMENTS (
    TERM 'animals'
    SEARCH "FILECONTENT" FROM "SYSTEM"."TMDOCUMENTS"
    RETURN
      TOP 16
      KEY, FILECONTENT
  ) AS T;
```

5.6 The Public Text Mining Data Table Interface

For advanced users who want to implement text mining algorithms of their own, the text mining Term-Document matrix is exposed as a system table.

To expose this data, text mining follows the same pattern as the $TA table for text analysis, as described in the *SAP HANA Text Analysis Developer Guide*. For each full-text index configured to use text mining, the system creates an additional table with the name `$TM_MATRIX_<index_name>` in the same schema that contains the source table. This table portrays the term-document matrix data, i.e. the terms and their counts in each reference document.

The `$TM` matrix table layout has been designed to avoid empty entries, since a literal term-document matrix would be sparse and inefficient. It has also been designed to allow efficient SQL traversal by document or by term to calculate all required quantities for text mining algorithms.

Table 5: Structure of the $TM Matrix Table

Column ID	Key	Description	Data Type
<n key columns from source table>	Yes	A text mining document. To support a foreign key definition linking from the `$TM` matrix table to its related `$TA` table and source table, the `$TM` matrix table uses exactly the same key columns as its source table (in data type and ID). The `$TM` matrix table includes all keys from the source table. This serves as the key to identify a reference document.	Same as in source table
`TM_TERM`	Yes	A text mining term. This is a single-word or multi-word phrase of the type specified in `TM_TERM_TYPE` and is normalized as in the `$TA` table. The `TM_TERM/TM_TERM_TYPE` combination serves as the key to identify a term.	NVARCHAR(500)
`TM_TERM_TYPE`	Yes	The type of the text mining term (open-ended). This is typically the part of speech for single-word terms e.g. `noun`. It can also be an entity type or phrase type for multi-word terms. The `TM_TERM/TM_TERM_TYPE` combination serves as the key to identify a term.	NVARCHAR(100)
`TM_TERM_FREQUENCY`	-	The term frequency, or count of the number of instances of this `TM_TERM/TM_TERM_TYPE` in this document.	BIGINT

> **Note**
>
> If the source table has a key field name identical to one of the standard fields in the `$TA` table, you will receive an error message after the `CREATE FULLTEXT INDEX` statement, prompting you to rename the field in the source table. Once you have renamed the corresponding field, you can execute the `CREATE FULLTEXT INDEX` statement again.

The fields in the $TM matrix table have the following properties:

- There is an entry with the same document key for each term in that reference document.
- If the same term appears again within the same document, it is not repeated for that document – it increments TM_TERM_FREQUENCY.
- If a term appears again in a different document, it occurs again for that new document.
- The document key / term key combination is unique, i.e. there is only one row for each document key / term key combination.

SQL operations can collect information by document or by term using GROUP BY document key or GROUP BY term key, as well as per document or per term. Here are some examples of fundamental text mining formulas calculated from the $TM matrix table:

Table 6:

Formula	SQL
D = total #docs	`SELECT COUNT(DISTINCT <source_table_keys>) AS D FROM "$TM_MATRIX_<index_name>"`
T = total #distinct terms	`SELECT COUNT(DISTINCT TERM,TERM_TYPE) AS T FROM "$TM_MATRIX_<index_name>"`
DF(term) = #docs in which the term occurs	`SELECT TERM,TERM_TYPE, COUNT(*) AS DF FROM "$TM_MATRIX_<index_name>" GROUP BY TERM,TERM_TYPE`
TF(doc,term) = #occurrences of term in doc	`SELECT <source_table_keys>, TERM,TERM_TYPE, TERM_FREQ AS TF FROM "$TM_MATRIX_<index_name>"`
IDF(term) = ln(D/DF(term)) [embeds D and DF from above]	`SELECT TERM,TERM_TYPE, LN((SELECT COUNT(DISTINCT <source_table_keys>) FROM "$TM_MATRIX_<index_name>") / COUNT(*)) AS IDF FROM "$TM_MATRIX_<index_name>" GROUP BY TERM,TERM_TYPE`

It can be more efficient to cache intermediate data in tables, rather than embedding formulas as in the IDF example above.

> **Note**
>
> You might notice tables being created with names like $TM_DOCUMENTS_<index_name> and $TM_TERMS_<index_name>. These tables are for SAP use only and should not be used by applications. Their contents are not guaranteed.

If modifications are made to the source table containing the reference documents, the index must be recreated manually. Otherwise the $TM matrix table and other internally cached data will be out-of-date. There is currently no recommended solution for using triggers and foreign keys to update them automatically.

Note

SAP HANA SPS 12 introduces a new internal implementation of text mining. Its new features include the public text mining data table interface described above, as well as expanded term type possibilities such as text analysis entities and phrases.

The original SPS 11 text mining implementation is still available in SPS 12 and can be selected if necessary by using the following configuration parameter setting:

```
ALTER SYSTEM ALTER CONFIGURATION ( 'indexserver.ini', 'System' )
SET ( 'text', ' legacy_text_mining' ) = 'True' WITH RECONFIGURE;
```

Important Disclaimers and Legal Information

Coding Samples

Any software coding and/or code lines / strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended to better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, unless damages were caused by SAP intentionally or by SAP's gross negligence.

Accessibility

The information contained in the SAP documentation represents SAP's current view of accessibility criteria as of the date of publication; it is in no way intended to be a binding guideline on how to ensure accessibility of software products. SAP in particular disclaims any liability in relation to this document. This disclaimer, however, does not apply in cases of willful misconduct or gross negligence of SAP. Furthermore, this document does not result in any direct or indirect contractual obligations of SAP.

Gender-Neutral Language

As far as possible, SAP documentation is gender neutral. Depending on the context, the reader is addressed directly with "you", or a gender-neutral noun (such as "sales person" or "working days") is used. If when referring to members of both sexes, however, the third-person singular cannot be avoided or a gender-neutral noun does not exist, SAP reserves the right to use the masculine form of the noun and pronoun. This is to ensure that the documentation remains comprehensible.

Internet Hyperlinks

The SAP documentation may contain hyperlinks to the Internet. These hyperlinks are intended to serve as a hint about where to find related information. SAP does not warrant the availability and correctness of this related information or the ability of this information to serve a particular purpose. SAP shall not be liable for any damages caused by the use of related information unless damages have been caused by SAP's gross negligence or willful misconduct. All links are categorized for transparency (see: http://help.sap.com/disclaimer).

go.sap.com/registration/contact.html