CENTRO NACIONAL DE INTELIGENCIA ARTIFICIAL

dcc CIENCIAS DE LA COMPUTACIÓN UNIVERSIDAD DE CHILE

RiverText: A Framework for Training and Evaluating Incremental Word Embeddings from Text Data Streams

Felipe Bravo-Marquez

Supervisor

Gabriel Iturra-Bocaz

Thesis student

Artificial Intelligence

The incursion of artificial intelligence(AI) into our daily life is already a fact.

Machine translation

Chatbots

Recommender Systems

...

Artificial Intelligence

The incursion of artificial intelligence(AI) into our daily life is already a fact.

Machine translation

Chabots

Recommender Systems

...

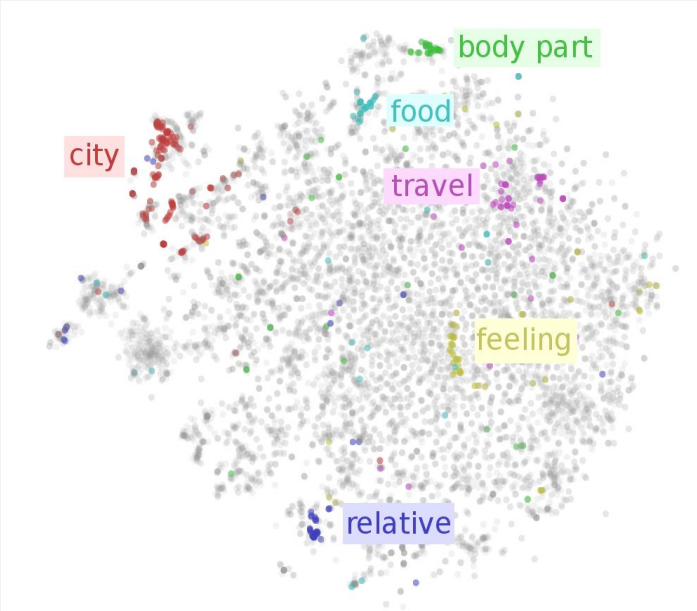
Natural Language Processing (NLP)

How can we computationally work with language?

Word Embeddings (WE)

- A word vector is an attempt to **mathematically represent the meaning** of a word
- Models that **capture the semantics and syntactic** properties of the words.
- They classify into two categories:
  - **Count-based methods** (e.g., Word-Context matrix).
  - **Distributed methods** (e.g., Skipgram, CBOW).
- Based on Distributional Hypothesis:

"Words that occur in similar contexts tend to have similar meanings".



https://colah.github.io/posts/2015-01-Visualizing-Representations/

Streaming in Word Embeddings

However, the **static nature** of standard Word Embeddings algorithms **prevents them from incorporating new words**, such as hashtags or new brand names, and **adapting to semantic changes in existing words**.

Streaming in Word Embeddings

Researchers have explored various techniques to represent word embeddings and incorporate **incremental learning** approaches that are trained from **data streams** [1,2,3,4].

Incremental learning allows the models to **adapt and learn from new data over time**.

Avoiding the **inefficient retraining of the whole** model when **new data appears**.

Research Problems

Although all the Incremental Word Embeddings models have similar **objectives** and share **common patterns.**

There are two major obstacles that **hinder a fair comparison** between the systems under consideration.

1. Firstly, the systems are s**tored in separate repositories** and l**ack a uniform standardization** process.
2. Secondly, **there is no established approach for evaluating** the incremental WE performance in streaming scenarios.

Research Problems

While several proposed models for incremental WE exist in the literature, they currently exist in isolation and lack a standardized approach for a clean comparison.

There is a lack of established evaluation processes for assessing the performance of incremental WE.

Research Problems	Contribution
While several proposed models for incremental WE exist in the literature, they currently exist in isolation and lack a standardized approach for a clean comparison.	A formalization and standardization of the incremental WE models into a unified framework that adheres to the fundamental principles of incremental learning.
There is a lack of established evaluation processes for assessing the performance of incremental WE.	We propose a Periodic Evaluation approach that utilizes intrinsic NLP tasks. Additionally, we conducted a benchmark study by varying the hyperparameter settings using our proposed evaluation scheme.

Research Problems

Contribution

While several proposed models for incremental WE exist in the literature, they currently exist in isolation and lack a standardized approach for a clean comparison.

→ A formalization and standardization of the incremental WE models into a unified framework that adheres to the fundamental principles of incremental learning.

There is a lack of established evaluation processes for assessing the performance of incremental WE.

→ We propose a Periodic Evaluation approach that utilizes intrinsic NLP tasks. Additionally, we conducted a benchmark study by varying the hyperparameter settings using our proposed evaluation scheme.

RiverText: A Framework for Training and Evaluating Incremental Word Embeddings from Text Data Streams

Publications

1. G. Iturra-Bocaz and F. Bravo-Marquez RiverText: A Python Library for Training and Evaluating Incremental Word Embeddings from Text Data Stream). In Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2023), Taipei, Taiwan. Association for Computational Linguistics.

2. A Python Library available for the research community with the models implemented at https://dccuchile.github.io/rivertext/.

Outline

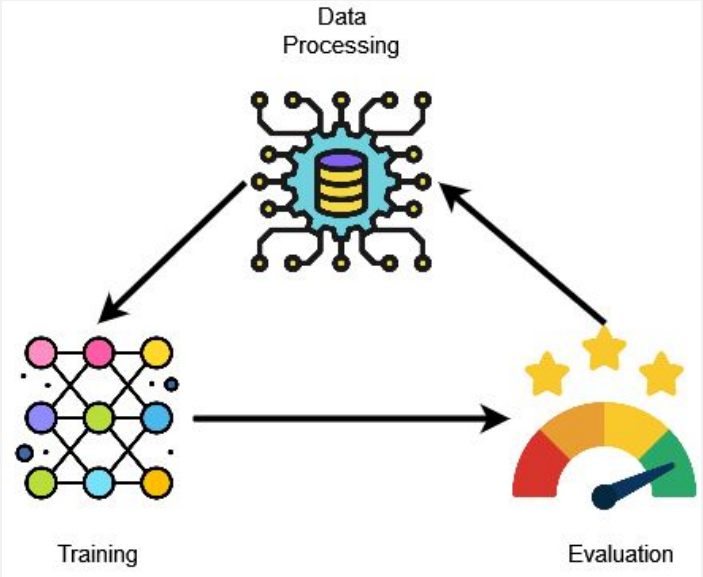
- Fundamental concepts and some details of RiverText
- Benchmark Study: Ranking Different Incremental Word Embedding Models Against Different Intrinsic NLP Tasks.
- Conclusions and Future Work

Incremental Learning Approaches

There are two main types of incremental learning approaches:

1. Instance learning, which involves training **each instance immediately** upon its arrival and **subsequently discarding it**.

2. Incremental batch learning involves **gathering a small batch of instances** before training on them.



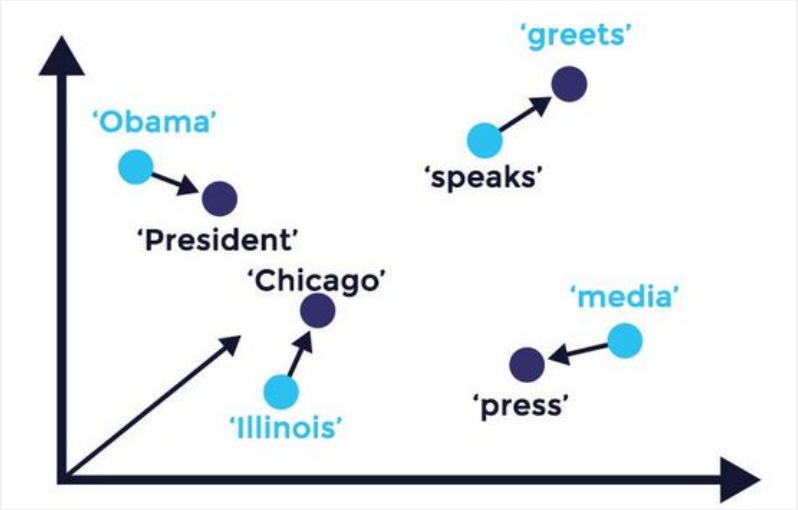
Intrinsic NLP tasks

WE intrinsic evaluation is a family of evaluation techniques for measuring the syntactic and semantic properties captured by these vectors that include three types of tasks:

- Word Similarities.
- Analogies.
- Categorization.

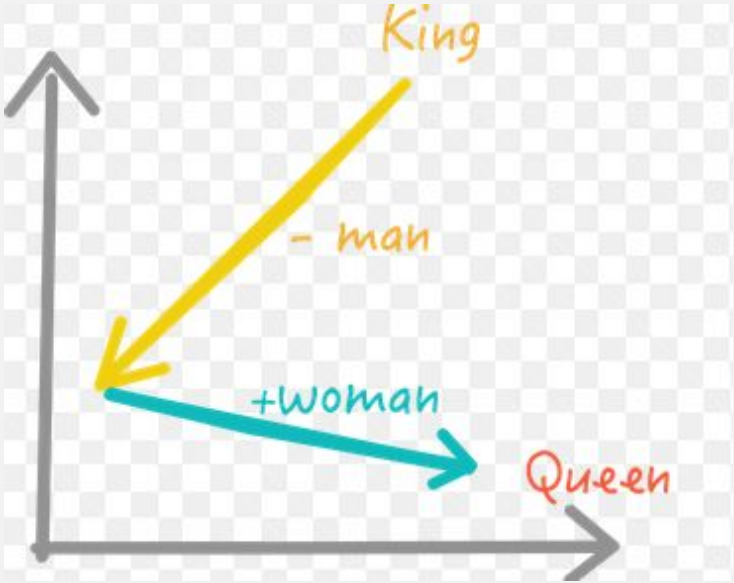
Word Similarities

- Measure whether the similarity between two-word vectors correlates with a human judgment of relatedness.
- It quantifies measuring the Spearman correlation between two vector embedding vectors.



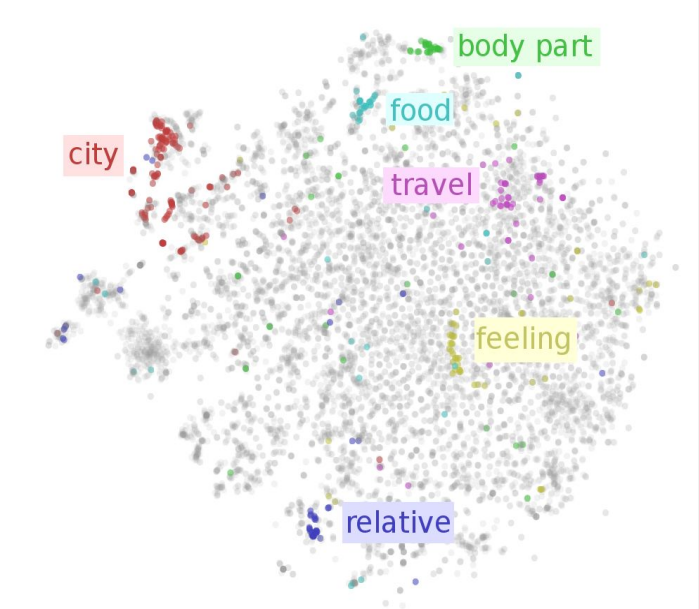
Analogies

- It determines by the relations in the form of "a is to b as c is to d," which can be obtained from arithmetic operations on the vectors.
- Accuracy is used to count the number of correctly obtained words from an analogy equation, comparing the dataset of analogy words.



Categorization

- It is determined by groups of words that are aligned with predefined categories, such as animals
- Purity clustering is used to count the total number of correctly classified words, comparing the categories obtained from the word embeddings.



https://colah.github.io/posts/2015-01-Visualizing-Representations/

Periodic Evaluation

The proposed method for evaluating our incremental WE performance is called Periodic Evaluation. This method **applies a series of evaluations** to the entire model, using a **test dataset associated with intrinsic NLP tasks** after a **fixed number, *p*, of instances**, have been processed and trained. The algorithm takes as input the following arguments:

- The parameter p represents the number of instances between the evaluation series.
- The incremental WE model, referred to as M, is to be evaluated.
- The input text data stream, referred to as TS, used to train the incremental WE model.
- A test dataset, GR, associated with intrinsic NLP tasks.

Periodic Evaluation

Algorithm 1: Periodic Evaluation Algorithm. The evaluator function takes the words and their mapped vectors, and an intrinsic dataset.

Input: Stream ST, Incremental WE model, Intrinsic Dataset GR, int p

```
1 c = 0
2 while batch in ST do
      // train the model
3     learn_many(model, batch)
      // evaluate the model during certain periods
4     if c ≠ 0 ∧ c mod p then
5         result = evaluator(model.wv, GR)
          // the result is stored in a JSON file
6         save(result)
7     c += length(batch)
```

Incremental Word Embedding

The incremental WE model is a departure from the traditional static WE approach, as it follows an incremental methodology that assumes certain requirements to function effectively:

- Be able to process one instance (or mini-batch) at a time, and inspect it (at most) once.
- Be able to process data with limited resources (time and memory).
- Be able to generate a prediction or transformation at any time.
- Be able to adapt to temporal changes

Incremental Word Embedding

The process in which these vectors are trained in our software is as follows:

- **Connect** to a **continuous source** of a text data stream (e.g., Twitter).
- Tokenize the text and traverse its words.
- If a **new word** is found, it is **added to the vocabulary** and a **new vector** is assigned to it.
- If the **word is known**, its corresponding **vector is updated** according to its context (i.e., its surrounding words).
- **At any time** during training, it is possible to **get the vector** associated with a vocabulary word.

For this thesis, we implemented an incremental version of the models: **Word Context Matrix (WCM), SkipGram (SG), and Continuous Bag of Words (CBOW)**.

Incremental Word-Context Matrix (IWCM)

The IWCM model represents an augmentation of the word context matrix method. Its main features are:

- It uses a matrix of V x C to store the correlation between the target and its context words.
- The weights are computed using an incremental version of the Positive Pointwise Mutual Information.
- The vectors are **updated by the counts across** the text stream.
- The vectors are **reduced** using the **Incremental PCA algorithm**.

Incremental Word-Context Matrix Algorithm

Algorithm 2: IWCM model method.

Input: Stream ST, window size W, vocab size V, context size C

Output: Matrix Mat V x C

```
d = 0
while batch in ST do
    for tweet in batch do
        tokens = tokenize(tweet)
        for token in tokens do
            if token not in vocab then
                addToVocab(vocab, token)
            d += 1
            contexts = getContexts(token, tokens, W)
            updateDictCounter(token, contexts)
            for cont in contexts do
                Mat(token, cont) = max(0, log((count(token,cont)·d) / (count(token)·count(cont))))
        // reduce the embedding dimension by incremental PCA
        reduceEmbDimByIPCA(tokens)
```

Incremental Word2Vec

The incremental Word2Vec architecture comprises two ISG and ICBOW models based on the static version. The ISG model predicts the context words for a given target word, and the ICBOW model aims to predict the target word using its context words. Its main features are:

- The training process is accelerated with the **Incremental Negative Sampling** developed by Kaji and Hobayashi [1], called **Adaptive Unigram Algorithm**.
- The neural network structure has been implemented using PyTorch as backend.

Adaptive Unigram Table Algorithm

Algorithm 3: Adaptive Unigram Table.

Input: Array word_indexes, Array T, int size_T, Array Freqs, float α

Output: Array T

```
1  z = 0
2  for index in word_indexes do
3      Freqs[index] += 1
4      F = Freqs[index] - (Freqs[index] - 1)^α
5      z += F
6      if |T| < size_T then
7          add F copies of index to T
8      else
9          for j = 1, ..., (size_T·F)/z do
10             T[j] = index with probability F/z
```

Adaptive Unigram Table Algorithm

Algorithm 3: Adaptive Unigram Table.

Input: Array word_indexes, Array T, int size_T, Array Freqs, float α

Output: Array T

```
1  z = 0
2  for index in word_indexes do
3      Freqs[index] += 1
4      F = Freqs[index] - (Freqs[index] - 1)^α
5      z += F
6      if |T| < size_T then
7          add F copies of index to T
8      else
9          for j = 1, ..., size_T·F / z do
10             T[j] = index with probability F / z
```

Adaptive Unigram Table Algorithm

Algorithm 3: Adaptive Unigram Table.

Input: Array word_indexes, Array T, int size_T, Array Freqs, float α

Output: Array T

```
1  z = 0
2  for index in word_indexes do
3      Freqs[index] += 1
4      F = Freqs[index] - (Freqs[index] - 1)^α
5      z += F
6      if |T| < size_T then
7          add F copies of index to T
8      else
9          for j = 1, ..., (size_T·F)/z do
10             T[j] = index with probability F/z
```

Incremental Word2Vec Algorithm

Algorithm 4: Incremental Word2Vec method

Input: Stream ST, Vocab size V, Unigram Table Size T,int num_ns

```
vocab = Vocab(V)
ut = UnigramTable(T)
while batch in ST do
    for tweet in batch do
        tokens = tokenize(tweet)
        for w in tokens do
            if w not in vocab then
                addToVocab(vocab, w)
        updateTokenFreq(w)
        updateUnigramTable(w)
        contexts = getContexts(w, tokens)
        for c in contexts do
            draw num_ns indexes from ut:
              ns_1, ns_2, ..., ns_num_ns
            // convert the word indexes to the neural
               model input
            v_w, v_c, ns_1, ns_2, ..., v_ns_num_ns =
              preprocessing(w, c, ns_1, ns_2, ..., v_ns_num_ns)
            // performs SGD to update the word embedding
            SGD(v_w, v_c, v_ns_1, v_ns_2, ..., ns_num_ns)
```

RiverText: A Framework for Training and Evaluating Incremental Word Embeddings from Text Data Streams

RiverText is a framework for training word embeddings in a streaming context.

It:

- **Formalizes** existing Incremental WE models into a **unified framework**.
- **Develops** common interfaces that follow the principal **incremental learning approaches**.
- Proposes an **evaluation scheme** for the Incremental WE models, using **intrinsic NLP tasks**.
- Extends the **functionality of River**, a Python machine learning library designed for **data streams**.

River

River is a library to build online machine learning models. Such models operate on ***data streams****.*

Online processing *is the act of processing a data stream* ***one element at a time****.*

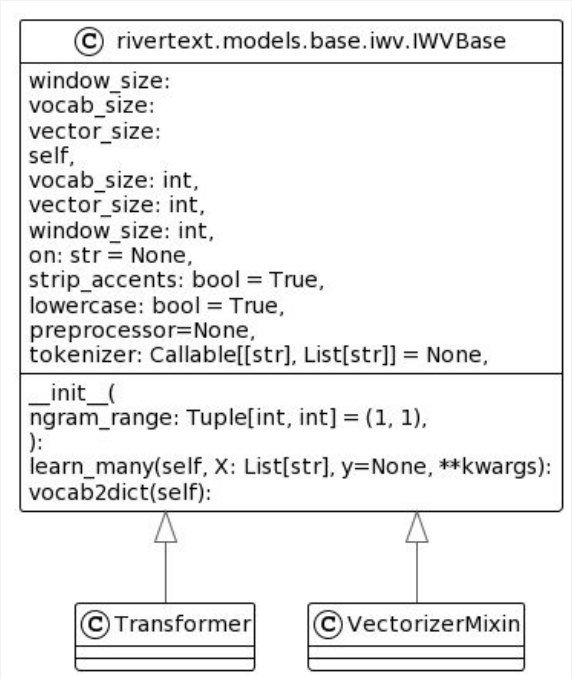
RiverText extends the main ***functionalities*** *of River, adding support for* ***processing and training WE.***

`IWVBase` Class

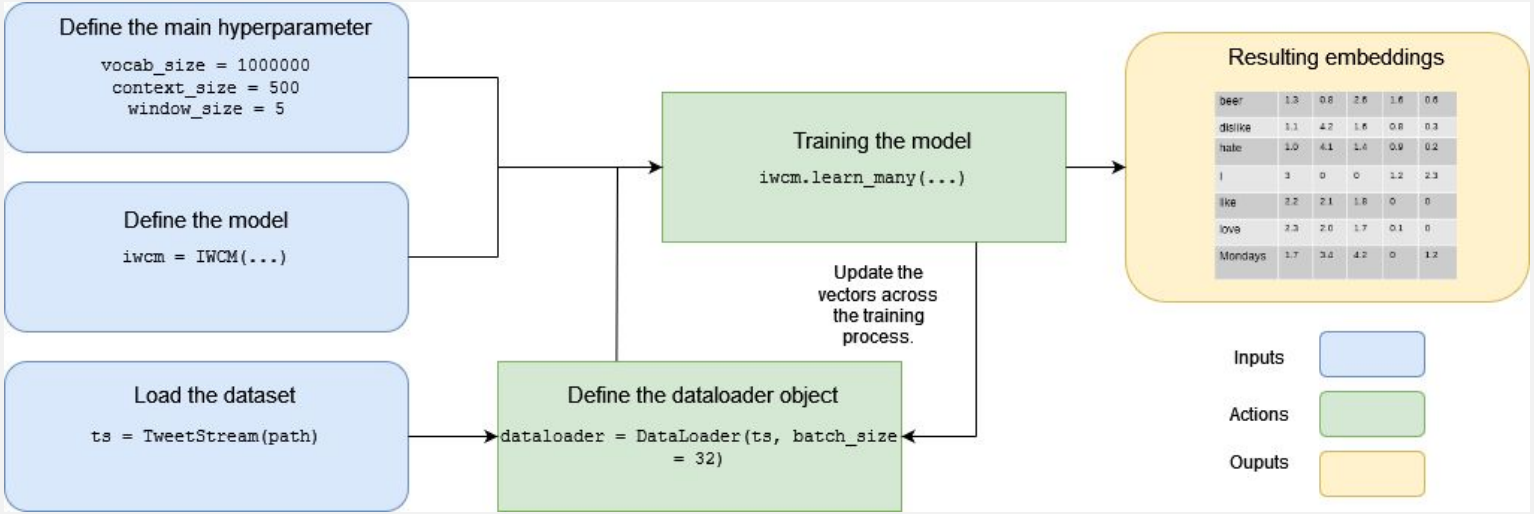
The base class represents a **general interface** for implementing **incremental WE** methods. This class extend:

- The **`Transformer`** class contains the **`learn_one`** and **`learn_many`**.

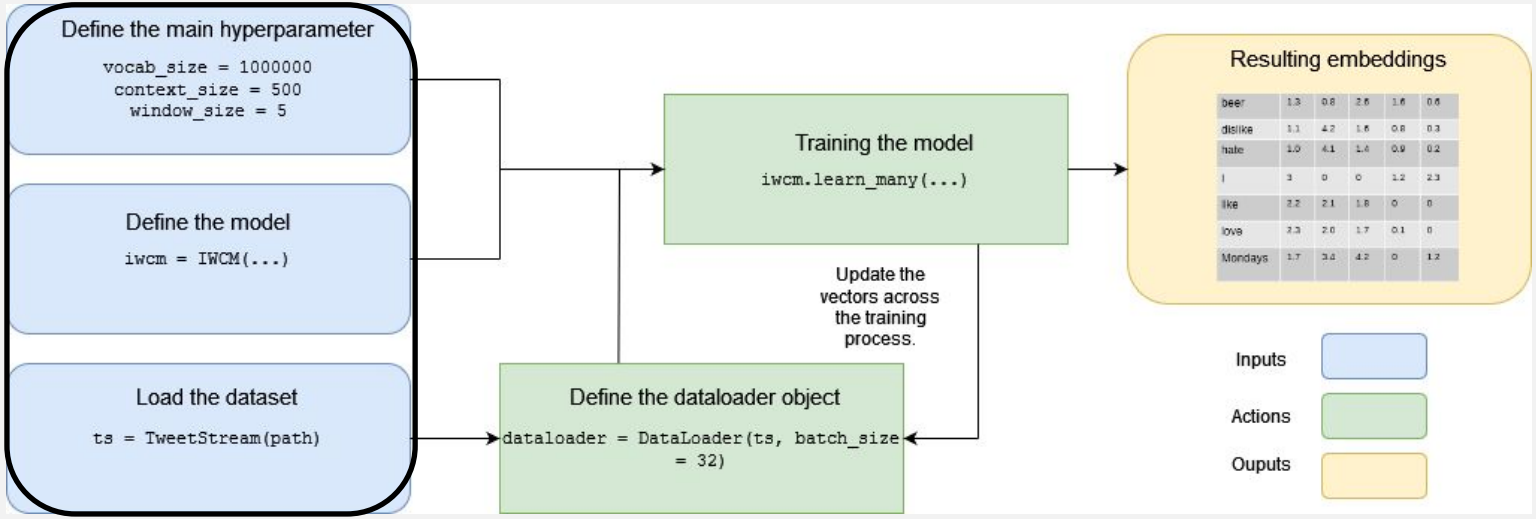
- **`VectorizeMixin`** contains standard functions for **text preprocessing**, such as tokenization.



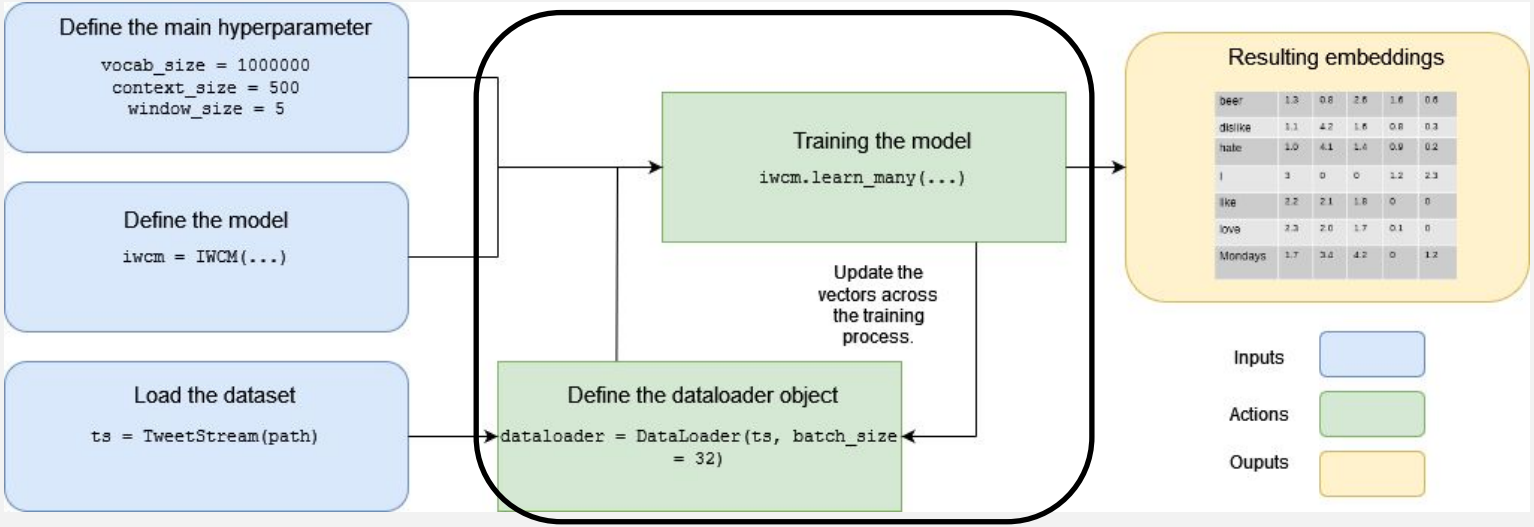
Training flow of the IWCM model



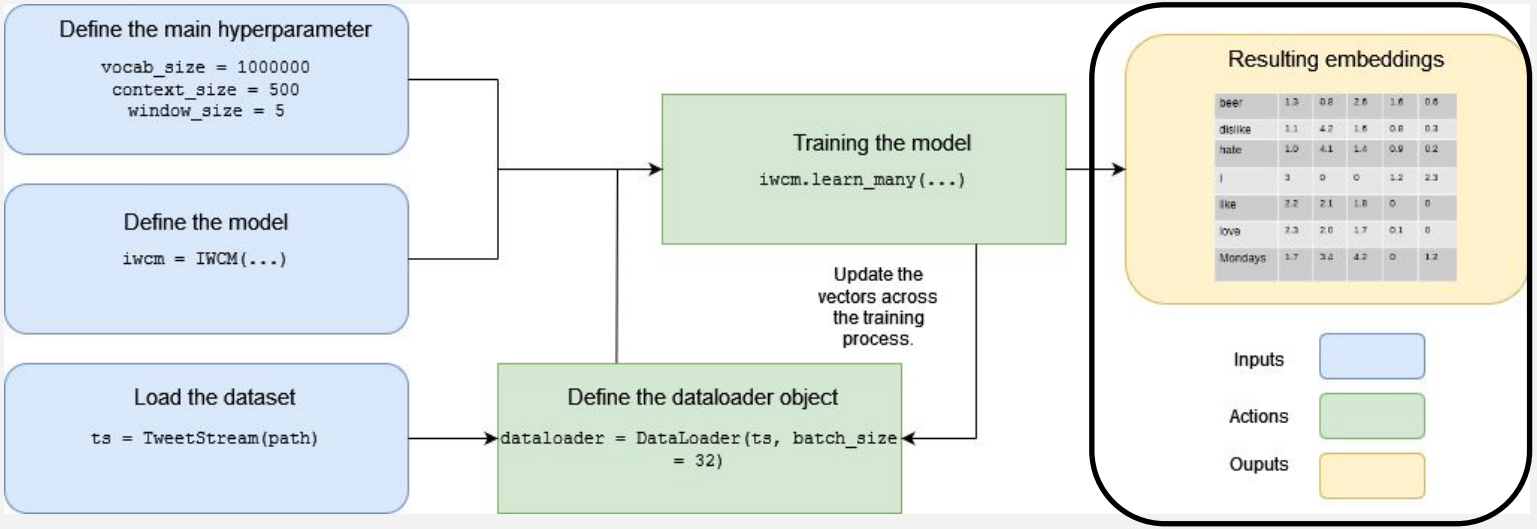
Training flow of the IWCM model



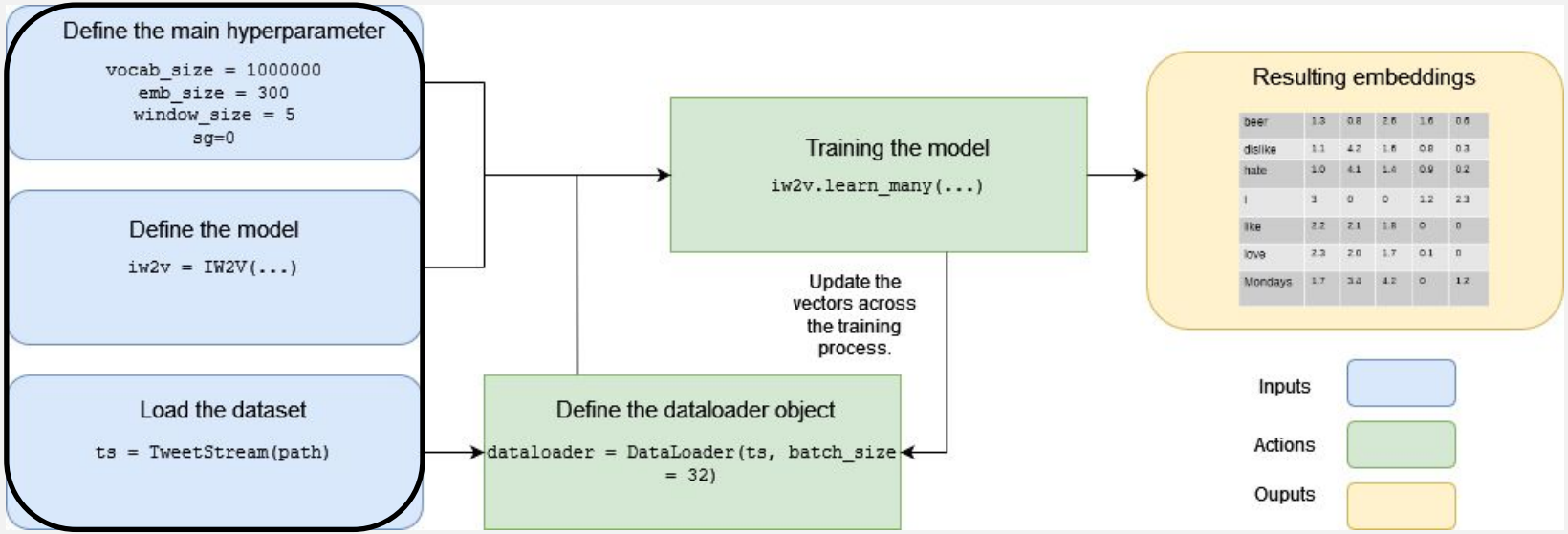
Training flow of the IWCM model



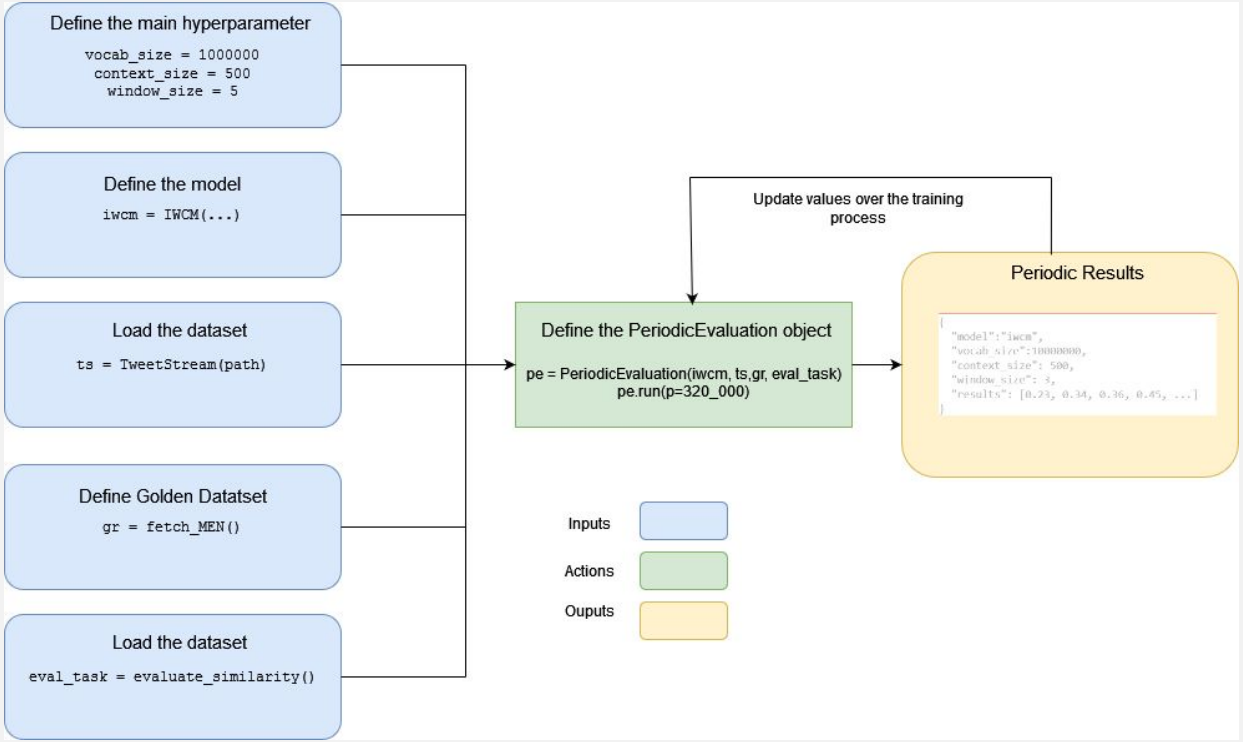
Training flow of the IWCM model



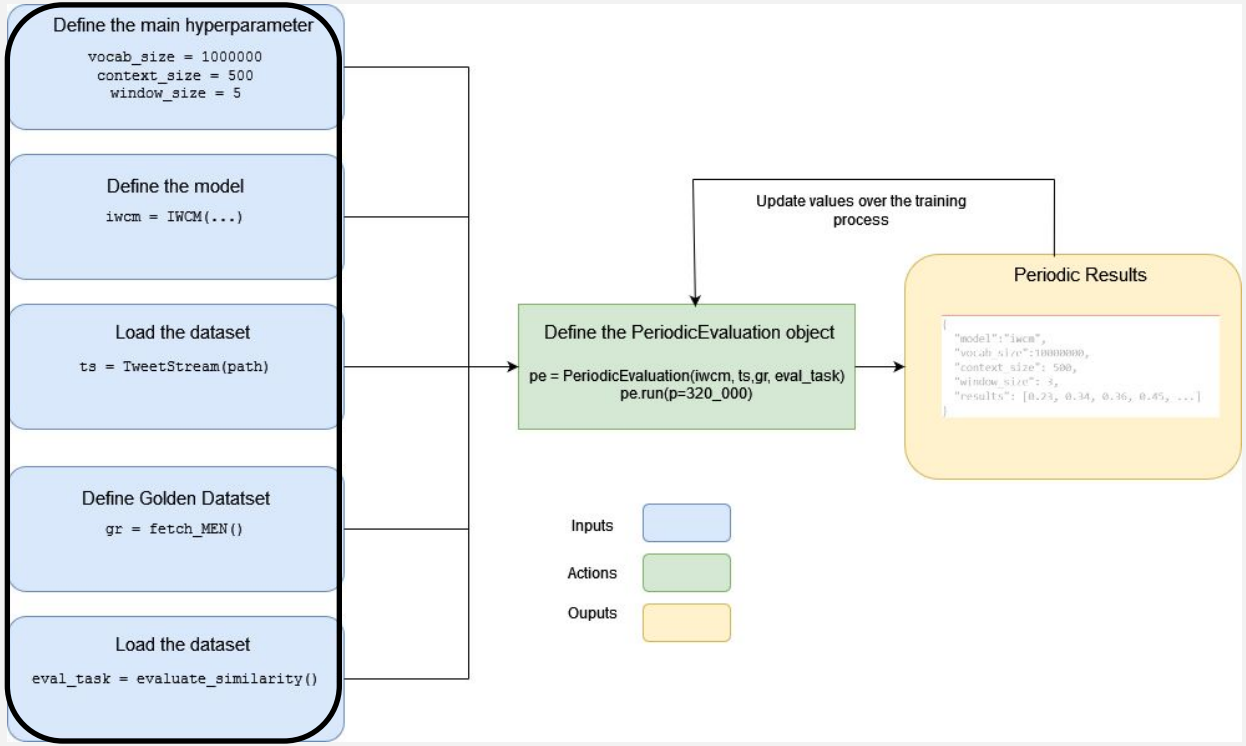
Training flow of the IW2V model



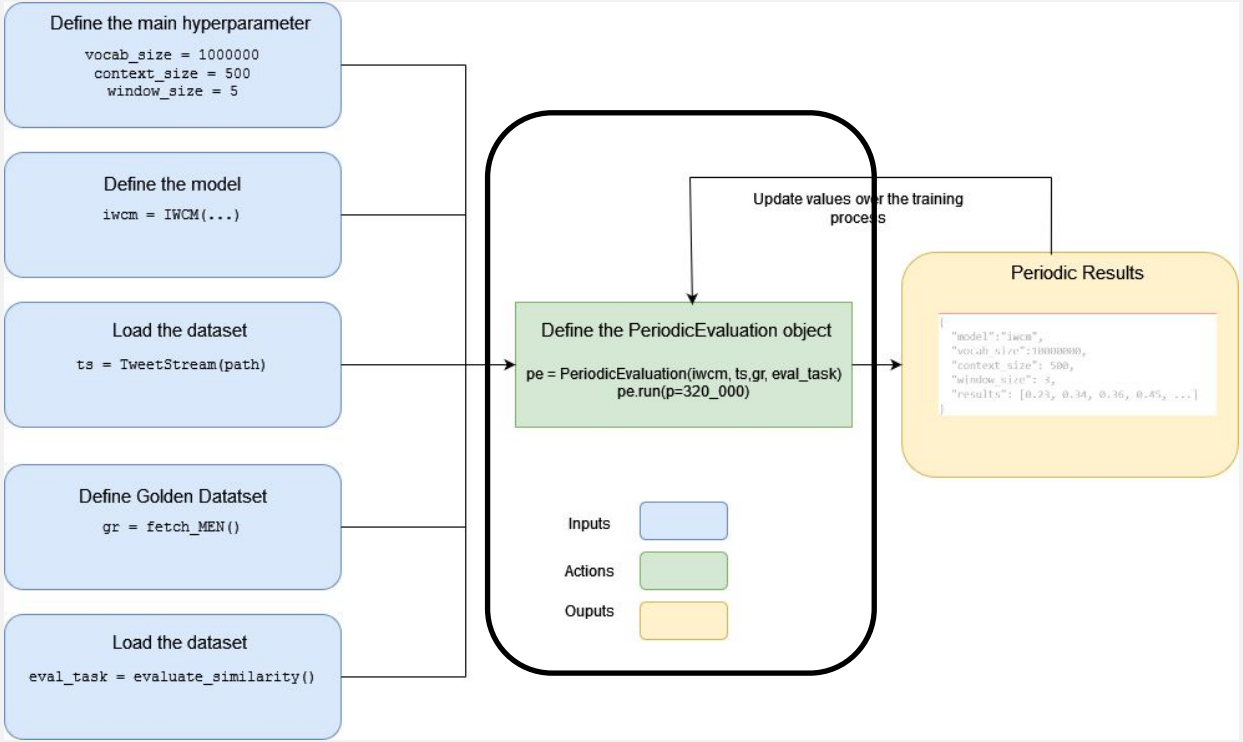
Periodic Evaluation workflow



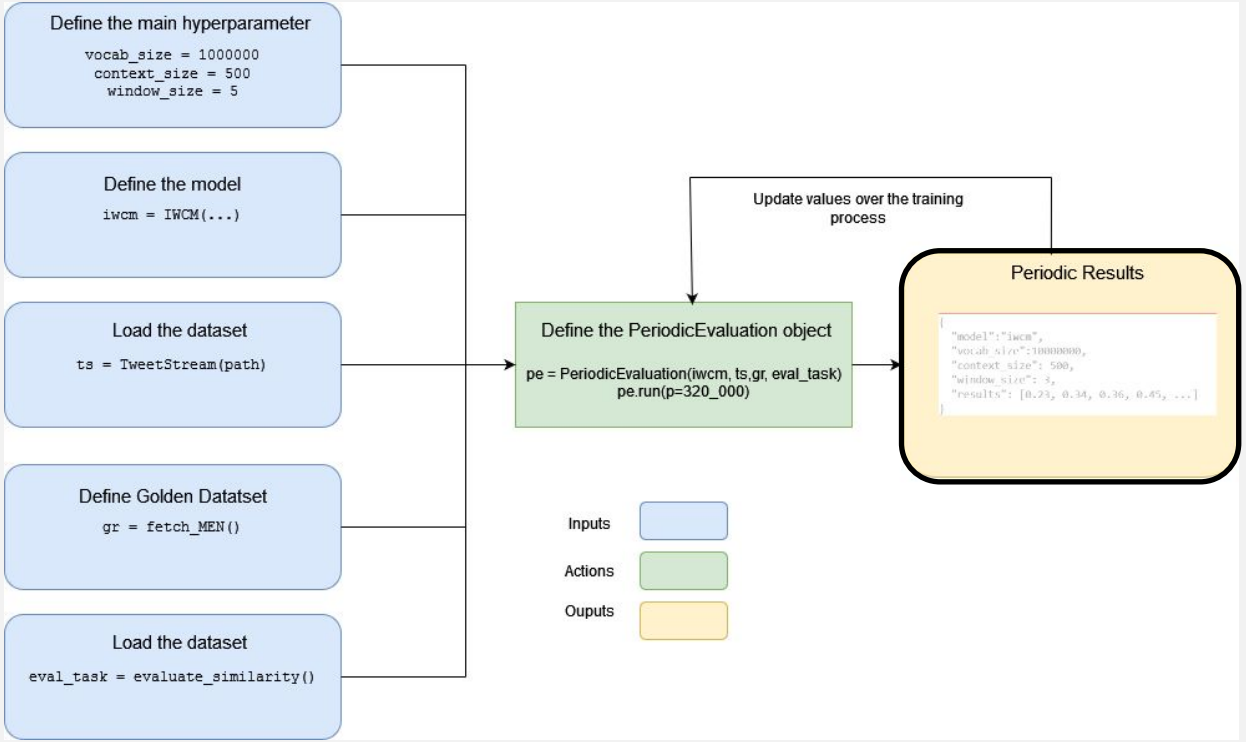
Periodic Evaluation workflow



Periodic Evaluation workflow



Periodic Evaluation workflow



We released RiverText as an **open source library!**

The full documentation can be found in

https://dccuchile.github.io/rivertext/getting_started/

And the repository in:

https://github.com/dccuchile/rivertext

Data

- Our experiment uses a dataset of unlabeled tweets to simulate a text stream of tweets.
- Twitter provides an excellent source of text streams, given its widespread use and real-time updates from its users.
- We draw a set of ten million tweets in English from the Edinburgh corpus.

Experimental Design

- We executed the Periodic Evaluation using a diverse range of datasets and hyperparameter settings.
- The evaluation was conducted on multiple architectural configurations and intrinsic test datasets.
- The hyperparameters under consideration were the size of the embedding, the window size, the context size, and the number of negative samples.
- The results of this evaluation provide valuable insights into the performance of the different architectural configurations and hyperparameter settings.

Experimental Design

Models:

- IWCM
- ISG
- ICBOW

Intrinsic NLP Task:

- MEN dataset (Similarity).
- Mturk (Similarity).
- AP (Categorization)

For the ISG and ICBOW model:

- Emb_size = {100, 200, 300}
- Window_size = {1, 2, 3}
- ns_samples = {6, 8, 10}

For IWCM model:

- Emb_size = {100, 200, 300}
- Window_size = {1, 2, 3}
- Context_size = {500, 750, 1000}

Model	Emb. size	Window size	Num. N. S.	Mean	T1	...	T15	...	T31
ICBOW	**300**	**3**	**6**	**0,5075**	**0,3546**	**...**	**0,5236**	**...**	**0,5194**
ICBOW	300	3	8	0,507	0,3381	...	0,5298	...	0,5275
ICBOW	300	3	10	0,505	0,3366	...	0,5291	...	0,5327
ICBOW	100	3	8	0,4891	0,3499	...	0,5233	...	0,5013
ICBOW	200	3	10	0,4884	0,3433	...	0,4971	...	0,5112
ICBOW	100	3	6	0,4879	0,3128	...	0,4985	...	0,5108
ICBOW	200	3	8	0,4828	0,3562	...	0,5041	...	0,5081
ICBOW	100	3	10	0,4828	0,3288	...	0,5058	...	0,5168
ICBOW	300	2	6	0,4827	0,3312	...	0,4988	...	0,5242
ICBOW	200	3	6	0,4778	0,3359	...	0,4892	...	0,5097
ICBOW	300	2	8	0,4758	0,3474	...	0,4967	...	0,4895
ICBOW	300	2	10	0,4677	0,3446	...	0,4762	...	0,4854
ICBOW	200	2	6	0,4555	0,3067	...	0,4865	...	0,4969
ICBOW	100	2	8	0,4541	0,3102	...	0,4979	...	0,4613
ICBOW	200	2	10	0,4516	0,3264	...	0,4597	...	0,4784
ICBOW	200	2	8	0,4476	0,2761	...	0,436	...	0,4626
ICBOW	100	2	6	0,4475	0,3197	...	0,4547	...	0,4961
ICBOW	100	2	10	0,4402	0,3261	...	0,4465	...	0,4557
ICBOW	300	1	8	0,4226	0,2779	...	0,438	...	0,4406

Results

Table 1: The Overall Ranking of the benchmark results is based on the average of the Periodic Evaluation applied across the text stream. The result tasks are calculated by finding the mean of the evaluation, and the overall mean determined by taking the average of these result tasks. This overall mean then determines the position in the ranking.

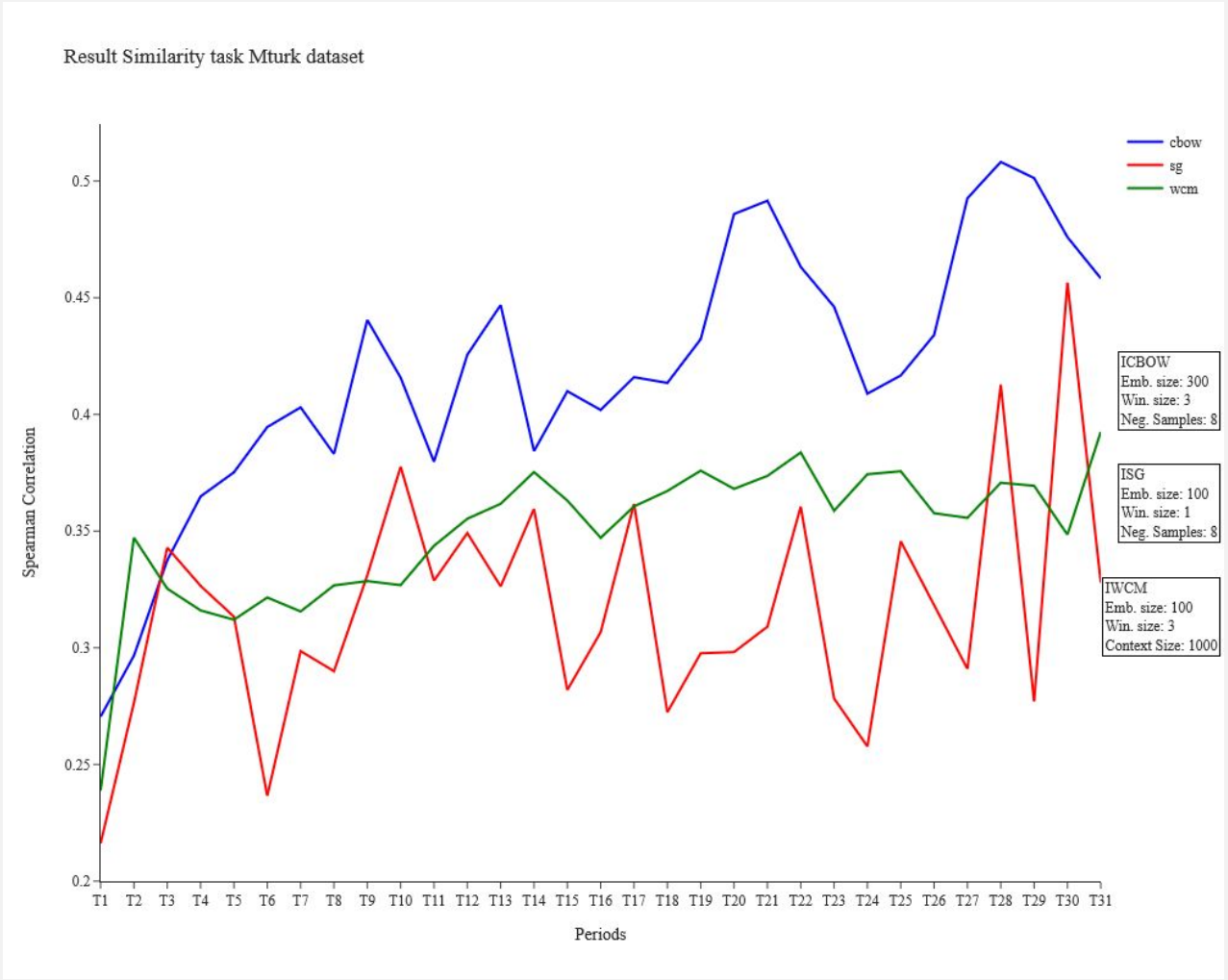
		Hyperparameters				Result tasks			
Position	Model	Emb. size	Win. size	Num. N.S	Context size	Mean MEN	Mean Mturk	Mean AP	Overall mean
1	ICBOW	100	3	6	-	0.488	0.439	0.294	0.407
2	ICBOW	300	3	8	-	0.507	0.428	0.284	0.406
3	ICBOW	300	3	6	-	0.508	0.416	0.289	0.404
4	ISG	100	1	8	-	0.44	0.4	0.321	0.387
5	ISG	100	1	6	-	0.443	0.393	0.312	0.383
6	ISG	100	2	10	-	0.421	0.399	0.309	0.376
7	IWCM	100	3	-	1000	0.44	0.343	0.319	0.367
8	IWCM	200	3	-	1000	0.438	0.351	0.307	0.366
9	IWCM	300	3	-	1000	0.439	0.35	0.307	0.365

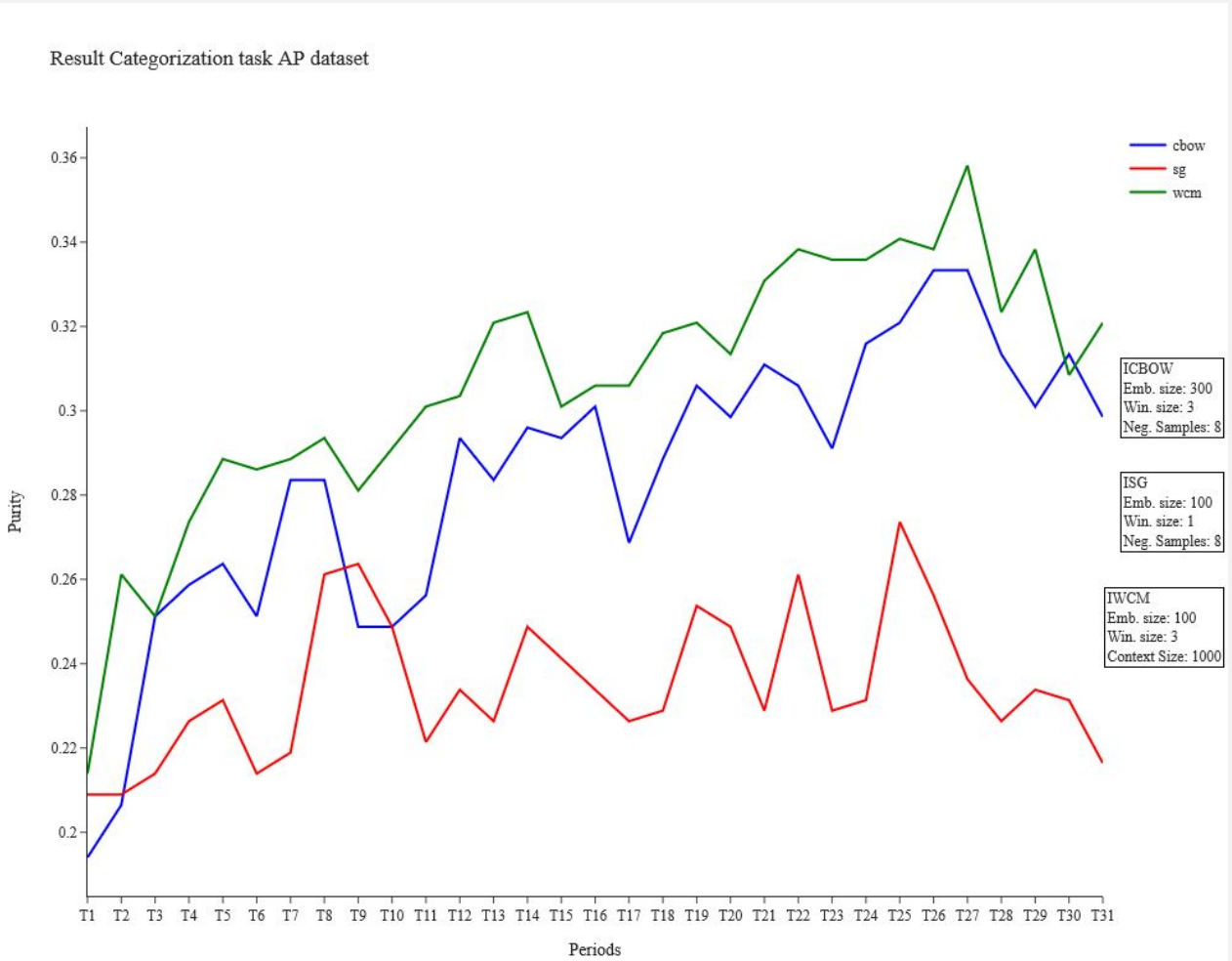
Results

Table 1: The Overall Ranking of the benchmark results is based on the average of the Periodic Evaluation applied across the text stream. The result tasks are calculated by finding the mean of the evaluation, and the overall mean determined by taking the average of these result tasks. This overall mean then determines the position in the ranking.

		Hyperparameters				Result tasks			
Position	Model	Emb. size	Win. size	Num. N.S	Context size	Mean MEN	Mean Mturk	Mean AP	Overall mean
1	ICBOW	100	3	6	-	0.488	0.439	0.294	0.407
2	ICBOW	300	3	8	-	0.507	0.428	0.284	0.406
3	ICBOW	300	3	6	-	0.508	0.416	0.289	0.404
4	ISG	100	1	8	-	0.44	0.4	0.321	0.387
5	ISG	100	1	6	-	0.443	0.393	0.312	0.383
6	ISG	100	2	10	-	0.421	0.399	0.309	0.376
7	IWCM	100	3	-	1000	0.44	0.343	0.319	0.367
8	IWCM	200	3	-	1000	0.438	0.351	0.307	0.366
9	IWCM	300	3	-	1000	0.439	0.35	0.307	0.365


Result Similarity task MEN dataset
ICBOW
ISG
IWCM
ICBOW
Emb. size: 300
Win. size: 3
Neg. Samples: 8
ISG
Emb. size: 100
Win. size: 1
Neg. Samples: 8
IWCM
Emb. size: 100
Win. size: 3
Context Size: 1000
Spearman Correlation
0.4
0.35
0.3
0.25
0.2
T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 T11 T12 T13 T14 T15 T16 T17 T18 T19 T20 T21 T22 T23 T24 T25 T26 T27 T28 T29 T30 T31
Periods


Result Similarity task Mturk dataset
cbow
sg
wcm
ICBOW
Emb. size: 300
Win. size: 3
Neg. Samples: 8
ISG
Emb. size: 100
Win. size: 1
Neg. Samples: 8
IWCM
Emb. size: 100
Win. size: 3
Context Size: 1000
Spearman Correlation
0.5
0.45
0.4
0.35
0.3
0.25
0.2
T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 T11 T12 T13 T14 T15 T16 T17 T18 T19 T20 T21 T22 T23 T24 T25 T26 T27 T28 T29 T30 T31
Periods


Result Categorization task AP dataset
cbow
sg
wcm
ICBOW
Emb. size: 300
Win. size: 3
Neg. Samples: 8
ISG
Emb. size: 100
Win. size: 1
Neg. Samples: 8
IWCM
Emb. size: 100
Win. size: 3
Context Size: 1000
Purity
0.36
0.34
0.32
0.3
0.28
0.26
0.24
0.22
0.2
T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 T11 T12 T13 T14 T15 T16 T17 T18 T19 T20 T21 T22 T23 T24 T25 T26 T27 T28 T29 T30 T31
Periods

Discussion

- **Hyperparameter tuning** plays a **critical role** in optimizing a model's performance. It involves adjusting the settings of a model's parameters to achieve better results.

- The **ICBOW** model shows **higher performance in the similarity task** compared to the categorization task.

Discussion

- On the other hand, the **ISG and IWCM** models exhibit better performance in the **categorization task** and outperform the ICBOW model.

- It's important to note that the **results of a model** for a specific intrinsic evaluation task and dataset **can vary significantly** and **may not always be related** to other tasks or datasets.

Weaknesses

- One disadvantage of the periodic evaluation approach is that it **cannot detect concept drift**, which is a common problem in streaming data.
- Concept drift refers to **changes in the data distribution** that may occur over time, significantly affecting the models' performance.
- Although the proposed approach allows for visualizing the model's performance throughout the training process, **it fails to capture the effect of concept drift on the models**.
- We strongly recommend that the results be taken with **caution** in streaming environments.

Conclusions

- RiverText provides a systematic approach for **training and evaluating** Incremental WE models using text **data streams**.
- Allows a **standardization** for the incremental WE into a **unified methodology** for comparing them.
- It provides a robust **evaluation method** based on intrinsic NLP tasks, which are **adapted to a streaming environment**.

Future Work

- We plan to incorporate additional **incremental text representation methods**, such as **incremental Glove** [3].

- Introducing a `Dataloader` capable of **connecting and monitoring social media** platforms such as Twitter.

Future Work

Our current periodic evaluation approach assumes static golden relations (e.g., word pair similarities, categories) throughout the stream, which is **insufficient for evaluating the adaptability** of our word vectors to change.

- We aim to propose a new evaluation methodology that considers **concept drift**, which reflects **semantic changes** in words.
- The idea is to develop an approach that **creates synthetic data**, inducing **semantic changes in the text**, and extending it to the intrinsic NLP tasks.

Future Work

- Add additional algorithms for **tracking new words** in the text stream, such as the **Space Saving** algorithm.
- Incorporate more **sketching techniques** that allow for gathering **more information** from the text stream while it is being processed.
- Integrate incremental **detection of collocations or phrases** for the representation of multi-word expressions, such as "New Zealand" or "New York", within our vocabulary.

Thanks for your Attention

RiverText: A Framework for Training and Evaluating Incremental Word Embeddings from Text Data Streams.

https://dccuchile.github.io/rivertext/

References

- [1] Nobuhiro Kaji and Hayato Kobayashi. 2017. Incremental Skip-gram Model with Negative Sampling. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 363–371, Copenhagen, Denmark. Association for Computational Linguistics.
- [2] Bravo-Marquez, Felipe, Arun Khanchandani, and Bernhard Pfahringer. "Incremental word vectors for time-evolving sentiment lexicon induction." Cognitive Computation (2022): 1-17.
- [3] Peng, Hao, et al. "Incremental term representation learning for social network analysis." Future Generation Computer Systems 86 (2018): 1503-1512
- [4] May, Chandler, et al. "Streaming word embeddings with the space-saving algorithm." arXiv preprint arXiv:1704.07463 (2017).
- [5] Montiel, Jacob, et al. "River: machine learning for streaming data in python." The Journal of Machine Learning Research 22.1 (2021): 4945-4952.