root

Go Up

Name	TextVectors
Version	1.0.0
Description	Text Vectorization for words and sentences
License	http://www.apache.org/licenses/LICENSE-2.0
Copyright	Copyright (C) 2019 HPCC Systems
Authors	HPCCSystems
DependsOn	ML_Core 3.2.0
Platform	7.0.0

Table of Contents

SentenceVectors

Go Up

IMPORTS

Types | Internal | std.Str | std.system.Thorlib | Internal.svUtils |

DESCRIPTIONS

MODULE SentenceVectors

	SentenceVectors
	`(UNSIGNED2 vecLen=100, REAL trainToLoss = .05, UNSIGNED4 numEpochs= 0, UNSIGNED4 batchSize=0, UNSIGNED4 negSamples=10, REAL4 learningRate=.2, REAL4 discardThreshold = .0001, UNSIGNED4 minOccurs = 10, UNSIGNED4 wordNGrams = 1, UNSIGNED4 dropoutK = 3, UNSIGNED4 noProgressEpochs = 1, UNSIGNED4 maxTextDistance = 3, UNSIGNED4 maxNumDistance = 9, BOOLEAN saveSentences=TRUE)`

Module to learn and manipulate sentence and word vectors.

It implements the Sent2Vec algorithm from "Unsupervised Learning of Sentence Embeddings using Compositional n-Gram Features" by Matteo Pagliardini, Prakhar Gupta and Martin Jaggi (https://arxiv.org/abs/1703.02507)

Text vectors reduce words, paragraphs, sentences or phrases to a vector of numbers that are best thought of as coordinates in N-dimensional space. A typical vector size of 100 provides coordinates in 100 dimensional space. Vectorization takes advantage of a linguistic theory that says that "a word is known by the company it keeps". This suggests that words that tend to occur in the same context (i.e. with the same surrounding words) have a similar meaning. This is known as "contextual meaning", which is very difficult to distinguish from our inherent understanding of "meaning". Assuming we have a large Corpus (i.e. the set of all the text in our domain – e.g. All of wikipedia, all medical journals), "contextual meaning" and "meaning" are nearly synonymous. Consider two words that are always found in the exact same context: It would be surprising if those two words were not very closely related since there is, in

effect, nothing that can be (or at least has been) said of one that can't be (hasn't been) said about the other.

Vectors that are close together (in N dimensional space) will have similar contextual meaning. Thus, synonyms can be readily found for words, and sentences can be identified as similar, even when they use very different wording.

Sentence Vectors is a very fast mechanism (compared to other mechanisms) for learning high-quality vectors for both words and sentences. Sentences are vectorized by taking the average vector for all of the words in the sentence. Word and Sentence Vectors can be used in a number of ways: - They form low-dimensional representations that can be used as input features to supervised machine learning tasks (e.g. to categorize the meaning of a sentence). The attributes GetWordVectors() and GetSentVectors() can be used to extract the learned vectors. - Words can be analyzed for similarity of meaning by utilizing ClosestWords() or by retrieving the vectors and calling Similarity(). - A sentence not in the training set can be compared to sentences in the training set using ClosestSentences() or by retrieving the vectors and calling Similarity(). - Word analogies of the form A is to B as C is to ? can be solved by calling WordAnalogy(). - Collections of words can be analyzed to find outliers using FindLeastSimilar().

Sentence Vectors can be configured to incorporate word order and multi-word concepts using the wordNGrams feature. When set to one, only individual words are considered, while when set to 3, one, two, and three word phrases are all considered words. In this way a phrase like "traffic light" can have a very different meaning than either "traffic" or "light", and can also have a very different meaning from "light traffic". "New York Times" can have a different meaning than "new", "york" or "times", as well as a different meaning than "New York" or "York Times". With wordNGrams set to 1, compound words as well as word order cannot be distinguished. This can be useful for corpora that contain very terse cryptic sentences which may be equally valid in any order, and without compound word usage.

Sentence Vectors are trained using a specialized Neural Network that is task-optimized. Batch Synchronous Stochastic Gradient Descent is used to parallelize the neural network training across the HPCC Cluster.

PARAMETER **saveSentences** ||| BOOLEAN — Set to TRUE to save the training sentences in the model. It should always be set to TRUE if you plan to call GetClosestSentences(...). Setting to FALSE will result in a smaller model with only the word vectors (default TRUE).

PARAMETER **numEpochs** ||| UNSIGNED4 — The maximum number of times to loop through the full set of training data. Each pass through the training data is known as an Epoch. Default 0 (recommended) means no fixed number of epochs. Will use trainToLoss and noProgressEpochs to terminate. Note that the larger the corpus, the lower this value can become. For an extremely large corpus, a single Epoch may be enough.

PARAMETER **negSamples** ||| UNSIGNED4 — The number of random negative samples to generate for each positive sample (default 10). This may be reduced to as little as 2 for an extremely large corpus.

PARAMETER **maxTextDistance** ||| UNSIGNED4 — This implementation provides an advanced feature that handles comparison sentences with words that were not found in the trained vocabulary. The system attemts to match previously unseen words to words in the vocabulary using edit distance. The goal is to handle typos, misspellings, and initialisms (e.g. 'dog' vs 'dogs'). Edit distance allows for character level inserts, removals, or replacements. If this number is too

high, unrelated words may be matched. If too low, typos may not be mapped (Default 3). Zero implies no edit distance matching, in which case unseen words will be ignored.

PARAMETER **minOccurs** ||| UNSIGNED4 — Words that occur below this number of times in the corpus will be ignored (default 10). Rare words with few occurrences in the corpus generally add little to the accuracy and can significantly extend training time.

PARAMETER **dropoutK** ||| UNSIGNED4 — The number of NGrams to randomly drop from a sentence (per Sent2Vec paper). Default 3.

PARAMETER **learningRate** ||| REAL4 — The rate at which the neural network learns (default .1).

PARAMETER **wordNGrams** ||| UNSIGNED4 — The maximum size NGrams to consider. For example, 3 would cause all Unigrams, Bigrams, and Trigrams to be used (Default 1 – Unigrams only).

PARAMETER **batchSize** ||| UNSIGNED4 — The number of word pairs to process before synchronizing weights across the nodes during Stochastic Gradient Descent (SGD) (Default 0 – auto assign – recommended). Note that larger values imply longer periods between synchronization of weights across nodes. Larger values also speed up the training as less time is spent synchronizing. If the network fails to converge, this may be set too high.

PARAMETER **maxNumDistance** ||| UNSIGNED4 — If a number was seen that was not in the vocabulary, the system will attempt a numeric match rather than an edit distance match. The numbers 10 and 100 have an edit distance of only 1 (i.e. add a zero), yet the meaning can be vastly different. For numeric words, we use the actual numeric distance instead of edit distance. For example, if this parameter is set to 9, then 43 will match to 50 if it is the closest number. A value of zero disables numeric matching (Default 9).

PARAMETER **noProgressEpochs** ||| UNSIGNED4 — Controls early stopping of training. Determines how many Epochs can go by without progress before training is terminated. There is generally no advantage to continue training once we fail to make progress on each epoch, since accuracy does not typically go up with overtraining.

PARAMETER **vecLen** ||| UNSIGNED2 — The length in bytes of the sentence and word vectors to be learned (default 100).

PARAMETER **trainToLoss** ||| REAL8 — Stop training when the average Loss level reaches this level. This is the recommended method of controlling training duration. The Default .05 (i.e. 5 percent loss) seems to give good results.

PARAMETER **discardThreshold** ||| REAL4 — The threshold of word frequency below which each occurrence of the word will be trained on. For frequencies above this level, the word will be stochastically discarded to reduce over-training of common words and to speed up training (default .0001).

Children

1. GetModel : Train and return a model
2. GetTrainStats : Return the parameters under which the model was generated

3. GetSentVectors : Obtain sentence vectors for a set of sentences given a previously trained model
4. GetWordVectors : Obtain word vectors for a set of words
5. ClosestWords : Find the closest words in the training set, given a set of words
6. LeastSimilarWords : Find the outlier(s) given a set of words
7. WordAnalogy : Solves an analogy of the form: A is to B as C is to ?
8. ClosestSentences : Find the closest sentence or sentences for each test sentence
9. Similarity : Compute the similarity level between two word or sentence vectors

---

FUNCTION GetModel

SentenceVectors \

DATASET(TextMod)	**GetModel**
(DATASET(Sentence) sentences)	

Train and return a model. The model consists of a word portion, containing all the words in the vocabulary along with their vectors, and a sentence portion containing all of the sentences in the corpus along with their vectors.

PARAMETER **sentences** ||| TABLE (Sentence) — The sentences comprising the corpus to be trained in Types.Sentence format.

RETURN **TABLE ({ UNSIGNED1 typ , UNSIGNED8 id , STRING text , SET (REAL8) vec })** — A model in DATASET(TextMod) format.

SEE Types.Sentence

FUNCTION GetTrainStats

SentenceVectors \

	GetTrainStats
(DATASET(TextMod) mod)	

Return the parameters under which the model was generated. This function should only be called during the training phase, or when the module was invoked with the same parameters used in the training phase. It may return incorrect information if called with a previously persisted model when the module was invoked e.g. with default values.

PARAMETER **mod** ||| TABLE (TextMod) — A model as previously returned from GetModel.

RETURN **TABLE ({ UNSIGNED4 vecLen , UNSIGNED4 nWeights , UNSIGNED4 nSlices , UNSIGNED4 sliceSize , UNSIGNED4 nWords , UNSIGNED8 nSentences , UNSIGNED4 maxNGramSize , UNSIGNED4 nEpochs , UNSIGNED4 negSamples , UNSIGNED4 batchSize , UNSIGNED4 minOccurs , UNSIGNED4 maxTextDist , UNSIGNED4 maxNumDist , REAL4 discardThreshold , REAL4 learningRate , UNSIGNED4 upb , UNSIGNED4 upbPerNode , REAL4 updateDensity , REAL4 udPerNode })** — A DATASET(Types.TrainStats) containing a single record.

SEE Types.TrainStats

FUNCTION GetSentVectors

SentenceVectors \

DATASET(SentInfo)	GetSentVectors
(DATASET(TextMod) mod, DATASET(Sentence) sentences)	

Obtain sentence vectors for a set of sentences given a previously trained model.

PARAMETER **sentences** ||| TABLE (Sentence) — The set of sentences to vectorize in Types.Sentence format.

PARAMETER **mod** ||| TABLE (TextMod) — A model as previously returned from GetModel

RETURN **TABLE ({ UNSIGNED8 sentId , STRING text , SET (REAL8) vec })** — A DATASET(SentInfo) containing the vectors for each sentence.

SEE Types.Sentence

SEE Types.SentInfo

FUNCTION GetWordVectors

SentenceVectors \

DATASET(WordInfo)	GetWordVectors
(DATASET(TextMod) mod, DATASET(Word) words)	

Obtain word vectors for a set of words.

PARAMETER **words** ||| TABLE (Word) — The set of words to vectorize in Types.Word format.

PARAMETER **mod** ||| TABLE (TextMod) — A model as previously returned from GetModel

RETURN **TABLE ({ UNSIGNED4 wordId , STRING text , UNSIGNED4 occurs , REAL8 pdisc , SET (REAL8) vec })** — A DATASET(Types.WordInfo) containing the vectors for each word.

SEE Types.Word

SEE Types.WordInfo

FUNCTION ClosestWords

SentenceVectors \

DATASET(Closest)	ClosestWords
(DATASET(TextMod) mod, DATASET(Word) words, UNSIGNED1 N=1, BOOLEAN showNGrams=FALSE)	

Find the closest words in the training set, given a set of words.

For each word in the provided set, find the most similar words in the training set. Note that the requested word is never returned. It is filtered out of the results and only words close to but not equal to that word are returned.

PARAMETER **showNGrams** ||| BOOLEAN — True if NGrams (n greater than 1) are to be considered in the matching (Default FALSE).

PARAMETER **N** ||| UNSIGNED1 — The number of closest matches to return (Default 1).

PARAMETER **words** ||| TABLE (Word) — A dataset of Types.Word with the words to be matched.

PARAMETER **mod** ||| TABLE (TextMod) — A model as returned from GetModel.

RETURN **TABLE ({ UNSIGNED8 id , STRING text , SET (STRING) closest , SET (REAL8) similarity })** — A dataset of Types.Closest showing the closest words to each test word as well as their numeric similarity (i.e. Cosine Similarity).

SEE Types.Closest

FUNCTION LeastSimilarWords

SentenceVectors \

DATASET(Word)	**LeastSimilarWords**
(DATASET(TextMod) mod, DATASET(Word) words, UNSIGNED1 N=1)	

Find the outlier(s) given a set of words.

Implements "One of these things is not like the other".

PARAMETER **N** ||| UNSIGNED1 — The number of least similar words to return (Default 1).

PARAMETER **words** ||| TABLE (Word) — A list of words from which to find the least similar.

PARAMETER **mod** ||| TABLE (TextMod) — A model as returned from GetModel.

RETURN **TABLE ({ UNSIGNED4 id , STRING text })** — A list of words (Types.Word) in order of increasing similarity.

SEE Types.Word

FUNCTION WordAnalogy

SentenceVectors \

DATASET(Closest)	**WordAnalogy**
(DATASET(TextMod) mod, STRING A, STRING B, STRING C, UNSIGNED2 N=1, BOOLEAN showNGrams=FALSE)	

Solves an analogy of the form: A is to B as C is to ?

PARAMETER **showNGrams** ||| BOOLEAN — TRUE to include NGrams (N greater than 1) in the list of solutions.

PARAMETER **B** ||| STRING — The 'isTo' word.

PARAMETER **A** ||| STRING — The main word.

PARAMETER **N** ||| UNSIGNED2 — The number of best matches to return (Default 1).

PARAMETER **C** ||| STRING — The 'as' word.

PARAMETER **mod** ||| TABLE (TextMod) — No Doc

RETURN **TABLE ({ UNSIGNED8 id , STRING text , SET (STRING) closest , SET (REAL8) similarity })** — A set of best solutions in Types.Closest format.

SEE Types.Closest

FUNCTION ClosestSentences

SentenceVectors \

DATASET(Closest)	**ClosestSentences**
(DATASET(TextMod) mod, DATASET(Sentence) sentences, UNSIGNED1 N=1)	

Find the closest sentence or sentences for each test sentence.

PARAMETER **N** ||| UNSIGNED1 — The number of closest matches to return for each test sentence.

PARAMETER **sentences** ||| TABLE (Sentence) — The list of test sentences in Types.Sentence format.

PARAMETER **mod** ||| TABLE (TextMod) — A model as returned from GetModel.

RETURN **TABLE ({ UNSIGNED8 id , STRING text , SET (STRING) closest , SET (REAL8) similarity })** — A list of closest matches for each test sentence in Types.Closest format.

SEE Types.Closest

SEE Types.Sentence

FUNCTION Similarity

SentenceVectors \

REAL4	**Similarity**
`(t_Vector vec1, t_Vector vec2)`	

Compute the similarity level between two word or sentence vectors.

The Cosine Similarity of the two vectors is returned. This is a number between -1 and 1, where 1 implies exact similarity, zero implies orthogonality (i.e. non-correlation) meaning the words are unrelated. Negative values imply a sort of inverse correlation, although they do not necessarily imply antonymity. Note that this is a symmetric function so the order of the vectors is not significant.

PARAMETER **vec2** ||| SET (REAL8) — The second vector in Types.t_Vector format.

PARAMETER **vec1** ||| SET (REAL8) — The first vector in Types.t_Vector format.

RETURN **REAL4** — The Cosine Similarity between the two vectors.

SEE Types.t_Vector

Types

Go Up

DESCRIPTIONS

MODULE Types

	Types

Common Type definitions for TextVectors bundle

Children

1. t_WordId : Type definition for a Word Id attribute
2. t_SentId : Type definition for a Sentence Id attribute
3. t_TextId : Type definition for an attribute that can hold either a Word Id
4. t_Vector : Type definition for a vector attribute (used for Word or Sentence Vectors)
5. t_Word : Type definition for the text of a Word
6. t_Sentence : Type definition for the text of a Sentence
7. t_ModRecType : Enumeration of the record type for a Text Model
8. Sentence : Dataset Record to hold a Sentence
9. SentInfo : Sentence Information record, including the sentence vector
10. Word : Dataset Record to hold a Word
11. WordInfo : Dataset record to hold information about a word, including its vector, number of occurrences in the corpus, and discard probability
12. Vector : Dataset record to hold a sentence or word vector
13. TextMod : Record definition for the TextVectors Model
14. Closest : Used to return a set of closest items (words or sentences) for each of a given set of items to match

15. TrainStats : Record to hold the return values from GetTrainStats function

---

ATTRIBUTE t_WordId

Types \

	t_WordId

Type definition for a Word Id attribute

RETURN UNSIGNED4 —

ATTRIBUTE t_SentId

Types \

	t_SentId

Type definition for a Sentence Id attribute.

RETURN UNSIGNED8 —

ATTRIBUTE t_TextId

Types \

	t_TextId

Type definition for an attribute that can hold either a Word Id. or a Sentence Id.

RETURN **UNSIGNED8** —

ATTRIBUTE t_Vector

Types \

	t_Vector

Type definition for a vector attribute (used for Word or Sentence Vectors).

RETURN **SET (REAL8)** —

ATTRIBUTE t_Word

Types \

	t_Word

Type definition for the text of a Word.

RETURN **STRING** —

ATTRIBUTE t_Sentence

Types \

	t_Sentence

Type definition for the text of a Sentence.

RETURN **STRING** —

ATTRIBUTE t_ModRecType

Types \

	t_ModRecType

Enumeration of the record type for a Text Model.

RETURN **UNSIGNED1** —

RECORD Sentence

Types \

	Sentence

Dataset Record to hold a Sentence.

FIELD **sentId** ||| UNSIGNED8 — The numeric record id for this sentence.

FIELD **text** ||| STRING — The text content of the sentence.

RECORD SentInfo

Types \

	SentInfo

Sentence Information record, including the sentence vector.

FIELD **sentId** ||| UNSIGNED8 — The numeric record id for this sentence.

FIELD **text** ||| STRING — The text content of the sentence.

FIELD **vec** ||| SET (REAL8) — The Text Vector for the sentence.

RECORD Word

Types \

	Word

Dataset Record to hold a Word.

FIELD **text** ||| STRING — The text content of the word.

FIELD **id** ||| UNSIGNED4 — The numeric record id for this word.

RECORD WordInfo

Types \

	WordInfo

Dataset record to hold information about a word, including its vector, number of occurrences in the corpus, and discard probability.

FIELD **pdisc** ||| REAL8 — The computed probability of discard, based on the frequency of the word in the corpus.

FIELD **occurs** ||| UNSIGNED4 — The number of times this word occurs in the Corpus.

FIELD **text** ||| STRING — The text content of the sentence.

FIELD **wordId** ||| UNSIGNED4 — The numeric record id for this word.

FIELD **vec** ||| SET (REAL8) — The Text Vector for the word.

RECORD Vector

Types \

	Vector

Dataset record to hold a sentence or word vector.

FIELD **vec** ||| SET (REAL8) — The contents of the vector.

FIELD **id** ||| UNSIGNED4 — The record id for this vector.

RECORD TextMod

Types \

	TextMod

Record definition for the TextVectors Model. Text Model contains both the word and sentence vectors for the trained corpus.

FIELD **text** ||| STRING — The textual content of the item (word or sentence).

FIELD **id** ||| UNSIGNED8 — The id of the word or sentence.

FIELD **vec** ||| SET (REAL8) — The vector for the word or sentence.

FIELD **typ** ||| UNSIGNED1 — The type of the record – Word or Sentence (see t_ModRecType above).

RECORD Closest

Types \

	Closest

Used to return a set of closest items (words or sentences) for each of a given set of items to match. Closest contains the set of closest items, while Similarity is the cosine similarity between the text sentence and each of the closest items. For example, Similarity[1] is the Cosine similarity between Text and Closest[1]. Likewise for each of the K items in Closest and Similarity.

FIELD **similarity** ||| SET (REAL8) — The cosine similarity between this word / sentence and each of the K closest words or sentences. This set corresponds 1:1 to the contents of the 'closest' field.

FIELD **text** ||| STRING — The text of the word or sentence

FIELD **closest** ||| SET (STRING) — The text of the K closest words or sentence.

FIELD **id** ||| UNSIGNED8 — The id of the word or sentence.

RECORD TrainStats

Types \

	TrainStats

Record to hold the return values from GetTrainStats function. This records describes the set of parameters used for the current training session.

FIELD **nWeights** ||| UNSIGNED4 — The number of weights needed to train the word vectors.

FIELD **upb** ||| UNSIGNED4 — Updates per batch. The approximate number of weights that are updated across all nodes during a single batch.

FIELD **nSlices** ||| UNSIGNED4 — The number of slices used to hold the weights.

FIELD **batchSize** ||| UNSIGNED4 — The batch size used to train the vectors. Zero (default) indicates auto-compute.

FIELD **upbPerNode** ||| UNSIGNED4 — Updates per batch per node. The number of weights updated by each node during a single batch.

FIELD **negSamples** ||| UNSIGNED4 — The number of negative samples used in training for each training sample.

FIELD **nWords** ||| UNSIGNED4 — The number of words in the vocabulary including N-Grams.

FIELD **maxNumDist** ||| UNSIGNED4 — The maximum numeric distance to consider one previously unseen number a match for a number in the vocabulary.

FIELD **minOccurs** ||| UNSIGNED4 — The minimum number of occurrences in the Corpus in order for a word to be considered part of the vocabulary.

FIELD **sliceSize** ||| UNSIGNED4 — The number of weights in each weight Slice.

FIELD **learningRate** ||| REAL4 — The learning rate used to train the Neural Network.

FIELD **updateDensity** ||| REAL4 — The proportion of weights updated across all nodes during a single batch.

FIELD **maxNGramSize** ||| UNSIGNED4 — The maximum N-Gram size to consider.

FIELD **nSentences** ||| UNSIGNED8 — The number of sentences in the Corpus.

FIELD **maxTextDist** ||| UNSIGNED4 — The maximum number of edits (in edit distance) to make in matching a previously unseen word to a word in the vocabulary.

FIELD **udPerNode** ||| REAL4 — The proportion of weights updated by a single node during a single batch.

FIELD **vecLen** ||| UNSIGNED4 — The dimensionality of the word and sentence vectors.

FIELD **nEpochs** ||| UNSIGNED4 — The maximum number of epochs for which to train. Zero (default) means auto-compute.

FIELD **discardThreshold** ||| REAL4 — Words with frequency below this number are never discarded from training data. Words with frequency above this number are stochastically sampled, based on their frequency.

Internal

Go Up

Table of Contents

Corpus

Go Up

IMPORTS

Types | std.Str | std.system.Thorlib |

DESCRIPTIONS

MODULE Corpus

	Corpus
	`(DATASET(Sentence) sentences_in=DATASET([], Sentence), UNSIGNED4 wordNGrams = 2, REAL4 discThreshold = .0001, UNSIGNED4 minOccurs = 5, UNSIGNED4 dropoutK = 3)`

Analyze a corpus of sentences to support vectorization activities.

Tokenizes sentences into words, provides a Vocabulary of unique words, and supports conversion of sentences into training data.

PARAMETER **discThreshold** ||| REAL4 — Discard threshold. Words with frequency greater than or equal to this number are probabilistically discarded based on their frequency. Words with frequencies below this threshold are never discarded (Default .0001).

PARAMETER **minOccurs** ||| UNSIGNED4 — Words that occur less than this number of times in the corpus are eliminated (Default 5). Words with very few occurrences in the corpus may not get properly trained due to lack of context.

PARAMETER **dropoutK** ||| UNSIGNED4 — The number of NGrams to drop from a sentence (per Sent2Vec paper). Default 5.

PARAMETER **wordNGrams** ||| UNSIGNED4 — The maximum sized NGram to generate. 1 indicates unigrams only. 2 indicates unigrams and bigrams. 3 indicates uni, bi, and trigrams. Defaults to 1 (unigrams only).

PARAMETER **sentences_in** ||| TABLE (Sentence) — No Doc

Children

1. sentences : Return a dataset of Sentences, distributed evenly
2. getNGrams : Produce a series of nGrams from the set of words in a sentence
3. sent2wordList : Convert a sentence to a list of words (including n-grams if requested)
4. tokenizedSent : Each sentence transformed to a word list
5. VocabSize : The size of the vocabulary
6. wordCount : No Documentation Found
7. Vocabulary : The set of all the unique words in the corpus
8. wordIdList : No Documentation Found
9. WordList2WordIds : Convert a list of textual words making up a sentence to a set of ids representing each word's wordId in the vocabulary
10. GetTraining : Generate training data based on the corpus
11. NegativesTable : Negatives Table is a record containing the discard probability of each word in the vocabulary as a single SET, indexed by the wordId

---

ATTRIBUTE sentences

Corpus \

	sentences

Return a dataset of Sentences, distributed evenly.

RETURN **TABLE (Sentence)** —

EMBED getNGrams

Corpus \

SET OF VARSTRING	**getNGrams**
(SET OF VARSTRING words, UNSIGNED4 ngrams, UNSIGNED4 dropoutk = 0)	

Produce a series of nGrams from the set of words in a sentence. For example, if the parameter ngrams is three, it will produce the set of Bigrams (i.e. 2grams) as well as the set of Trigrams (i.e. 3grams). Ngrams are formatted as _Word1_Word2_Word3. Given the sentence ['the', 'quick', 'brown', 'fox'] and ngrams set to 3, it will return: ['_the_quick', '_quick_brown', '_brown_fox', '_the_quick_brown', '_quick_brown_fox'].

PARAMETER **ngrams** ||| UNSIGNED4 — No Doc

PARAMETER **dropoutk** ||| UNSIGNED4 — No Doc

PARAMETER **words** ||| SET (VARSTRING) — No Doc

RETURN **SET (VARSTRING)** —

FUNCTION sent2wordList

Corpus \

DATASET(WordList)	**sent2wordList**
(DATASET(Sentence) sent)	

Convert a sentence to a list of words (including n-grams if requested). Strip out punctuation, cleanup whitespace, and split the words.

PARAMETER **sent** ||| TABLE (Sentence) — No Doc

RETURN **TABLE ({ UNSIGNED4 sentId , SET (STRING) words })** —

ATTRIBUTE tokenizedSent

Corpus \

DATASET(WordList)	**tokenizedSent**

Each sentence transformed to a word list.

RETURN **TABLE ({ UNSIGNED4 sentId , SET (STRING) words }) —**

ATTRIBUTE VocabSize

Corpus \

	VocabSize

The size of the vocabulary

RETURN **INTEGER8 —**

ATTRIBUTE wordCount

Corpus \

	wordCount

No Documentation Found

RETURN **INTEGER8 —**

ATTRIBUTE Vocabulary

Corpus \

	Vocabulary

The set of all the unique words in the corpus. Note: returned vocabulary is distributed by HASH32(text), and sorted by text.

RETURN TABLE ({ UNSIGNED4 wordId , STRING text , UNSIGNED4 occurs , REAL8 pdisc , SET (REAL8) vec }) —

RECORD wordIdList

Corpus \

	wordIdList

No Documentation Found

FIELD pdisc ||| REAL4 — No Doc

FIELD wordids ||| SET (UNSIGNED4) — No Doc

FIELD ord ||| UNSIGNED2 — No Doc

FIELD sentid ||| UNSIGNED4 — No Doc

FUNCTION WordList2WordIds

Corpus \

DATASET(wordIdList)	WordList2WordIds
(DATASET(WordList) sent)	

Convert a list of textual words making up a sentence to a set of ids representing each word's wordId in the vocabulary.

PARAMETER **sent** ||| TABLE (WordList) — No Doc

RETURN **TABLE ({ UNSIGNED4 sentId , UNSIGNED2 ord , SET (UNSIGNED4) wordIds , REAL4 pdisc }) —**

ATTRIBUTE GetTraining

Corpus \

DATASET(TrainingDat)	**GetTraining**

Generate training data based on the corpus. Each record is a main word and a set of context words (words that occur with that word in a sentence).

RETURN **TABLE ({ UNSIGNED4 main , SET (UNSIGNED4) context }) —**

ATTRIBUTE NegativesTable

Corpus \

	NegativesTable

Negatives Table is a record containing the discard probability of each word in the vocabulary as a single SET, indexed by the wordId. It is not currently used but is left here for possible future use.

RETURN **TABLE ({ UNSIGNED2 nodeId , SET (REAL4) probs }) —**

Internal/

SGD

Go Up

IMPORTS

Types | std.system.Thorlib | Internal | std |

DESCRIPTIONS

MODULE SGD

	SGD
	`(SET OF INTEGER4 shape, REAL trainToLoss=.05, UNSIGNED numEpochs=0, UNSIGNED miniBatchSize=1000, REAL lr=.1, UNSIGNED negSamp=10, UNSIGNED noProgressEpochs = 5)`

Synchronous Gradient Descent.

Perform distributed training of a neural network using Synchronous Batch Gradient Descent.

Weights are carried as slices, which are managed by the Weights module.

PARAMETER **numEpochs** ||| UNSIGNED8 — The maximum number of times to loop through the full set of training data. Each pass through the training data is known as an Epoch. Default 0 (recommended) means no fixed number of epochs. Will use trainToLoss and noProgressEpochs to terminate.

PARAMETER **miniBatchSize** ||| UNSIGNED8 — The number of training samples to train on before re-synchronizing the weights across nodes. The larger the number, the faster it will run, but a number too large may make convergence difficult. 1000 seems to be a good number.

PARAMETER **noProgressEpochs** ||| UNSIGNED8 — The number of epochs without progress before early termination of training.

PARAMETER **shape** ||| SET (INTEGER4) — The shape of the neural network expressed as a set of dimensions. For example, [100, 10, 100] means that there are 100 weights in the input layer, 10 in the hidden layer and 100 in the output layer.

PARAMETER **trainToLoss** ||| REAL8 — Stop training when the average Loss level is reached. This is the recommended method of controlling training duration. The Default .05 (i.e. 5 percent loss) seems to give good results.

PARAMETER **negSamp** ||| UNSIGNED8 — The number of negative samples to use for each main word during training.

PARAMETER **lr** ||| REAL8 — The maximum learning rate to use. .1 seems to be a good default level.

Children

1. Train_Dupl : Train the network and return weights duplicated on each node
2. Train : Train the network and return a set of weights distributed among the HPCC nodes by SliceId

---

FUNCTION Train_Dupl

SGD \

DATASET(SliceExt)	**Train_Dupl**
(DATASET(trainingDat) trainData)	

Train the network and return weights duplicated on each node.

Returns weights duplicated on all nodes using the SliceExt format. If you want a single set of weight slices rather than a replicated set of extended slices, use Train(. . .) below.

PARAMETER **traindata** ||| TABLE (TrainingDat) — No Doc

RETURN **TABLE ({ UNSIGNED2 nodeId , UNSIGNED2 sliceId , REAL4 loss , REAL4 minLoss , UNSIGNED4 minEpoch , UNSIGNED4 maxNoProg , UNSIGNED8 batchPos , SET (REAL8) weights })** —

FUNCTION Train

SGD \

DATASET(Slice)	**Train**
(DATASET(trainingDat) trainData)	

Train the network and return a set of weights distributed among the HPCC nodes by SliceId. If you want a set of weights that are DUPLICATED on each node (i.e. for LOCAL processing on each node), use Train_Dupl above.

PARAMETER **traindata** ||| TABLE (TrainingDat) — No Doc

RETURN **TABLE ({ UNSIGNED2 sliceId , SET (REAL8) weights })** —

Internal/

svTrainNN

Go Up

IMPORTS

Types |

DESCRIPTIONS

EMBED svTrainNN

STREAMED DATASET(SliceExt)	**svTrainNN**
(STREAMED DATASET(SliceExt) wts, STREAMED DATASET(trainingDat) train, UNSIGNED4 slicesize, UNSIGNED4 nWeights, UNSIGNED4 numwords, UNSIGNED4 dim, UNSIGNED4 mbsize, REAL lr, UNSIGNED4 negsamp)	

Neural Network Training for SentenceVectors.

Train specialized SentenceVector neural network given a batch of training data. Takes in weights as a set of weights slices (SliceExt), and returns a set of weight adjustments, also formatted as slices.

PARAMETER **nWeights** ||| UNSIGNED4 — The total number of weights across all slices.

PARAMETER **dim** ||| UNSIGNED4 — The dimensionality of the vectors being trained

PARAMETER **negsamp** ||| UNSIGNED4 — The number of negative samples to choose for each main word.

PARAMETER **numwords** ||| UNSIGNED4 — The number of words in the vocabulary.

PARAMETER **train** ||| TABLE (TrainingDat) — The batch of training data formatted as a main word and set of context words.

PARAMETER **mbsize** ||| UNSIGNED4 — The number of training records in the mini-batch.

PARAMETER **wts** ||| TABLE (SliceExt) — The weights slices.

PARAMETER **lr** ||| REAL8 — The learning rate to use for this batch.

PARAMETER **slicesize** ||| UNSIGNED4 — The maximum number of weights in a slice.

RETURN **TABLE (SliceExt)** — weight updates as DATASET(SliceExt). Note that these are additive changes to the weights, not the final weight values.

---

svUtils

Go Up

IMPORTS

Types |

DESCRIPTIONS

MODULE svUtils

	svUtils

Various utility functions used by TextVectors

Children

1. normalizeVector : Normalize a vector by dividing by the its length to create a unit vector
2. calcSentVector : Calculate a Sentence Vector by taking the average of the word vectors for all words in the sentence
3. cosineSim : Cosine similarity a and b are unit vectors
4. isNumeric : Returns TRUE if a string represents a number (integer)
5. numDistance : Calculates the numeric distance between two numeric strings as ABS(n1 - n2)
6. addVecs : Implements vec1 + (vec2 * multiplier) Allows (potentially) scaled addition of vectors as well as subtraction (using a negative multiplier

---

EMBED normalizeVector

svUtils \

t_Vector	**normalizeVector**
(t_Vector vec)	

Normalize a vector by dividing by the its length to create a unit vector

PARAMETER **vec** ||| SET (REAL8) — No Doc

RETURN **SET (REAL8)** —

EMBED calcSentVector

svUtils \

t_Vector	**calcSentVector**
(t_Vector wordvecs, UNSIGNED2 veclen)	

Calculate a Sentence Vector by taking the average of the word vectors for all words in the sentence.

PARAMETER **veclen** ||| UNSIGNED2 — The length of each word vector and the resulting sentence vector.

PARAMETER **wordvecs** ||| SET (REAL8) — A concatenated set of vectors for all the words in the sentence.

RETURN **SET (REAL8)** —

EMBED cosineSim

svUtils \

REAL8	**cosineSim**
(t_Vector a_in, t_Vector b_in, UNSIGNED4 veclen)	

Cosine similarity a and b are unit vectors. Theta is the angle between vectors. Cosine similarity is Cos(theta). Cos(theta) = (a . b) / (L2Norm(a) * L2Norm(b)) Note: a . b = L2Norm(a) * L2Norm(b) * Cos(theta) Since we assume the inputs to be unit vectors, the norms will be 1. We therefore simplify the calculation to a . b.

PARAMETER **b_in** ||| SET (REAL8) — No Doc

PARAMETER **a_in** ||| SET (REAL8) — No Doc

PARAMETER **veclen** ||| UNSIGNED4 — No Doc

RETURN **REAL8** —

EMBED isNumeric

svUtils \

BOOLEAN	**isNumeric**
(STRING instr)	

Returns TRUE if a string represents a number (integer). Otherwise FALSE.

PARAMETER **instr** ||| STRING — No Doc

RETURN **BOOLEAN** —

EMBED numDistance

svUtils \

UNSIGNED4	**numDistance**
(VARSTRING str1, VARSTRING str2)	

Calculates the numeric distance between two numeric strings as ABS(n1 - n2).

PARAMETER **str1** ||| VARSTRING — No Doc

PARAMETER **str2** ||| VARSTRING — No Doc

RETURN **UNSIGNED4** —

EMBED addVecs

svUtils \

t_Vector	addVecs
(t_Vector vec1, t_Vector vec2, UNSIGNED4 multiplier = 1)	

Implements vec1 + (vec2 * multiplier) Allows (potentially) scaled addition of vectors as well as subtraction (using a negative multiplier.

PARAMETER **vec2** ||| SET (REAL8) — No Doc

PARAMETER **vec1** ||| SET (REAL8) — No Doc

PARAMETER **multiplier** ||| UNSIGNED4 — No Doc

RETURN **SET (REAL8)** —

Internal/

Weights

Go Up

IMPORTS

Types | std.system.Thorlib |

DESCRIPTIONS

MODULE weights

	weights
(SET OF INTEGER4 shape)	

Module to perform calculations to manage the weights, and their storage as slices.

Weights are stored in fixed size slices for ease of distribution and management. Currently only supports 3 layer Neural Network weights as used in word vectorization. Will need to be extended to handle a general Neural Network shape.

PARAMETER **shape** ||| SET (INTEGER4) — The number of neurons in each layer of the Neural Network. For example, [100, 10, 200] describes a neural network with 100 neurons in the input layer, 10 in the hidden layer, and 200 in the output layer.

Children

1. nWeights : The total number of weights in the network
2. toFlatIndex : Convert the compound index: (layer, j, i) to a contiguous flat index into a set of weights

3. fromFlatIndex : Convert a flat index into a list of weights into a compound index into the weights of the neural network: (layer, j, i)
4. slicesPerNode : The number of slices needed for each node
5. nSlices : The total number of slices used to hold the weights
6. sliceSize : The number of weights in each slice
7. nWeightSlots : The number of slots to hold weights across all slices
8. initWeights : Return an initial set of weight slices with weights set to random values
9. distributeAllSlices : Copy weights to all nodes and assign the node id to the copy on each node
10. toSliceExt : Make Extended Weights
11. fromSliceExt : Take a set of Extended Weight slices (i.e
12. slices2Linear : Convert a dataset of replicated slices (SliceExt) into a single SliceExt replicated on each node and containing one linear array (i.e
13. compressOne : Compress a set of weights (assumed to be sparse) by converting to a sparse representation [. . .] packed into a DATA field
14. decompressOne : Decompress a set of compressed weights in sparse format (i.e
15. compressWeights : Compress a set of extended slices (e.g
16. decompressWeights : Decompress a set of compressed slices into the native extended slice format

---

ATTRIBUTE nWeights

weights \

	nWeights

The total number of weights in the network

RETURN **INTEGER8** —

FUNCTION toFlatIndex

weights \

UNSIGNED4	**toFlatIndex**
(UNSIGNED2 l, UNSIGNED4 j, UNSIGNED4 i)	

Convert the compound index: (layer, j, i) to a contiguous flat index into a set of weights.

PARAMETER **i** ||| UNSIGNED4 — No Doc

PARAMETER **l** ||| UNSIGNED2 — No Doc

PARAMETER **j** ||| UNSIGNED4 — No Doc

RETURN **UNSIGNED4** —

FUNCTION fromFlatIndex

weights \

wIndex	**fromFlatIndex**
(UNSIGNED4 indx)	

Convert a flat index into a list of weights into a compound index into the weights of the neural network: (layer, j, i).

PARAMETER **indx** ||| UNSIGNED4 — No Doc

RETURN **ROW (wIndex)** —

ATTRIBUTE slicesPerNode

weights \

	slicesPerNode

The number of slices needed for each node

RETURN **INTEGER8** —

ATTRIBUTE nSlices

weights \

	nSlices

The total number of slices used to hold the weights.

RETURN **INTEGER8** —

ATTRIBUTE sliceSize

weights \

	sliceSize

The number of weights in each slice.

RETURN **INTEGER8** —

ATTRIBUTE nWeightSlots

weights \

	nWeightSlots

The number of slots to hold weights across all slices. This may be different from nWeights because nWeights does not always divide exactly into nSlices.

RETURN **INTEGER8** —

ATTRIBUTE initWeights

weights \

DATASET(slice)	initWeights

Return an initial set of weight slices with weights set to random values.

RETURN **TABLE (Slice)** —

FUNCTION distributeAllSlices

weights \

DATASET(SliceExt)	distributeAllSlices
(DATASET(SliceExt) slices)	

Copy weights to all nodes and assign the node id to the copy on each node.

If running on 7.2 or greater, use the DISTRIBUTE(.., ALL) facility. Otherwise, use NORMALIZE to make copies of each and assign nodeId, then DISTRIBUTE by nodeId.

PARAMETER **slices** ||| TABLE (SliceExt) — No Doc

RETURN **TABLE ({ UNSIGNED2 nodeId , UNSIGNED2 sliceId , REAL4 loss , REAL4 minLoss , UNSIGNED4 minEpoch , UNSIGNED4 maxNoProg , UNSIGNED8 batchPos , SET (REAL8) weights })** —

FUNCTION toSliceExt

weights \

DATASET(SliceExt)	**toSliceExt**
(DATASET(Slice) weights)	

Make Extended Weights. Return a dataset of weight slices that have been replicated to all nodes and converted to SliceExt record type that includes a node id.

PARAMETER **weights** ||| TABLE (Slice) — No Doc

RETURN **TABLE ({ UNSIGNED2 nodeId , UNSIGNED2 sliceId , REAL4 loss , REAL4 minLoss , UNSIGNED4 minEpoch , UNSIGNED4 maxNoProg , UNSIGNED8 batchPos , SET (REAL8) weights })** —

FUNCTION fromSliceExt

weights \

DATASET(Slice)	**fromSliceExt**
(DATASET(SliceExt) extWeights)	

Take a set of Extended Weight slices (i.e. replicated to all nodes) and return a dataset of Weight slices that are distributed by sliceId. The duplicated copies are filtered out except on the node that owns each slice.

PARAMETER **extweights** ||| TABLE (SliceExt) — No Doc

RETURN **TABLE ({ UNSIGNED2 sliceId , SET (REAL8) weights })** —

FUNCTION slices2Linear

weights \

SliceExt	slices2Linear
(DATASET(SliceExt) slices)	

Convert a dataset of replicated slices (SliceExt) into a single SliceExt replicated on each node and containing one linear array (i.e. SET) of weights.

PARAMETER **slices** ||| TABLE (SliceExt) — No Doc

RETURN **ROW ({ UNSIGNED2 nodeId , UNSIGNED2 sliceId , REAL4 loss , REAL4 minLoss , UNSIGNED4 minEpoch , UNSIGNED4 maxNoProg , UNSIGNED8 batchPos , SET (REAL8) weights })** —

EMBED compressOne

weights \

DATA	compressOne
(t_Vector wts, UNSIGNED4 slicesize)	

Compress a set of weights (assumed to be sparse) by converting to a sparse representation [...] packed into a DATA field.

PARAMETER **wts** ||| SET (REAL8) — No Doc

PARAMETER **slicesize** ||| UNSIGNED4 — No Doc

RETURN **DATA** —

EMBED decompressOne

weights \

t_Vector	decompressOne
(DATA cwts, UNSIGNED4 slicesize)	

Decompress a set of compressed weights in sparse format (i.e. [...] into a dense set of weights.

PARAMETER **cwts** ||| DATA — No Doc

PARAMETER **slicesize** ||| UNSIGNED4 — No Doc

RETURN **SET (REAL8)** —

FUNCTION compressWeights

weights \

DATASET(CSlice)	**compressWeights**
(DATASET(SliceExt) slices)	

Compress a set of extended slices (e.g. SliceExt) into CSlice format.

PARAMETER **slices** ||| TABLE (SliceExt) — No Doc

RETURN **TABLE ({ UNSIGNED2 nodeId , UNSIGNED2 sliceId , REAL4 loss , REAL4 minLoss , UNSIGNED4 minEpoch , UNSIGNED4 maxNoProg , UNSIGNED8 batchPos , DATA cweights })** —

FUNCTION decompressWeights

weights \

DATASET(SliceExt)	**decompressWeights**
(DATASET(CSlice) cslices)	

Decompress a set of compressed slices into the native extended slice format.

PARAMETER **cslices** ||| TABLE (CSlice) — No Doc

RETURN **TABLE ({ UNSIGNED2 nodeId , UNSIGNED2 sliceId , REAL4 loss , REAL4 minLoss , UNSIGNED4 minEpoch , UNSIGNED4 maxNoProg , UNSIGNED8 batchPos , SET (REAL8) weights }) —**
