Benchmarking Feature Extraction Techniques for Textual Data Stream Classification

Bruno Siedekum Thuma, Pedro Silva de Vargas
Polytechnic School
Pontifícia Universidade Católica do Paraná (PUCPR)
Curitiba, Brazil
{bruno.thuma, pedro.vargas}@pucpr.edu.br

Cristiano Garcia, Alceu de Souza Britto Jr.,
Jean Paul Barddal
Graduate Program in Informatics (PPGIa)
Pontifícia Universidade Católica do Paraná (PUCPR)
Curitiba, Brazil
{cristiano.garcia, alceu, jean.barddal}@ppgia.pucpr.br

***Abstract*—Feature extraction regards transforming unstructured or semi-structured data into structured data that can be used as input for classification and sentiment analysis algorithms, among other applications. This task becomes even more challenging and relevant when textual data becomes available over time as a continuous data stream since the lexicon and semantics can be ever-evolving. Data streams are, by definition, potentially infinite sequences of data that may have ephemeral characteristics, that is, where the data behavior changes, it leads to a phenomenon named concept drift. Textual data streams are specialized data streams, in which texts arrive over time from a continual data source, such as social media, raising challenges in which feature extractors are of great help. In this paper, we benchmark different feature extraction algorithms, i.e., Hashing Trick, Word2Vec, BERT, and Incremental Word-Vectors; in textual data stream classification, considering different stream lengths. The evaluation was performed over a binary and a multiclass classification task, considering two different datasets. Results show that pre-trained models, such as BERT, achieve interesting results, while Hashing Trick also performs competitively. We also observe that incremental methods such as Word2Vec and Incremental Word-Vectors are the most prepared for changing scenarios, yet, they are much more computationally intensive compared to the former when applied to larger streams.**

***Index Terms*—Data stream mining, textual data, classification, feature extraction.**

I. Introduction

Machine learning has been increasingly studied in the last years, with several usages in day-to-day technologies. Due to the massive amounts of data generated at high speeds on social media, much effort has been put into processing and extracting useful knowledge from textual data streams. A common source for textual data is Twitter, where data can be easily accessed and analyzed, such as shown in [1], [2], where the authors loaded data streams from Twitter to continuously update sentiment lexicons automatically. Textual data are usually represented by texts and phrases, and thus, they require feature extraction techniques to be applied so that a vector representation is obtained prior to the application of learning algorithms.

In this work, we are particularly interested in scenarios in which textual data is made available for a learning system over time. In such scenarios, it is relevant to develop systems that understand and interpret human phrases in which words or sentences may exhibit drifting behavior over time due to temporal or cultural factors. The machine learning area that focuses on handling evolving data sequences is called data stream mining [3], and the phenomenon in which data behavior changes is called concept drift [4], [5]. Therefore, it is relevant to assess whether different feature extraction techniques are suitable for data stream classification. More specifically, we target four approaches for feature extraction from data streams: (i) Hashing Trick [6], which vectorizes words into indices of a vector; (ii) Word2Vec [7], which essentially consists in neural network methods for generating high-quality word representation; (iii) BERT [8], a bidirectional language representation model based on Transformers [9] that learn word contexts from their surrounding (left and right) words; and (iv) Incremental Word-Vectors [2], an incremental method that maintains a co-occurrence matrix, taking advantage of the positive point-wise mutual information (PPMI) calculation to generate a representation vector.

The interest relies on determining whether more complex techniques, such as Word2Vec and BERT, overcome traditional Hashing Tricks, which encode word tokens without contextual information, and Incremental Word-Vectors, which encode words using a co-occurrence matrix, regarding accuracy and processing speed.

This paper is divided as follows. Section II introduces data stream classification and concept drift. Section III details existing feature extraction for text data in data stream scenarios. Section IV brings forward the experimental protocol adopted for the proposed benchmark. Section V discusses the results obtained and the main findings. Finally, Section VI concludes this work and states envisioned future works.

II. Data Stream Classification

Classification is one of the most popular data mining tasks. In this task, there are the predictive information and their respective labels (a target attribute, or class). The idea of classification essentially regards mapping predictive information into categorical labels [10]. Formally, in classification, a predictive model is trained to map input data $\vec{x} = (x_1, x_2, \ldots, x_d)$ into a set of classes y accurately.

In order to train classifiers in traditional machine learning settings, it is necessary to create partitions over a dataset: one for training, and a second for evaluation. However, streaming environments have characteristics that hamper traditional machine learning methods. For instance, (i) data is continuously made available over time, (ii) is potentially unbounded, and (iii) is potentially nonstationary [11], meaning that the relationship between the features available and the classes may drift over time. In other words, traditional machine learning models cannot learn data one by one, and they are unable to recognize different patterns from those known during the training process, quickly becoming obsolete.

Therefore, the challenges for machine learning in streaming settings include: (i) learning from the data stream one by one or in small batches; (ii) using a limited amount of storage and memory; and (iii) being capable of updating itself in order to better reflect the data changes in the stream.

In particular, data changes are called concept drifts [4], [5], [12], and such drifts may render classification models obsolete. Consequently, it is of the utmost importance for classification models to be updated over time. In the textual data stream context, it is relevant to verify whether adaptive learning algorithms depend on techniques that incrementally update data representation based on the application context or not. This research question is relevant to determine whether more effort should be put into researching new learning or data representation algorithms.

III. Feature Extraction from Textual Data Streams

Data stream mining and natural language processing have grown in importance and acknowledgment over the last few years. Yet, the number of works that tackle their intersection is relatively small compared to the number of proposals brought forward to each area individually. In this work, we focus on techniques for feature extraction from textual data streams, i.e., Hashing Trick, Word2Vec, Bidirectional Encoder Representations from Transformers (BERT), and Incremental Word-Vector (IWV); and the application of such features in the classification task. These techniques were chosen due to their incremental capabilities, i.e., the ability to handle new word tokens as they appear in textual data streams, even though such methods were not initially developed for such scenarios.

A. Hashing Trick

Hashing Trick [6] uses hash functions for sentence vectorization. In practice, Hashing Trick applies arithmetic operations or sequences of them to compute term frequencies while reducing the data representation dimensionality into a fixed length n. As a result, these operations return non-reversible tabulated values, i.e., they cannot be translated back into the original expression. Furthermore, one of the hashing consequences is the incidence of collisions, i.e., when different words are translated into the same value in $[0; n-1]$. Such approaches are widely applied for classification problems in tools such as Vowpal Wabbit[1] [13].

A popular application of Hashing Trick in online learning systems is spam filtering [6], in which authors developed dynamic spam filters where large amounts of e-mail contents were vectorized using Hashing Trick and later applied to an incremental classifier. Hashing Trick was a suitable choice in this application since no previous knowledge of the lexeme was required. Furthermore, the hashing function handles new words by embedding them into one of the slots available in the n-sized vector. Nonetheless, this technique does not account for context information, and collisions may occur with others with different (or even contrasting) meanings.

[1]Vowpal Wabbit is available at https://vowpalwabbit.org

B. Word2Vec

Unlike Hashing Trick, which uses mathematical functions to map words into a predefined number of features, Word2Vec [7] is built upon neural networks that allow contextual analysis by associating words to one another and also to the contexts it is associated with. Word2Vec creates and selects numerical vectors for each word in its lexeme, so that related terms have similar representation when assuming dissimilarity metrics. Since Word2Vec relies on neural networks, it is relevant to highlight that (i) the training of these models is preferably performed on top of large datasets and (ii) they allow incremental learning as new words can be added to the representation as new data becomes available, yet, at the expense of elevated training times

C. Bidirectional Encoder Representations from Transformers (BERT)

Bidirectional Encoder Representations from Transformers (BERT) is a deep learning approach for textual data representation learning proposed in [8]. BERT's goal is to provide a generic solution that requires minimal customization for specific tasks before its potential application with predictive algorithms. BERT's structure relies on Transformers, which are deep learning-based techniques that analyze sentences both from the beginning to the end and vice-versa, thus giving rise the the 'bidirectional' term in its name.

Similarly to Word2Vec, BERT allows incremental learning as new data becomes available since it relies on neural networks. However, BERT depends on fine-tuning to keep internal structures updated, which requires more significant training times. In this paper, we used a pre-trained Sentence-BERT (SBERT), a BERT siamese-network-based architecture for sentence representation [14], without any update mechanism (e.g. without fine-tuning). We used the flavor 'paraphrase-MiniLM-L6-v2'[2], which encodes sentences into 384-dimension vectors.

D. Incremental Word-Vectors

Incremental Word-Vector [2] is a technique for vector representation generation that relies on a co-occurrence matrix. A co-occurrence matrix is a $|V| \times |C|$ matrix, in which $|V|$ is vocabulary size, while $|C|$ is the context size. Considering a window w and a target word t, the context of t are the w words previous to t, and the w words after t. Between t and each of the context words, the positive point-wise mutual information (PPMI) is calculated. PPMI is computed as follows:

$$PPMI(t,c) = \max\left(0, \log_2\left(\frac{\text{count}(t,c) \times D}{\text{count}(t) \times \text{count}(c)}\right)\right), \quad (1)$$

where t is the target (central) word, c is a word belonging to the context of t, D is the number of words in the text stream until the moment it is calculated, $\text{count}(\cdot)$ is the number of appearances of a specific word until the moment it is calculated. The idea behind PPMI is that it measures

[2]https://huggingface.co/sentence-transformers/paraphrase-MiniLM-L6-v2

TABLE I

SAMPLES FROM THE SENTIMENT140 DATASET, CONTAINING THE FEATURES INDEX, TARGET, AND TEXT.

index	target	text
0	0	"@switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer. You shoulda got David Carr of Third Day to do it. ;D"
1	0	"is upset that he can't update his Facebook by texting it... and might cry as a result School today also. Blah!"
1599997	4	"Are you ready for your MoJo Makeover? Ask me for details"
1599998	4	"Happy 38th Birthday to my boo of alll time!!! Tupac Amaru Shakur"

the degree of relationship between words. A negative value reveals that two words co-occur less than by chance [2].

Finally, in opposition to the previous approaches, Incremental Word-Vector allows the context to be interpreted, since its dimensions correspond to real words.

IV. EXPERIMENTAL PROTOCOL

In this section, we bring forward aspects related to the experimental protocol, detailing datasets, pre-processing, and evaluation methods.

A. Datasets

In this experimentation, two datasets were chosen to evaluate the feature extractors in a streaming fashion: Sentiment140 [15][3] and Yelp [16]. Both datasets contain texts in the English language. The datasets are described as follows.

1) Sentiment140 Dataset: Sentiment140 Dataset contains 1.6 million rows (tweets) and six features, namely: (i) target, (ii) tweet id, (iii) date of posting, (iv) flag, (v) username, and (vi) text (post content). A sample (first and last two instances) of this dataset is displayed in Table I, in which only index, target, and text columns are displayed. In this experiment, only the text content and target columns were used. It is also worth mentioning that the polarity feature (target) is evenly distributed, i.e., the classes were balanced.

2) Yelp Dataset: Yelp Dataset[4] consists of data (reviews, pictures, and so on) collected from over 150 thousand businesses. We selected the dataset related to reviews, having as features the reviews' texts and associated ratings (stars). The dataset version utilized in this work contains 5,261,670 reviews distributed in an imbalanced manner among five rating values from 1 to 5. The data distribution is given in Figure 1.

Table II shows two selected reviews from Yelp Dataset. Comparing the contents of Table II against those of Table I, we notice that reviews usually contain more words than regular *tweets*. Calculating the average word count per row in each dataset, Sentiment140 has 14.4 ± 7.1 words, while Yelp has 114.1 ± 106.9 words.

B. Pre-processing and Implementation

Regarding data pre-processing, we transformed all texts into lowercase and removed special characters. The implementations used in this work were Gensim [17] and SentenceTransformers [14], for Word2Vec and BERT, respectively. For

[3]https://www.kaggle.com/datasets/kazanova/sentiment140

[4]https://www.yelp.com/dataset

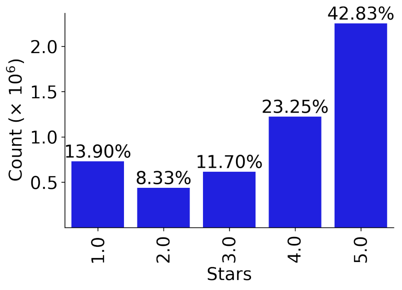

Fig. 1. Count plot and respective percentages of reviews by rating (stars) in Yelp Dataset

Hashing Trick and Incremental Word-Vectors, the implementation was coded by the authors following instructions from [6] and [2]. Gensim's Word2Vec allows incremental training, so we trained using common_texts (a toy in-library dataset) and updated on-the-fly. For BERT, we used a pre-trained model, namely `paraphrase-MiniLM-L6-v2`. Due to BERT's incremental updating time (in a fine-tuning fashion), it was unfeasible to use it incrementally. Therefore, we experimented with BERT with 384-dimension representations, while Hashing Trick, Incremental Word-Vectors, and Word2Vec used 100, 384, and 500 dimensions for both. Henceforth, these approaches are referred to as BERT, Hashing Trick (100), Hashing Trick (384), Hashing Trick (500), Incremental Word-Vectors (100), Incremental Word-Vectors (384), Incremental Word-Vectors (500), Word2Vec (100), Word2Vec (384), and Word2Vec (500).

To make the experiment reproducible, we describe the parameters of each approach in this work: (i) BERT: no parameter needed ; (ii) Hashing Tricks: hashing function used was the modulo operation (rest of a division); (iii) Word2Vec: window=5, min_count=1 (defines the least word frequency to take it into account); and (iv) Incremental Word-Vectors: vocabulary_size=10000; window_size=7.

The window_size (for both Word2Vec and Incremental Word-Vectors) impacts directly the time for learning/updating vector representations. Also, according to [2], the window length is commonly "between 3 and 17". Yet, "whereas shorter windows are likely to capture syntactic information, longer windows are more useful for representing meaning" [2], [18]. Thus, for Word2Vec, we chose window=5 since it is the default value, while for Incremental Word-Vectors, window_size=7 would capture slightly more information on the words' semantics.

TABLE II

SAMPLE REVIEWS FROM YELP DATASET.

index	stars	text
0	5.0	"Super simple place but amazing nonetheless. It's been around since the 30's and they still serve the same thing they started with: a bologna and salami sandwich with mustard. \n\nStaff was very helpful and friendly"
9	3.0	"Not bad!! Love that there is a gluten-free, vegan version of the cheese curds and gravy!!\n\nHaven't done the poutine taste test yet with smoke's but Im excited to see which is better. However poutini's might win as they are vegan and gluten-free"

The source code was written in Python 3, and the experiments were run on the Google Collaborate platform, which provides 25GB of RAM and 225.98GB of storage. The source codes will be made available after peer reviewing.

C. Evaluation Methods

To assess the feature extractors, evaluation can be either intrinsic or extrinsic, following the assessments performed in [19]. Intrinsic evaluation regards the comparison among the methods in terms of representation quality. Since different methods generate different embeddings in different positions of the vector space, it does not mean that one is necessarily better than the other. Since we target classification tasks, we conducted an extrinsic evaluation in which the downstream classification task assessed the feature extractors.

To evaluate the feature extraction methods in an extrinsic manner, we used the Updatable Naive Bayes as the classification algorithm. The metric used to assess the classification was accuracy. Both Naive Bayes and accuracy metric are available in RiverML library [20] for online (incremental) use. Accuracy was chosen since it is a regular metric for models' performance in classification tasks. Naive Bayes was selected due to its simplicity and constant complexity regarding attributes and instances.

Further, we randomly selected shuffled subsets of different sizes from Sentiment140 and Yelp Datasets, hereafter indicated as stream length. The stream lengths are 10,000; 20,000; 30,000; 50,000; 100,000, and 200,000. These lengths are referred to in this paper as 10k, 20k, 30k, 50k, 100k, and 200k, respectively. Since the data was shuffled to avoid bias, each experiment was executed ten times in a streaming fashion, thus obtaining average accuracy and standard deviation as results.

It is well-known that, in streaming environments, data arrives continuously and at high speeds. Therefore we considered the run time a critical aspect to be monitored. Hence, in the analysis section, we also compare the run times for the feature extractors in the aforementioned extrinsic experiment.

V. Analysis

The results obtained for the Sentiment140 dataset are available in Table III. In all the proposed scenarios, BERT reached the best accuracy values. It is also possible to state that BERT provides stable results ranging the stream lengths from 10k to 200k. Besides, we can notice that Word2Vec, using 100 and 384 dimensions, reached roughly 54% for 10k but only approximated this value when feeding the model with 200k rows. One hypothesis for this decay between 10k and 50k is that Word2Vec may take longer to stabilize new words representation. Also, it is relevant to notice that Hashing Trick overcame Word2Vec in the classifier accuracy in all settings.

For Incremental Word-Vectors, all the variants performed similarly. However, using 100 dimensions led to a slightly smaller accuracy among the variants.

Comparing the approaches, simple vector representations provided by Hashing Trick, although with no learning mechanism, can better help the model, even without accounting for contextual data. Only BERT could provide better representations than Hashing Trick in this scenario. It is plausible

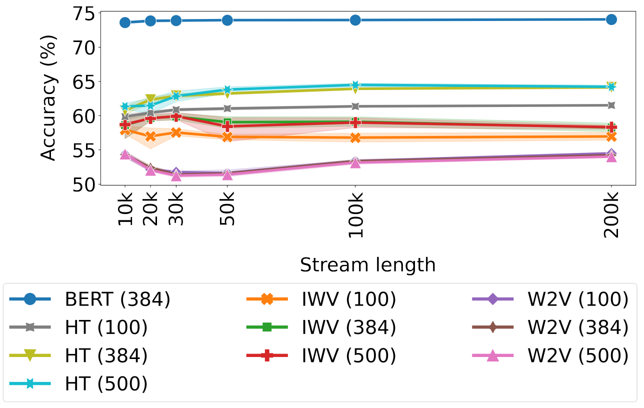

Fig. 2. Accuracy rates (%) over stream lengths per extractor in Sentiment140.

since we used a pre-trained BERT model, which previously has learned representations from several texts.

However, Incremental Word-Vectors overcame Word2Vec, while performing poorly compared to BERT and competitively compared to Hashing Tricks. Figure 2 depicts the accuracy rates per feature extractor across the stream lengths. We see that Word2Vec had roughly the same behavior across all the settings, regardless of the dimensionality.

Also, Table IV describes the run times of each experiment, considering both total run times and run times per row. For the sake of readability, consider "Hash. T" as Hashing Trick, "IWV" as Incremental Word-Vectors, and "W2V" as Word2Vec. It is straightforward to notice that both BERT, Incremental Word-Vectors, and Hashing Trick (in all configurations) performed steadily, although BERT showed a slight increase in the setting with 200k of stream length, considering the total time. We also see that Word2Vec almost doubled the time spent per row from one setting to another. A reason for that is Word2Vec's complexity, which increases with the number of words and vocabulary size (i.e., roughly $N \times D + N \times D \times \log_2(V)$). Since N and V increase over time, it is expected to have a growing processing time accompanying processed rows. Figure 3 shows the time spent per row. It is possible to confirm the steadiness of both BERT, Incremental Word-Vectors, and Hashing Trick in all configurations, and also the increase of Word2Vec's processing time as the stream length increases.

For Yelp Dataset, we also calculated accuracy, total run times, and run times per row for each setting in the downstream task. The accuracy rates obtained are shown in Table V. We notice that Hashing Trick obtained competitive results compared to BERT for all stream lengths. Furthermore, it is interesting to notice that, for this dataset, Word2Vec provided poor word representation (for streams of length 10k and 20k). We can affirm that because the performance rates obtained for these settings are comparable to a random classifier. However, as the number of rows used in the stream increased, Word2Vec generated better representations which helped Naive Bayes achieve higher accuracy rates. All the variations of Incremental Word-Vectors achieved approximately 30% of accuracy. The lower accuracy obtained by this approach may have happened due to the small dimension sizes. According

TABLE III
ACCURACY RATES (%) PER FEATURE EXTRACTOR AND DIFFERENT STREAM LENGTHS FOR SENTIMENT140.

Extractor / Stream length	**10k**	**20k**	**30k**	**50k**	**100k**	**200k**
BERT	73.59 ± 0.30	73.83 ± 0.30	73.87 ± 0.25	73.93 ± 0.12	73.95 ± 0.14	74.04 ± 0.10
Hashing Trick (100)	59.85 ± 0.75	60.42 ± 0.42	60.86 ± 0.37	61.04 ± 0.37	61.36 ± 0.28	61.50 ± 0.14
Hashing Trick (384)	60.74 ± 1.03	62.32 ± 0.81	62.88 ± 0.63	63.22 ± 0.31	63.91 ± 0.24	64.13 ± 0.32
Hashing Trick (500)	61.35 ± 0.78	61.46 ± 1.95	62.84 ± 1.32	63.80 ± 0.75	64.48 ± 0.47	64.21 ± 0.43
Incremental Word-Vectors (100)	57.92 ± 0.96	56.98 ± 2.77	57.54 ± 0.86	56.89 ± 0.65	56.79 ± 1.07	56.96 ± 0.99
Incremental Word-Vectors (384)	58.61 ± 2.86	59.66 ± 0.74	59.85 ± 0.83	59.04 ± 1.50	59.10 ± 1.19	58.27 ± 1.07
Incremental Word-Vectors (500)	58.71 ± 3.11	59.60 ± 0.76	59.91 ± 0.90	58.43 ± 3.10	58.99 ± 1.20	58.32 ± 0.51
Word2Vec (100)	54.33 ± 0.82	52.15 ± 0.52	51.75 ± 0.35	51.62 ± 0.82	53.37 ± 0.34	54.49 ± 0.28
Word2Vec (384)	54.31 ± 0.70	52.41 ± 0.56	51.39 ± 0.35	51.55 ± 0.41	53.34 ± 0.43	54.21 ± 0.25
Word2Vec (500)	54.42 ± 0.78	52.05 ± 0.45	51.24 ± 0.40	51.40 ± 0.52	53.14 ± 0.42	54.01 ± 0.28

TABLE IV
RUN TIME (s) PER FEATURE EXTRACTOR AND DIFFERENT STREAM LENGTHS FOR SENTIMENT140.

Extractor / Stream length		**10k**	**20k**	**30k**	**50k**	**100k**	**200k**
BERT	**Total**	213.42 ± 4.88	422.77 ± 14.25	660.95 ± 14.98	1125.47 ± 35.66	2217.88 ± 26.56	4932.79 ± 310.93
	Per row	0.0213	0.0211	0.022	0.0225	0.0222	0.0247
Hash. T. (100)	**Total**	6.29 ± 0.35	12.7 ± 0.89	18.96 ± 1.03	31.92 ± 1.78	63.74 ± 4.24	127.42 ± 7.82
	Per row	0.0006	0.0006	0.0006	0.0006	0.0006	0.0006
Hash. T. (384)	**Total**	21.52 ± 1.96	42.89 ± 3.48	63.93 ± 4.93	106.39 ± 8.5	216.58 ± 12.81	434.85 ± 26.45
	Per row	0.0022	0.0021	0.0021	0.0021	0.0022	0.0022
Hash. T. (500)	**Total**	26.63 ± 0.4	53.07 ± 0.62	79.81 ± 0.95	133.45 ± 2.75	269.08 ± 2.23	533.36 ± 8.35
	Per row	0.0027	0.0027	0.0027	0.0027	0.0027	0.0027
IWV (100)	**Total**	710.73 ± 11.5	1729.4 ± 20.09	2420.92 ± 66.48	4858.93 ± 185.66	7348.09 ± 892.67	16692.16 ± 913.44
	Per row	0.0711	0.0865	0.0807	0.0972	0.0735	0.0835
IWV (384)	**Total**	668.11 ± 14.48	1659.94 ± 80.21	2501.69 ± 105.04	4325.03 ± 441.52	8708.62 ± 789.87	18060.92 ± 1836.85
	Per row	0.0668	0.083	0.0834	0.0865	0.0871	0.0903
IWV (500)	**Total**	687.3 ± 14.88	1635.75 ± 22.61	2488.11 ± 227.41	3866.61 ± 282.6	8489.01 ± 600.57	17351.82 ± 1557.98
	Per row	0.0687	0.0818	0.0829	0.0773	0.0849	0.0868
W2V (100)	**Total**	225.36 ± 16.05	689.46 ± 35.05	1338.48 ± 120.98	2909.53 ± 324.99	10411.76 ± 422.12	36200.74 ± 1073.09
	Per row	0.0225	0.0345	0.0446	0.0582	0.1041	0.1810
W2V (384)	**Total**	360.79 ± 12.28	944.37 ± 119.06	1994.55 ± 81.9	3936.69 ± 724.47	14234.94 ± 737.3	44669.56 ± 4544.43
	Per row	0.0361	0.0472	0.0665	0.0787	0.1423	0.2233
W2V (500)	**Total**	401.74 ± 27.64	1005.27 ± 131.53	2128.81 ± 176.66	4744.57 ± 418.95	15465.48 ± 1910.43	48492.94 ± 4163.78
	Per row	0.0402	0.0503	0.071	0.0949	0.1547	0.2425

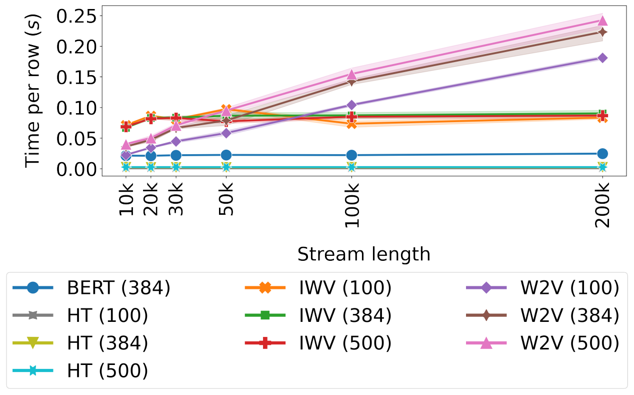

Fig. 3. Time spent per row (s) over stream lengths per extractor using Sentiment140 Dataset

to [2], once the contexts (limited by the dimension size) are defined in Incremental Word-Vectors structure, they do not change along the stream. It means that, although Incremental Word-Vectors is an incremental approach, its contexts are fixed and may become obsolete. Yet, a short context size can lead to poor vector representations. The experiments provided in [2] corroborates with this statement due to the smallest context size used in the aforementioned work was 500, but a context size of 10,000 led to the best accuracy. Thus, as mentioned before, since the average number of words per instance in Yelp dataset is higher than in Sentiment140 dataset, the contexts may become obsolete within a shorter time.

Figure 4 shows the performance of the extractors over the stream lengths. As aforementioned, we can see that Word2Vec starts with poor representations, improving them as the stream length increases. Also, in the 200k setting, Word2Vec (100) achieved competitive accuracy rates compared to BERT and Hashing Trick, which reached the best results among the approaches compared in this work. Also, Word2Vec (384) and Word2Vec (500) had more difficulty to provide good word representations compared to Word2Vec (100), meaning that representations with higher dimensions take longer to stabilize. Incremental Word-Vectors performed slightly better than a random classifier in all the dimension sizes tested in this work. We can also notice that the accuracy trend for this approach is decaying, suggesting the obsolescence of the contexts once internally defined by the approach.

It is worth noticing that Word2Vec, considering all the scenarios with both datasets, had completely different behaviors. A cause for that is the average number of words in the sentences in Sentiment140's tweets and Yelp's reviews. The input data (sentence representation) is calculated using the average of all words representations of the sentences. It means that, with more words, the sentence representation may not represent well the sentence itself, specially in scenarios with less data. Also, in scenarios with bigger sentences, more unknown words have their representations initialized at the same time. In general, the initial representations are poor and evolve over time, as the represented words appear

TABLE V

ACCURACY RATES (%) PER FEATURE EXTRACTOR AND DIFFERENT STREAM LENGTHS FOR YELP.

Extractor / Stream length	**10k**	**20k**	**30k**	**50k**	**100k**	**200k**
BERT	45.01 ± 0.73	44.81 ± 0.72	45.07 ± 0.34	45.05 ± 0.32	45.04 ± 0.30	45.00 ± 0.16
Hashing Trick (100)	41.43 ± 0.37	41.77 ± 0.24	41.97 ± 0.31	41.89 ± 0.20	42.00 ± 0.15	42.08 ± 0.13
Hashing Trick (384)	43.32 ± 0.38	43.32 ± 0.36	43.58 ± 0.30	43.69 ± 0.20	43.75 ± 0.20	43.87 ± 0.08
Hashing Trick (500)	43.44 ± 0.53	43.71 ± 0.39	43.88 ± 0.31	43.98 ± 0.23	44.05 ± 0.15	44.17 ± 0.15
Incremental Word-Vectors (100)	30.34 ± 5.01	31.60 ± 3.55	30.43 ± 5.47	32.67 ± 6.26	29.14 ± 5.02	31.66 ± 2.81
Incremental Word-Vectors (384)	31.07 ± 1.68	33.22 ± 3.50	31.24 ± 2.21	32.69 ± 3.80	31.92 ± 4.37	30.07 ± 2.10
Incremental Word-Vectors (500)	30.84 ± 1.86	31.35 ± 4.11	30.36 ± 2.33	32.83 ± 3.8	29.37 ± 2.12	28.92 ± 1.71
Word2Vec (100)	25.57 ± 5.18	28.38 ± 3.83	31.06 ± 1.93	35.35 ± 1.49	39.71 ± 0.60	41.81 ± 1.15
Word2Vec (384)	15.12 ± 3.66	24.24 ± 3.87	28.14 ± 1.70	32.54 ± 1.73	37.81 ± 1.21	39.82 ± 1.24
Word2Vec (500)	18.28 ± 5.18	25.31 ± 2.80	29.04 ± 2.17	32.42 ± 1.72	37.70 ± 0.86	39.26 ± 0.72

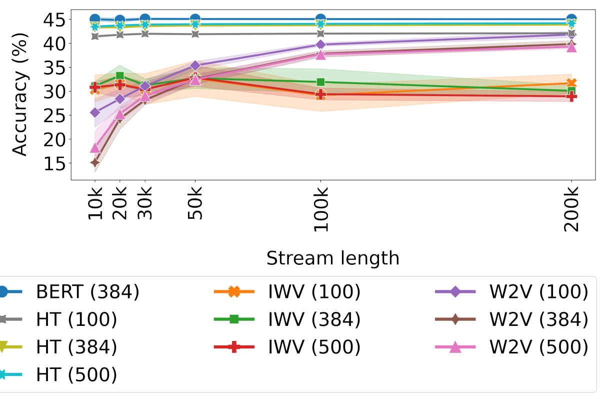

Fig. 4. Accuracy rates (%) across stream lengths per extractor using Yelp dataset.

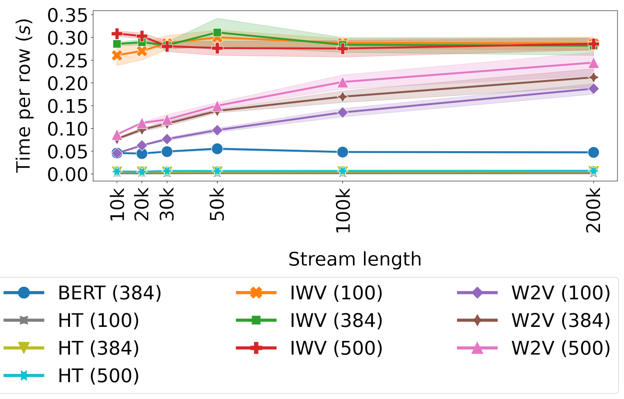

Fig. 5. Time spent per row (s) over stream lengths per extractor using Yelp dataset.

in the stream. Therefore, using Yelp dataset, it takes longer for Word2Vec to reach an acceptable accuracy.

Regarding run times, we can see that, at the most challenging scenario (i.e. 200k), Word2Vec (500) took about five times longer than BERT to process the stream. Hashing Trick (100) processed the inputs in 3% of the time BERT needed. It is expected since Hashing Trick use simpler mathematical operations. Using more dimensions, Hashing Trick (384) and Hashing Trick (500) were still faster than BERT. They approximately needed 10% and 14%, respectively. Still comparing to BERT, on the other hand, Word2Vec, using 100 dimensions, took almost four times more to run. In the setting with 500 dimensions, Word2Vec spent five times more than BERT to run. Incremental Word-Vectors performed steadily across all stream lengths. Incremental Word-Vectors (500) took almost six times more than BERT to run, and about 16% more than Word2Vec (500).

We also calculated the time spent per row for Yelp Dataset. These results can be seen in Table VI. Again, for the sake of readability, consider “Hash. T” as Hashing Trick, “IWV” as Incremental Word-Vectors, and “W2V” as Word2Vec. We can realize that BERT, Incremental Word-Vectors, and Hashing Trick perform steadily over all scenarios. Also, Word2Vec has similar increase compared to its performance on Sentiment140 dataset. Since text streams have the potential to be infinite [3], Word2Vec may become unfeasible with larger streams, and may not be the best choice.

Figure 5 shows the measured times per row. We can see that BERT took almost twice the time per row than in Sentiment140 dataset setting. It is caused by the increase of words per sentence implied by Yelp dataset. Word2Vec also increased the time spent across the stream lengths. Hashing Trick in all settings could keep a very low time consumption per row, which is a good characteristic in streaming environments. However, Hashing Trick is not prepared to react to concept-drifting scenarios, since there is no inner representation learning mechanism.

We statistically compared the approaches in the most challenging scenario (i.e. 200k) for each dataset by using Friedman test and Nemenyi test, according to the procedures described in [21], in terms of accuracy and run times, separately. The null hypothesis (H_0) suggests that there is no significant difference among the observed accuracy rates, and among the run times. The alternative hypothesis (H_a) states that there is indeed a statistically significant difference among the accuracy values and among the run times. The value chosen for α (significance) is 0.05. Friedman test was the choice due to the impossibility to check data normality considering 10 runs.

Thus, considering the accuracy in Sentiment140 dataset, the value of p obtained from Friedman test was 4.88e-15, which is lower than 0.05 and thus, we rejected H_0. Therefore, to find out where the difference lies, Nemenyi test was applied. The critical distance (CD) calculated resulted in 4.28, considering $k = 10$ (i.e. classifiers) and $N = 10$ (i.e. runs per feature extractor, under the same conditions). These values repeated for every test we performed. Figure 6 shows the average rank and the critical distance (horizontal bars). The approaches comprised by a horizontal bar are statistically similar. For example, BERT, HT (500), HT (384),

TABLE VI
RUN TIME PER ROW (s) OF EACH FEATURE EXTRACTOR, CONSIDERING DIFFERENT STREAM LENGTHS FOR YELP DATASET.

Extractor / Stream length		10k	20k	30k	50k	100k	200k
BERT	**Total**	457.59 ± 15.84	891.79 ± 26.41	1472.35 ± 80.17	2759.62 ± 202.64	4802.82 ± 269.52	9455.51 ± 796.55
	Per row	0.0458	0.0446	0.0491	0.0552	0.0480	0.0473
Hash. T. (100)	**Total**	12.36 ± 1.71	22.25 ± 2.35	39.8 ± 3.33	71.31 ± 9.73	128.98 ± 6.08	281.91 ± 15.93
	Per row	0.0012	0.0011	0.0013	0.0014	0.0013	0.0014
Hash. T. (384)	**Total**	41.56 ± 5.82	85.55 ± 13.35	130.56 ± 23.91	233.95 ± 28.93	468.39 ± 60.74	1024.36 ± 47.93
	Per row	0.0042	0.0043	0.0044	0.0047	0.0047	0.0051
Hash. T. (500)	**Total**	53.76 ± 8.23	94.11 ± 6.12	193.25 ± 20.83	314.37 ± 41.71	643.09 ± 74.36	1352.68 ± 170.68
	Per row	0.0054	0.0047	0.0064	0.0063	0.0064	0.0068
IWV (100)	**Total**	2606.1 ± 376.33	5401.17 ± 627.06	8613.28 ± 775.04	14994.74 ± 1560.28	28765.25 ± 1841.39	57363.06 ± 4912.19
	Per row	0.2606	0.2701	0.2871	0.2999	0.2877	0.2868
IWV (384)	**Total**	2856.34 ± 68.31	5784.05 ± 219.52	8480.43 ± 394.3	15540.24 ± 2697.07	28364.13 ± 2840.33	56342.19 ± 6753.83
	Per row	0.2856	0.2892	0.2827	0.3108	0.2836	0.2817
IWV (500)	**Total**	3084.02 ± 109.61	6055.89 ± 215.65	8413.2 ± 583.8	13820.59 ± 1254.9	27543.86 ± 2971.54	57168.65 ± 4459.68
	Per row	0.3084	0.3028	0.2804	0.2764	0.2754	0.2858
W2V (100)	**Total**	449.46 ± 20.19	1250.88 ± 53.61	2293.47 ± 153.13	4802.98 ± 410.2	13500.21 ± 1607.22	37476.41 ± 3766.92
	Per row	0.0449	0.0625	0.0764	0.0961	0.135	0.1874
W2V (384)	**Total**	768.18 ± 45.21	1944.72 ± 130.91	3311.66 ± 266.42	6915.53 ± 350.01	16952.47 ± 2259.14	42417.82 ± 6594.94
	Per row	0.0768	0.0972	0.1104	0.1383	0.1695	0.2121
W2V (500)	**Total**	865.15 ± 33.36	2239.88 ± 43.22	3581.79 ± 559.01	7487.44 ± 572.58	20194.69 ± 2532.98	48901.26 ± 8709.55
	Per row	0.0865	0.112	0.1194	0.1497	0.2019	0.2445

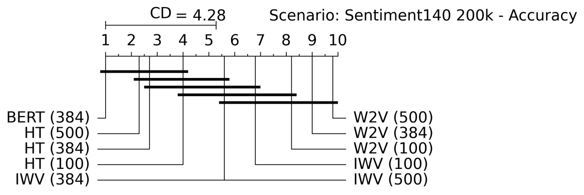

Fig. 6. Average ranking of approaches for Sentiment140 dataset and 10 runs, considering accuracy and stream lengths of 200k

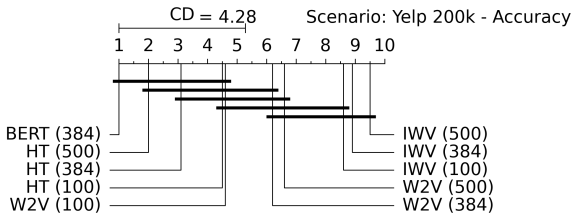

Fig. 8. Average ranking of approaches for Yelp dataset and 10 runs, considering accuracy and stream length of 200k

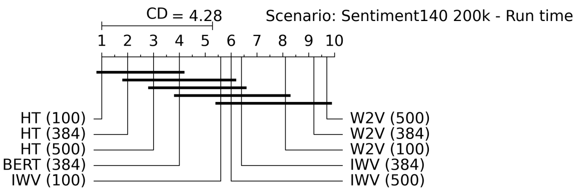

Fig. 7. Average ranking of approaches for Sentiment140 dataset and 10 independent runs, considering run times and stream length of 200k

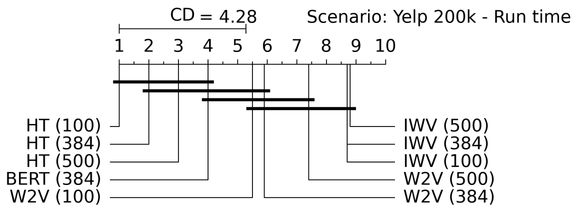

Fig. 9. Average ranking of approaches for Yelp dataset and 10 runs, considering run times and stream length of 200k

and HT (100) are statistically similar, taking into account the configurations of the experiments.

For Sentiment140 dataset, now considering the run times, the value obtained for p from Friedman test is 5.29×10^{-15}. Figure 7 shows the average rankings throughout 10 runs. We can clearly see that Hashing Tricks in all configurations appeared among the highest ranks, being statistically similar to BERT. On the other hand, Word2Vec featured in the last ranks, ordered by the dimension number.

For Yelp dataset in the most challenging scenario, the value obtained for p in Friedman test was 9.76×10^{-15}. The Figure 8 shows the average rank and the critical distance (horizontal bars) for this specific scenario. It is possible to notice that, for the best ranks, it followed the same pattern as for Sentiment140. However, Word2Vec inverted positions with Incremental Word-Vectors, which featured in the last ranks in this scenario.

Considering the run times, the p obtained from Friedman test was 3.97e-14, lower than 0.05. Thus, we rejected H_0. The graph of average rank is shown in Figure 9. Again, we notice an inversion of positions between Incremental Word-Vectors and Word2Vec. Hashing Tricks and BERT remained in the same average ranks as in Sentiment140.

Thus, it is possible to notice that BERT ranked first in all the experiments considering accuracy, for both datasets. However, considering the number of runs, it is not possible to say BERT is the best. It is also possible to check that Word2Vec and Incremental Word-Vectors are not as good in both datasets as the other approaches. It happens, for Word2Vec, due to the well-known necessity of a huge amount of data to train neural networks. For Incremental Word-Vectors, the obsolescence of the context may negatively impact its accuracy along the stream.

Considering run times, Hashing Trick was also the fastest, appearing among the first three positions. The fewer the dimensions of representations, the fastest, considering only the Hashing Trick. Also, the number of dimensions of Word2Vec makes it slower and harder to learn from the data. For Incremental Word-Vectors, the dimension size also impacts directly the time for learning representations.

VI. Conclusion

In this paper, we described the use of different feature extractors (i.e. BERT, Hashing Trick, Incremental Word-Vectors, and Word2Vec) in a data stream classification setting, using two different datasets: Sentiment140 and Yelp. The results obtained show that pre-trained models such as BERT can be of great utility. Also, Hashing Trick was competitive, although it has no internal learning method. Word2Vec, on the other hand, achieved competitive results only in the most challenging scenario using Yelp. The Incremental Word-Vectors obtained slightly above-average results for both datasets. In spite of the fact that Word2Vec and Incremental Word-Vectors could not greatly perform in some settings in this work, they are the only ones that have incremental learning and therefore, they can update internal word representations and learn new ones.

In addition to the measurement of accuracy, we also measured total time and time per row in each setting, per approach. The values obtained show that BERT and Hashing Trick (in all settings) performed steadily, having minor increases across stream lengths. On the other hand, Word2Vec and Incremental Word-Vectors, since they have internal representation learning mechanism and they are updated on-the-fly, they take longer to perform actions. In spite of that, both Word2Vec and Incremental Word-Vectors are the only approaches among the compared in this work that are really capable of incrementally adapting themselves for changes in words representations.

Many of the datasets undoubtedly containing short-time concept drift in texts used in literature were collected by the authors and are not available online. Also, short-time drifting data mostly emerges during events such as elections, pandemics outbreaks and war. As future works, we intend to collect drifting text data, assess it and apply it to other feature extractor methods, such as Incremental Skip Gram with Negative Sampling [19], in order to propose even more challenging comparisons.

References

[1] A. Bifet and E. Frank, "Sentiment Knowledge Discovery in Twitter Streaming Data," in *International Conference on Discovery Science*. Springer, 2010, pp. 1–15.

[2] F. Bravo-Marquez, A. Khanchandani, and B. Pfahringer, "Incremental Word Vectors for Time-Evolving Sentiment Lexicon Induction," *Cognitive Comput.*, pp. 1–17, 2021.

[3] A. Bifet, G. Holmes, B. Pfahringer, P. Kranen, H. Kremer, T. Jansen, and T. Seidl, "MOA: Massive Online Analysis, a Framework for Stream Classification and Clustering," in *Proceedings of the First Workshop on Appl. of Pattern Analysis*, 2010, pp. 44–50.

[4] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A Survey on Concept Drift Adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, pp. 1–37, 2014.

[5] H. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, and T. Abdessalem, "Adaptive Random Forests for Evolving Data Stream Classification," *Machine Learning*, vol. 106, no. 9, pp. 1469–1495, 2017.

[6] J. Attenberg, K. Weinberger, A. Dasgupta, A. Smola, and M. Zinkevich, "Collaborative Email-Spam Filtering with the Hashing Trick," in *Proceedings of the Sixth Conference on Email and Anti-Spam*. Citeseer, 2009.

[7] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," *Adv. Neural Inf. Process. Syst.*, vol. 26, 2013.

[8] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is All You Need," *Adv. Neural Inf. Process. Syst.*, vol. 30, 2017.

[10] R. Goldschmidt, E. Passos, and E. Bezerra, *Data Mining*. Elsevier Brasil, 2015.

[11] J. Gama, *Knowledge Discovery from Data Streams*. CRC Press, 2010.

[12] D. Leite, I. Škrjanc, and F. Gomide, "An Overview on Evolving Systems and Learning from Stream Data," *Evol. Syst.*, vol. 11, no. 2, pp. 181–198, 2020.

[13] Q. Shi, J. Petterson, G. Dror, J. Langford, A. Smola, and S. Vishwanathan, "Hash Kernels for Structured Data," *J. Mach. Learn. Res.*, vol. 10, pp. 2615–2637, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1577069.1755873

[14] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. [Online]. Available: https://arxiv.org/abs/1908.10084

[15] A. Go, R. Bhayani, and L. Huang, "Twitter Sentiment Classification using Distant Supervision," *CS224N Project Report, Stanford*, vol. 1, no. 12, p. 2009, 2009.

[16] N. Asghar, "Yelp Dataset Challenge: Review Rating Prediction," *arXiv preprint arXiv:1605.05362*, 2016.

[17] R. Rehurek and P. Sojka, "Gensim–Python Framework for Vector Space Modelling," *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, vol. 3, no. 2, 2011.

[18] J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Pearson/Prentice Hall, 2009.

[19] N. Kaji and H. Kobayashi, "Incremental skip-gram model with negative sampling," *arXiv preprint arXiv:1704.03956*, 2017.

[20] J. Montiel, M. Halford, S. M. Mastelini, G. Bolmier, R. Sourty, R. Vaysse, A. Zouitine, H. M. Gomes, J. Read, T. Abdessalem, and A. Bifet, "River: Machine Learning for Streaming Data in Python," 2020.

[21] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *The Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.