Extending the PMML Text Model for Text Categorization

Benjamin De Boe
InterSystems Corporation
One Memorial Drive
Cambridge, MA 02142, USA
Benjamin.De.Boe@intersystems.com

Misha Bouzinier
InterSystems Corporation
One Memorial Drive
Cambridge, MA 02142, USA
Misha.Bouzinier@intersystems.com

Dirk Van Hyfte, MD, PhD
InterSystems Corporation
One Memorial Drive
Cambridge, MA 02142, USA
Dirk.VanHyfte@intersystems.com

ABSTRACT

As the de facto standard for expressing data mining models, the Predictive Model Markup Language (PMML) offers a way to decouple the activities of building a data mining model from actually running it on new data. This enables more flexibility for data scientists and application developers alike, as they now have a common language to express models, independent of the technologies and architectures of their respective environments.

In this paper, we will look into using PMML for Text Categorization scenarios. Broadly speaking, Text Categorization is about deciding which of a predefined set of categories a piece of free text is most likely to belong to, typically by looking for the occurrences of distinctive terms in the input text and feeding them into a classification model. Both the choice of terms to consider and the coefficients of the model are derived as part of a data mining effort, involving the analysis of large volumes of text. Given that text mining technology is still not mainstream, a standard such as PMML for separating the building and execution of the model gets even more important.

The PMML specification includes a model type for predicting an outcome based on text input, in the form of the "document" the input text is most similar to, selected from a set of documents defined in the model. In this paper, we will discuss examples of simple Text Categorization scenarios that cannot be expressed through the current Text Model specification in PMML version 4.1. We will propose extensions to expand its scope to true Text Categorization, from the individual document similarity approach offered today. We will describe a generalization of the idea of "documents" to "categories" and refine the available options for calculating the similarity of text input to the categories defined in a model. In addition, we will describe an alternative approach for allowing other model types to transparently work with text input, emulating derived fields for weighted term frequencies.

Categories and Subject Descriptors

H.3.1 **[Information Storage and Retrieval]**: Content Analysis and Indexing - *Indexing methods*;

H.3.3 **[Information Storage and Retrieval]**: Information Search and Retrieval - *Information filtering*

General Terms

Algorithms, Standardization, Languages, Theory

PMML'13, August 11 2013, Chicago, Illinois, USA

ACM 978-1-4503-2336-9/13/08…$15.00

Keywords

Predictive Analytics, Data Mining, PMML, Predictive Model Markup Language, Text Categorization, Text Classification, iKnow

1. INTRODUCTION

Even if we completely disregard the twitters and facebooks of today, it's clear the amount of digitally available textual information available to organizations and individuals today is overwhelming and growing. Electronic medical records in healthcare, analyst and regulatory reports in finance and claims in an insurance context are just three examples of textual information that's becoming available at an ever increasing rate and meant to support better decision-making. Moreover, in all three of these examples, the textual information is not only useful for making the decision, but in many cases simply essential or even legally binding to be taken into account. In the first example, a clinician who's ignoring parts of his patients medical history notes when prescribing a medication or procedure could find himself part of the insurance example soon after.

Unfortunately, the availability of so much information doesn't make the lives of clinicians, traders or claims adjusters any easier by itself. The sheer volume often makes it infeasible to read through all of it, or the velocity at which it arrives might make it hard to act upon it in an acceptable timeframe. If only computers could read and understand natural language, they could suggest which decision to take and humans should only verify and confirm those cases where the computer wasn't sure or where legal or risk factors wouldn't allow a computer to press the button.

The scenario just described might not be that far away into the future. First, dedicated software exists today for *mining* information from free text and modeling the patterns and characteristics discovered in that text mining effort. Second, dedicated software also exists for managing electronic medical records, for trading on stock markets and for managing insurance claims. And third, an industry standard exists to express the results of the data mining effort into a format that could be consumed from the production application managing the live data: PMML [1] [2] [3].

In section 2 of this paper, we'll describe the capabilities in PMML version 4.1 for using text input in predictive models. We'll illustrate how the current version of the standard only covers a small subset of a broad array of use cases named Text Categorization. In the next section, we'll look in more detail at the field of Text Categorization and discuss a number of well-known categorization techniques. For each of these, we'll use examples of an extended PMML Text Model to show how the standard can be enriched to cover this broader area. Then, section 4 will present an alternative approach to using text input in PMML, which does

not imply a separate model type. In this section, we'll also explain how dedicated text analysis technology can be used in a formalized preprocessing step, expanding the scope of any PMML model to text input. In this context, we'll present iKnow, a dedicated text analysis technology distributed by InterSystems Corporation, as a sample tool to derive from text input those metrics that can make any predictive model not just text-enabled, but truly text-aware. Finally, in section 5 we'll summarize this paper into a conclusion.

2. CURRENT TEXT MODEL

As a standard, PMML covers a variety of well-known and widely adopted data mining model types, including Naïve Bayes, Support Vector Machines and Neural Networks. For each of these, the PMML specification describes an XML element representing the model and all of its parameters as calculated by the model producer, which means a model consumer gets all there is to know to be able to execute the model on new data.

Amongst these model types, the Text Model represents a predictive model which predicts an outcome based on text input. In this section, we'll first present the specification in version 4.1 of the PMML standard and then review its limitations.

2.1 Current specification

While most model types covered by PMML relate back to a specific algorithm, technique or functional approach in data mining, the Text Model[1] is a little different in that it rather represents a specific type of input (text) and a single use case than anything else. Essentially, this use case comes down to, for a given input text, identifying which of a predefined set of documents it is most similar to. Obviously, based on this "most similar document" outcome, a decision can be made either explicitly within the model definition using an `OutputField` element representing a *decision* feature, or through custom code interpreting the raw outcome outside of the scope of PMML.

The formal PMML specification describes the concept of text similarity through these six components:

- model attributes;
- dictionary of terms;
- corpus of text documents
- document-term matrix;
- text model normalization;
- text model similarity.

The first component, "model attributes", does not encompass any specific new XML elements, neither are the ones referred to used any differently from other model types. We'll therefore ignore this component for now. We'll look at each of the other elements in more detail.

2.1.1 Dictionary of terms

The dictionary of terms is represented by the `TextDictionary` element, which consists mainly of an array of strings holding the terms of interest for this model. This means, the Text Model will only consider the occurrence of these terms when calculating the similarity between the input text and the existing documents.

Optionally, a taxonomy can be defined on these terms through the formal `Taxonomy` element[2], as an external source of information on how to group terms hierarchically. However, the current specification does not detail how this taxonomy could or should be treated.

2.1.2 Corpus of text documents

The next element in the Text Model specification describes the corpus of documents this model will compare input text against. The `TextCorpus` element explicitly enumerates each document in a `TextDocument` element, with optional attributes for a description, file name and document length in bytes.

2.1.3 Document-term matrix

The `DocumentTermMatrix` then ties the dictionary terms to the documents by specifying a frequency matrix, with the element on the i^{th} row and j^{th} column representing the frequency of the term at index j in the document at index i.

2.1.4 Text model normalization

As to normalize the term frequencies in the document-term matrix, a `TextModelNormalization` element defines how the model consumer should transform this frequency matrix into term weights that can be used for the similarity calculation. This is the first element to deal with any text-specific metrics or functions.

2.1.5 Text model similarity

The `TextModelSimiliarity` element finally closes the loop and details how the input fields should be compared to the normalized per-document term weights. The two similarity measures supported by the model (cosine similarity and Euclidean distance) assume a Vector Space Model, which will be described in more detail in Section 3 of this paper.

2.2 Current limitations

While it's clear the Text Model has its merits, not in the least by giving text the attention it deserves in the field of predictive analytics, we believe the current specification for the model only serves a rather basic use case and has a number of shortcomings for implementing Text Categorization scenarios. We'll discuss and illustrate them one by one.

2.2.1 No real category support

In many cases, you might not want to compare input text to individual documents. Even if they're reference documents, they almost certainly represent a whole category of documents and most literature on Text Categorization discusses this more coarse-grained approach[3].

For example, in a customer service scenario, the goal might be to route an incoming service request email to the right support engineer by looking at its textual contents. Typically, you'd like to find the *category* of requests this new email belongs to and forward to the team treating that category. However, the current Text Model specification would require you to find the single previous email it was most similar to and then route to the team that successfully handled that issue. This will either require your model to contain a probably unrealistic number of documents to compare against, or require you to select a number of "representative" emails to include in the model definition, at the

[1] See also: http://www.dmg.org/v4-1/Text.html

[2] See also: http://www.dmg.org/v4-1/Taxonomy.html

[3] There are some Text Categorization techniques that will look at similarity to individual documents, such as example-based classifiers and (to some extent) the *Rocchio method* [4], but these are still techniques to finding a category rather than a document and require more parameters than what's currently possible through the Text Model.

risk of selecting too many to perform quickly or too few to remain accurate.

One solution to this could be to simply use the `TextDocument` elements as representations for whole categories. The problem then becomes what to list in the `DocumentTermMatrix`. The total frequency of a term in a category is one option, but it might attribute improper weights to terms that occur very frequently in only one or two documents out of many in a category, or terms that occur just once, but in every category. Other options include the average term frequency (yielding a centroid approach), the number of documents within the category in which they appear and other aggregations, all of which might or might not be compatible with the weighting options presented by `TextModelNormalization`.

In Section 3, we'll describe an extension to the PMML Text Model specification that should accommodate these categorization scenarios more properly.

2.2.2 Limited classification options

As we'll elaborate in Section 3, there are many different techniques to implement Text Categorization scenarios [4]. Most of them are based on a representation of text that is commonly known as the Vector Space Model, in which every document is represented as vector of term weights. The current PMML Text Model specification relies on this model and offers two simple functions to measure the similarity between input vectors and the document vectors (as derived from the document term matrix). While these are popular and relatively straightforward for both model building and execution, they only represent a fraction of that broad set of available techniques.

In Section 3, we'll discuss the Vector Space Model in more detail and propose extensions to the current Text Model specification to cover some of the alternative classification techniques. In Section 4, we'll investigate an alternative approach, leveraging other existing PMML model types for classification.

2.2.3 No real text-awareness

The current specification of the Text Model does not seem to rely on any text-awareness of the model implementation technology. This is because the model requires term *frequencies* as inputs and, onwards, only works with numeric calculations that are independent of the text background. And while the normalization techniques and similarity metrics are indeed typical ones for dealing with text in a Vector Space Model, they are also used elsewhere and nothing would prevent users from deploying a PMML Text Model for input that has nothing to do at all with text. For example, a primitive market basket analysis model could represent customer buying behavior the same way as term frequencies and use a Text Model to predict which "reference customer" (document) a customer is most similar to by simply looking at the products (terms) he bought. Market basket analysis specialists might disagree with the accuracy of such a technique, but at least it illustrates the current Text Model specification doesn't have a lot to do with text by itself.

This approach, of course, can be considered an advantage, as it makes it easier to implement a model consumer for the Text Model. But at the same time it just defers the job of interpreting the text and providing the term frequencies to a preprocessing step for which PMML provides no guidance at all. We'll look at solutions to increase the text-awareness of PMML as a standard in general in Section 4.

2.2.4 Other remarks

The following remarks on the current Text Model specification are not strictly related to Text Categorization scenarios. They are included nonetheless as to make this section a more comprehensive review of the Text Model specification in PMML version 4.1.

- **Input mapping**: The current specification does not explicitly link `MiningField` or `DerivedField` elements to the `TextDictionary` elements or `DocumentTermMatrix` columns, but implicitly (without even mentioning in the specification) assumes they correspond based on their index. We think this is a gap in the specification and at least a formal description of how the order of plain and derived fields' indices should be interpreted, as well as how to treat non-active mining fields, is required. Preferably, a new element would allow the model producer to define this mapping explicitly as to avoid any misinterpretation. A proposal to accommodate this was submitted by the authors separately.
- **TextModelSimiliarity spelling**: There appears to be a misspelling in this element name, which is upheld throughout the specification and examples. As the word "similiarity" to our knowledge is not a specific concept in the literature on Text Analysis, we believe "correcting" this element's name in the specification should be considered.
- **Taxonomy support**: The specification allows adding a `Taxonomy` element to the term dictionary, but does not mention how this should be used. A formal description of how terms can be rolled up and how these "topics" can then be mapped to the columns in the `DocumentTermMatrix` would be a helpful extension.

3. TEXT CATEGORIZATION MODELS

In this section, we'll introduce the broader research area of Text Categorization and propose a number of extensions to the PMML standard to make the Text Model more compatible with Text Categorization scenarios. We'll present examples of three techniques that can be easily covered by the extended specification and give an outlook on further extensions in the future.

3.1 Introduction to Text Categorization

Sebastiani [4] provides an excellent overview of the different approaches to Text Categorization. He defines Text Categorization (TC, also known as Text Classification) as "the activity of labeling natural language texts with thematic categories from a predefined set". While this simple definition seems to refer mostly to the *execution* of a classification model, it's clear there has to be some logic or algorithm to build or train the model that provides this labeling service accurately.

Many properties of "normal" data mining classification also apply to TC. There is a (mostly algorithmic) difference between binary and multi-class classification. In binary classification, the classifier should only decide which of two categories a new document belongs to, whereas the broader multi-class classification can have any fixed number of categories. A classifier can also be used for "hard" classification, in which a document gets assigned a single category, or "ranking" classification, in which the result is a set of categories ranked by the likelihood the document belongs to them. Unless specified otherwise, we'll talk about hard, multi-class classification onwards, as binary classification then is just a special case and not all classification methods might allow for the ranked scenario (i.e. decision trees typically don't rank results).

3.1.1 Indexing and the Vector Space Model

The first step in building a TC model is choosing how to represent the unstructured text in a more structured format. By far the most common approach is the Vector Space Model, as presented by Salton et al [5]. In the Vector Space Model, each document is defined by a vector containing the frequency (or a weighted version thereof) of a specific term at each index. If documents or whole categories can be represented as vectors, suddenly a broad range of vector operations become available to express transformations or interpretations of what until then was purely unstructured text. For example, two vectors (documents) could be considered similar if the points in the multidimensional vector space they represent are close, based on the Euclidean distance. Alternatively, you can consider vectors to be similar if they point in the same direction, by using cosine similarity, as calculated through a dot product.

An essential step to transform unstructured text into a vector of (weighted) frequencies is *indexing* the text. The most basic of indexing algorithms would just count the frequencies of all individual words, but most text mining technology includes more intelligent logic for tasks such as discarding stop words, standardizing terms through *stemming* and identifying word groups that belong together. Technically, the only truly text-aware part of the whole TC process is this indexing step, as the resulting vector is again just a bunch of numbers that could have come from a completely different source. We'll discuss indexing in more detail in Section 4.

To be complete, the Vector Space Model can express more than just (weighted) term frequencies, as is documented in [6], and indexing too can yield much more than raw frequencies, which we'll elaborate on in Section 4. However, most TC techniques use plain or weighted term frequencies as their straightforward input.

3.1.2 Dimensionality reduction

One significant problem with the Vector Space Model, which extends to a number of similar approaches, is the high dimensionality of the vector space. Natural language does not restrict itself to a handful of terms and therefore the number of dimensions can easily stretch into the tens of thousands, even for a small dataset. Therefore, it's important to reduce this dimensionality through term selection and, optionally, term extraction, which conceptually corresponds to feature selection and feature extraction in data mining in general.

Term selection is about reducing the dimensionality by selecting a representative set of terms and disregarding all others. This selection of terms to be considered is one of the most crucial tasks when building a TC model and is usually dependent on the classification method chosen. Typically, a first set is selected based on some metric expressing the overall importance of the term in the corpus, such as TFIDF or more advanced alternatives [7] [8]. Then, after an initial model is built, the set is refined by adding and removing terms based on their impact on the overall model accuracy until a certain threshold is met and the accuracy no longer changes significantly. When it comes to executing a model, this list of terms to consider is part of the model and therefore only the result and not the selection process is what's relevant for PMML as a specification. The `TextDictionary` element covers this for the Text Model.

If the vocabulary used in the documents to categorize is rich, term selection alone might leave you with too high a dimensionality, or yield a low accuracy if too many terms were discarded. Term extraction solves this by creating new "virtual" terms that represent a linear combination of existing terms. Commonly, the whole vector space is transformed into a new vector space with fewer dimensions, using techniques such as Principal Component Analysis [9], truncated SVD [10] or other classic feature extraction algorithms.

3.1.3 Other ways of representing text for TC

A common alternative to the Vector Space Model is the probabilistic model described by Lewis in [11]. This approach, which is based on Bayes' theorem, is often used in Information Retrieval problems but also serves TC scenarios well. In practice, it doesn't differ much from the universal Naïve Bayes model as described elsewhere in the PMML specification, although in a TC scenario the dimensionality will typically be a lot higher.

3.2 Proposed PMML extensions

In this subsection, we'll propose a number of extensions to the PMML specification for the Text Model in the context of Text Categorization.

3.2.1 From documents to categories

To explicitly enable the Text Model for TC, we'll introduce a `TextCategory` element that's similar to `TextDocument`, but now represents a number of documents, which can be specified through a `numberOfDocuments` attribute. A `TextCorpus` element can now either contain a set of categories or a set of documents, but not both.

```
<xs:element name="TextCorpus">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Extension" minOccurs="0"
          maxOccurs="unbounded"/>
      <xs:choice>
        <xs:element ref="TextDocument"
          minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="TextCategory"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="TextCategory">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Extension" minOccurs="0"
          maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string"
          use="required"/>
    <xs:attribute name="name" type="xs:string"
          use="optional"/>
    <xs:attribute name="numberOfDocuments"
          type="INT-NUMBER" use="optional"/>
  </xs:complexType>
</xs:element>
```

Figure 1: Extended specification for `TextCorpus`

Figure 1 presents the extended specification introducing category support, highlighting new elements and attributes in blue.

In the `TextModel` element itself, we propose to deprecate the `numberOfDocuments` attribute in favor of a new one named `numberOfCategories`, which would represent both the number of documents and categories, as a set of `TextDocument` elements can be interpreted as a series of single-document categories. If the corpus is defined as a set of documents, the model producer can choose whether to use the deprecated

numberOfDocuments or the new numberOfCategories (but not both). If the corpus contains a list of categories, numberOfCategories has to be used.

3.2.2 Normalizing the document-term matrix

As highlighted in the previous section, moving from documents to categories has an impact on the meaning of the DocumentTermMatrix contents. Where it used to contain simple term frequencies, this would no longer be sufficient information to apply the normalizations expressed in the TextModelNormalization element. For example, if a model defines the inverse document frequency (IDF) as the global term weight function and term frequency as the local term weight, we'd need both the number of documents in which each term occurs, as the frequency it has in each document (or category), which cannot be covered in a single matrix.

At the same time, normalizing the DocumentTermMatrix is a transformation that only depends on the attribute values of the TextModelNormalization element, which in turn is only used to produce this transformation. This means that, from the model consumer perspective, it would be equivalent if the DocumentTermMatrix already contained the normalized matrix and the TextModelNormalization element was absent. For the model producer, it shouldn't be any different either, as he's likely to have calculated this normalized matrix at some point anyway, while researching the weighting scheme best fitting the training data.

Thus, we propose to abandon the TextModelNormalization element and require the DocumentTermMatrix to contain the normalized term weights rather than raw frequencies. Turning the document-term matrix into a category weight matrix this way would also make it a more transparent vehicle for use with other similarity metrics than the vector-centric cosine similarity and Euclidean distance.

3.2.3 Input normalization and similarity metrics

In Figure 2, we propose an extension to the TextModelSimilarity[4] element for defining how the input vector (input frequencies) should be normalized and compared to the contents of the normalized document-term matrix.

```
<xs:element name="TextModelSimiliarity">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Extension" minOccurs="0"
            maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="localTermWeights"
            default="termFrequency">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="termFrequency"/>
          <xs:enumeration value="binary"/>
          <xs:enumeration value="logarithmic"/>
          <xs:enumeration value="
            augmentedNormalizedTermFrequency"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="documentNormalization"
            default="none">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="none"/>
          <xs:enumeration value="cosine"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="similarityType">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="euclidean"/>
          <xs:enumeration value="cosine"/>
          <xs:enumeration value="linear"/>
          <xs:enumeration value="naiveBayes"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>
```

Figure 2: Extended specification for `TextModelSimilarity`

The normalization options for the input vector are similar to those in the discontinued TextModelNormalization element, except for the absence of a global term weighting option, which is typically already applied at the side of the category term weight. This simple addition provides an easy way to transform the term frequencies in the input vector into a normalized local term weight ready for further calculations by the model.

As for the similarity metric options, we propose adding two new options to the existing cosine similarity and Euclidean distance:

- **linear**: In this approach, the document-term matrix is considered to be a matrix of coefficients for a linear regression formula. Each document term vector is simply multiplied with the input vector and the document with the highest resulting score wins.
- **naiveBayes**: This similarity type expects the DocumentTermMatrix to contain the number of documents containing term i (column) in category j (row), which can then be used to calculate Naïve Bayes probabilities. Scoring then proceeds in the same way as for a normal Naïve Bayes model[5]. The base probabilities of each document or category are calculated based on the numberOfDocuments attribute in each TextDocument element, which is mandatory in this case. The localTermWeights and documentNormalization attributes are expected to be "binary" and "none" respectively.

For completeness: in the case of the cosine similarity and Euclidean distance, the numbers in the document-term matrix are still expected to represent vectors that can be compared to the input vector.

3.2.4 Further extensions

We believe the introduction of TextCategory elements, the removal of the TextModelNormalization element and the addition of input normalization options already offer significant flexibility towards supporting Text Categorization in PMML. But this extension is certainly not an endpoint and more similarity types as well as local term weighting schemes can easily be added to the proposed specification.

[4] Note the corrected spelling for TextModelSimilarity element (was "TextModelSimiliarity")

[5] See also: http://www.dmg.org/v4-1/NaiveBayes.html

Independent of adding more algorithmic options, another interesting addition would be to extend the specification to accept the text itself as an input, rather than the term frequencies. Section 4 discusses this option in more detail.

3.2.5 Backwards compatibility

The above changes were meant to preserve backwards compatibility where possible. For some elements, a more radical change might be preferable for clarity, at the risk of invalidating older model definitions (which model consumers should then treat based on the PMML version number). Given the limited adoption of the Text Model thus far [3], the impact of such changes should be relatively small.

- The `numberOfDocuments` attribute in the `TextModel` element could be dropped altogether, with `numberOfCategories` fully taking over its role. This should avoid any confusion between the two attributes.
- As an even more radical simplification, the `TextDocument` element could be abandoned completely, only accepting `TextCategory` elements onwards. Models wishing to predict individual document similarity can then represent these as single-document categories.
- The `DocumentTermMatrix` can be renamed to `CategoryWeightMatrix`, which better represents its contents.

3.3 Example

```
<TextModel modelName="example" numberOfTerms="5"
           numberOfCategories="2">
 <MiningSchema>
  <MiningField name="headcheFreq" />
  <MiningField name="feverFreq" />
  <MiningField name="nauseaFreq" />
  <MiningField name="wellFreq" />
  <MiningField name="bedFreq" />
  <MiningField name="OK" usageType="predicted" />
 </MiningSchema>
 <TextDictionary>
  <Array type="string">fever headache nausea well
           bed</Array>
 </TextDictionary>
 <TextCorpus>
  <TextCategory id="ill" numberOfDocuments="10"/>
  <TextCategory id="fine" numberOfDocuments="20"/>
 </TextCorpus>
 <DocumentTermMatrix>
  <Matrix>
   <Array type="real">1 .9 1 0 .5</Array>
   <Array type="real">0 0 0 1 .3</Array>
  </Matrix>
 </DocumentTermMatrix>
 <TextModelSimilarity localTermWeights="binary"
           documentNormalization="cosine"
           similarityType="linear"/>
</TextModel>
```

Figure 3: Example Text Model with extended specification

4. UNTANGLING TEXT MODELS

In the previous section, we presented a number of extensions to expand the scope of the Text Model towards true Text Categorization scenarios. In this section, we'll look more closely at what can be done to leverage more of the PMML specification to support scenarios involving text.

4.1 Is text all that different?

It's clear that text by itself cannot be used as an input for a typical data mining calculation. Basic mathematical operations make no sense on free text and real natural language sentences don't make good categorical fields either. As we discussed in section 3, *indexing* is the sort of preprocessing step that can translate unstructured text into a representation more edible for classic algorithms. Once past that hurdle, we're back in a scenario with just numeric and categorical fields, which means classic data mining algorithms can start crunching. And indeed, when looking at the algorithms often used for Text Classification, many classic algorithms and techniques reappear [4], such as Support Vector Machines, K-Nearest Neighbors, Neural Nets and the Naïve Bayes classifiers we already described in the previous section.

This means the technology setting Text Mining apart from Data Mining is the preprocessing step identifying the terms in the free text and attributing importance scores (of which frequency is the most basic one) and context to them. Thus, if the PMML standard can be extended to include specifications for an indexing step that translates a free text input field into derived fields containing the relevance metric (by default a simple frequency) for a given term, suddenly all PMML model types become available for Text Categorization and other scenarios involving text. This is, as was hinted in Section 2, what would make the whole PMML standard compatible with free text input, rather than being a standard that includes a single model type fit for use on indexed text (rather than free text). Also, it would more elegantly accommodate combining text and its context in the same predictive model, which was not explicitly nor implicitly allowed in the current Text Model specification. For example, this would enable transparently combining clinician's notes in free text with measurements such as blood pressure and lab results.

In the following subsections, we'll explore the indexing step in more detail and suggest an extension for the PMML vocabulary to specify indexing as part of a model definition.

4.2 A specification for indexing text

In this subsection, we'll propose a specification for indexing text as part of a PMML model definition, in order to allow model consumer technology to implement the full prediction operation, starting from the free text input. First, we will present a brief overview of indexing technology and its properties that need specification.

4.2.1 Technology overview

Indexing or text analysis technology exists in many different flavors and architectures. The most basic of indexing algorithms would be one that just cuts free text into single-word terms and counts their frequencies. In most commercial or open-source text analysis software, the indexing engine is part of a broader package or application, typically offered as a search engine such as Apache's Lucene, Oracle Secure Enterprise Search and the deployable versions of well-known web search engines such as Google Enterprise Search. Other text analysis software is more oriented towards certain use cases such as HP's Autonomy and Nuance, often with a specific industry focus. Only a small subset of these solutions offers the indexing engine as a standalone feature, through APIs that allow it to be embedded as part of a broader, custom application. Both Lucene and Oracle (through Oracle Text) have APIs for indexing, but these are very much oriented towards a search scenario and therefore might not offer the raw indexing results that are needed to implement the preprocessing step for text input to predictive models described in

the previous subsection. One exception is the iKnow technology distributed by InterSystems Corporation, which offers smart indexing APIs that were designed specifically for custom application development.

iKnow is a bottom-up text analysis technology that identifies multi-word concepts and the relationships between them in natural language, without requiring any predefined knowledge about the text's subject. On top of this indexing functionality, a number of analysis capabilities are built including support for matching against existing ontologies, intelligent browsing and text categorization. The technology has also been used in a number of research projects, both academic and with public and private organizations [12] [13] [14].

The crude word-counting algorithm presented earlier and iKnow find themselves at very different sides of the indexing solution spectrum, which makes them good candidates to consider when defining a specification for an indexing operation in PMML.

Term Selection and Term Extraction, both described in the previous section, are two techniques to reduce the high dimensionality typically resulting from straight indexing operations when building a model involving free text. For the model consumer, however, only the results of these operations are important. Term Selection simplifies the indexing step to be performed by limiting the terms to look for to a fixed subset. In the case of the PMML Text Model, this information was present already in the `TextDictionary` element and a similar, simple array representation should suffice for in a specification for an indexing operation. The transformation achieved by Term Extraction translates n input fields into m output fields where n > m. This operation can currently be expressed through normal PMML `DerivedField` elements and therefore needs no further attention.

The following aspects of an indexing operation will need to be addressed by this new specification:

- How to weigh and optionally normalize the raw frequencies of terms identified in the input text.
- Whether or not to perform *stemming*, standardizing terms that have the same morphological root. For example, whether "birds" and "bird" should be treated as the same term.
- Whether or not to detect multi-word terms (N-grams) and how to count these multi-word terms found in the text that only partially match a term defined in the model. For example, if a text mentions the term "bird flu", but only "flu" is defined as a term of interest in the model, how should this term be scored?

Opinions about the benefits of stemming and N-gram detection for Text Categorization seem to differ [15] [16] [17], but in general this is considered to be a tradeoff between semantic accuracy and statistical applicability. N-grams and non-stemmed words will be more precise, but their respective frequencies will be lower and variance higher than when only looking at single words and stemmed forms. Our own research seems to indicate that a precise semantic analysis and highly accurate N-gram detection such as provided by the iKnow technology, results in increased overall classification accuracy. This is in part thanks to iKnow's domain independence, yielding good N-gram detection across topics and industries. Results for clustering experiments (as a precursor to classification) incorporating not only the N-grams but also their relational context as provided by iKnow also seemed promising [12].

4.2.2 Proposed PMML extension

First, we propose introducing a new optype value "text" for representing free text, which cannot be directly used as an input for model types. Second, we'll introduce a new element representing the indexing preprocessing operation, to be nested in the transformation dictionary (local or global), as well as a new expression element for use in `DerivedField` elements. Figure 4 presents the definition of these new elements.

```
<xs:element name="TextIndex">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="Extension" minOccurs="0"
           maxOccurs="unbounded"/>
   <xs:group ref="STRING-ARRAY"/>
  </xs:sequence>
  <xs:attribute name="name" type="FIELD-NAME"
           use="required"/>
  <xs:attribute name="field" type="FIELD-NAME"/>
  <xs:attribute name="description"
           type="xs:string"/>
  <xs:attribute name="language" type="xs:string"/>
  <xs:attribute name="applyStemming"
           type="xs:boolean" default="false"/>
  <xs:attribute name="nGramPolicy"
           default="fullMatchOnly"/>
   <xs:simpleType>
    <xs:restriction base="xs:string">
     <xs:enumeration value="fullMatchOnly"/>
     <xs:enumeration value="acceptPartialMatch"/>
     <xs:enumeration value="scalePartialMatch"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="localTermWeights"
           default="termFrequency">
   <xs:simpleType>
    <xs:restriction base="xs:string">
     <xs:enumeration value="termFrequency"/>
     <xs:enumeration value="binary"/>
     <xs:enumeration value="logarithmic"/>
     <xs:enumeration value="
           augmentedNormalizedTermFrequency"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="documentNormalization"
           default="none">
   <xs:simpleType>
    <xs:restriction base="xs:string">
     <xs:enumeration value="none"/>
     <xs:enumeration value="cosine"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
 </xs:complexType>
</xs:element>
<xs:element name="TextIndexTerm">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Extension" minOccurs="0"
           maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="index" type="xs:string"
           use="required"/>
    <xs:attribute name="term" type="xs:string"
           use="required"/>
  </xs:complexType>
</xs:element>
```

Figure 4: TextIndex and TextIndexTerm element specification

The new `TextIndex` element defines the indexing operation itself. It contains an `Array` element listing all the terms to be identified and has a `name` attribute that's unique across the PMML file. The `field` attribute should refer to an existing field which has optype="text". The `language` attribute is optional and should either correspond to a two-letter ISO 639-1 language code, or be empty to indicate it's up to the indexing engine to identify the language. If `stemming` is set to true, the *stemmed* terms found in the input text should correspond to the terms defined in the `Array`. If the model consumer does not support stemming, this attribute should be ignored[6].

The values of `nGramPolicy` define how to treat partial matches if the indexing technology used by the model consumer supports N-gram detection. If it does not support N-grams, behavior defaults to "fullMatchOnly"[6]. A partial match is defined as the occurrence of an N-gram in the text that is longer than a term in `Array`, but contains all the words of that term, irrespective of word order. For example if the N-gram "diet coke" occurs in the text, it is a partial match for "coke", but not for "cheap diet coke".

- **fullMatchOnly**: only exact matches of the terms in `Array` are accepted and counted as a single occurrence, contributing 1 to the term frequency.
- **acceptPartialMatch**: both partial and exact matches are accepted and counted as a single occurrence of that term, contributing 1 to the term frequency.
- **scalePartialMatch**: both partial and exact matches are accepted, but contribute to the term frequency proportionally to the number of words matching. For example, if "diet coke" is a term in `Array`, an N-gram "diet coke" will contribute 1 and "cheap diet coke" will contribute 0.666.

As for term weighting and normalization options, the same options as described in section 3.2.3 are offered. Normalization happens based on the contents of the array of terms in the `TextIndex` element, regardless of whether all of these terms were referred through `DerivedField` elements.

For the `TextIndexTerm` elements, as nested in `DerivedField` elements, their `index` attribute should correspond to the name of a `TextIndex` in the same model's `LocalTransformations` or the `TransformationDictionary` and the `term` attribute should correspond to one of the terms in that `TextIndex`' `Array` of terms.

Note: the extensions proposed in this section are independent of the ones presented in Section 3.

4.2.3 Example

```
<PMML>
 <DataDictionary numberOfFields="2">
  <DataField name="weatherReport" optype="text"
             dataType="string"/>
  <DataField name="temperature" dataType="double"
             optype="continuous" />
 </DataDictionary>
 <TransformationDictionary>
  <TextIndex name="index" text="weatherReport"
             localTermWeights="binary">
   <Array type="string">sunny rainy</Array>
  </TextIndex>
  <DerivedField name="isSunny" >
   <TextIndexTerm index="index" term="sunny"/>
  </DerivedField>
  <DerivedField name="isRainy" >
   <TextIndexTerm index="index" term="rainy"/>
  </DerivedField>
 </TransformationDictionary>
 <NaiveBayesModel ... />
</PMML>
```

Figure 5: Example use of `TextIndex` and `TextIndexTerm`

4.2.4 Further extensions

The extensions presented in this section are meant to offer a flexible way of using text as an input for all PMML model types. While this already opens up a number of interesting possibilities and use cases, here are a number of additions that could further enhance PMML's capabilities for tackling text.

- Similar to the (currently unspecified) use of a `Taxonomy` element in the Text Models `TextDicationary`, it could be introduced at the level of a `TextIndex` to aggregate synonyms and related terms into a topic hierarchy. Using the higher-level terms would then simplify models using text by further reducing dimensionality in a transparent way. One example of using external taxonomies or ontologies we have found to be particularly useful when categorizing text in a clinical domain is mapping text input to High Level Concepts of the Unified Medical Language System® (UMLS) [7].
- Some text analysis solutions, such as the iKnow technology described earlier, support negation detection. An appropriate weighting of terms that appear in a negated context (such as "pain" in "the patient felt no pain") could further enhance the accuracy of text-aware models.
- Support for regular expressions as an alternative to literal term matching (partial and exact) could also increase the flexibility of this new capability. However, attention needs to be paid to appropriately identifying and weighting regular expression matches.

5. CONCLUSION

In this paper, we have introduced Text Categorization as a practical example of predictive analytics. We have presented the current specification of the Text Model as it appears in PMML version 4.1 and reviewed its applicability for expressing Text Categorization scenarios. This revealed a number of issues we believe limit its chances for a broader adoption in the market, despite the growing importance of text as an input for predictive models. To accommodate these challenges, we proposed an extension to the PMML Text Model that should make it a flexible standard for expressing Text Categorization models. Finally, we have discussed text indexing in more detail and proposed a separate set of extensions to implement text indexing as a dedicated transformation of free text input into normal, numeric input fields. This proposed specification assumes a text indexing engine is available to the model consumer, which could range from a simple word-counting algorithm to feature-rich text analysis software such as the iKnow technology presented in this paper. This second extension allows broadening the scope of every PMML model type to accept text input and implement Text Categorization and other scenarios involving free text alongside classic categorical and numerical fields.

[6] It is recommended the model consumer technology throws a warning if it is presented a PMML definition containing unsupported indexing features.

[7] See also: http://www.nlm.nih.gov/research/umls/

6. REFERENCES

[1] A. Guazzelli, M. Zeller, W. Lin, G. Williams. 2009. PMML: An Open Standard for Sharing Models. *The R Journal*, Volume 1/1, May 2009

[2] A. Guazzelli, W. Lin, T. Jena. 2010. PMML in Action: Unleashing the Power of Open Standards for Data Mining and Predictive Analytics. *CreativeSpace*

[3] R. Pechter. 2011. PMML Conformance Progress Report – Five years later. *KDD 2011, Proceedings of the 2011 workshop on Predictive markup language modeling*. DOI= http://dx.doi.org/10.1145/2023598.2023599

[4] F. Sebastiani. 2002. Machine Learning in Automated Text Categorization. *ACM Computing Surveys*, Vol. 34, No. 1, March 2002 DOI= http://dx.doi.org/10.1145/505282.505283

[5] G. Salton, A. Wong, and C. S. Yang. 1975. A Vector Space Model for Automatic Indexing. *Communications of the ACM*, Vol. 18, nr. 11, pages 613–620.

[6] P. D. Turney, P. Pantel. 2010. From Frequency to Meaning: Vector Space Models of Semantics. *Journal of Artificial Intelligence research*. Vol 37, pages 141-188

[7] E. Chisholm, T. G. Kolda. 1999. New Term Weighting Formulas for the Vector Space Method in Information Retrieval. *Computer Science and Mathematics Division, Oak Ridge National Laboratory*.

[8] P. Soucy, G. W. Mineau. 2005. Beyond TFIDF weighting for text categorization in the vector space model. *International Joint Conference on Artificial Intelligence*. Vol. 19.

[9] J. P. Benzécri. 1992. Correspondence analysis handbook. *Marcel Dekker Inc.*

[10] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, R. Harshman. 1990. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*. Vol 41. Pages 391-407

[11] D. D. Lewis. 1998. Naive (Bayes) at forty: The independence assumption in information retrieval. *Proceedings of ECML-98, 10th European Conference on Machine Learning*. Pages 4–15.

[12] A. Bronselaer, S. Debergh, D. Van Hyfte, G. De Tré. 2010. Text clustering based on concept-relational decomposition. *ICL 2010 Proceedings*. Pages 357–359

[13] D. Van Hyfte, M. Bouzinier, M. Tsatulin, S. Richards, K. Lee, C. Almond. 2013. Mining medical texts for cancer intelligence using iKnow. *Cancer Outcomes Conference 2013*.

[14] M.C. Hazewinkel, E. Hoencamp, R.F.P. de Winter, D. Wijnschenk, D. van Hyfte. 2013. Voorspellers van separatie door tekstanalyse van de verslaglegging in het elektronisch patiëntendossier (in Dutch). *Voorjaarsconferentie, Nederlandse Vereniging voor Psychiatrie*. Page 33 (Poster presentation)

[15] D. D. Lewis. 1992. An evaluation of phrasal and clustered representations on a text categorization task. *Proceedings of SIGIR-92, 15th ACM International Conference on Research and Development in Information Retrieval*. Pages 37–50.

[16] M. F. Caropreso, S. Matwin, F. Sebastiani. 2001. A learner-independent evaluation of the usefulness of statistical phrases for automated text categorization. *Text Databases and Document Management: Theory and Practice*. Pages 78–102

[17] D. Mladenic, M. Grobelnik. 1998. Word sequences as features in text-learning. *Proceedings of ERK-98, the Seventh Electrotechnical and Computer Science Conference*. Pages 145–148.