Journal of Visual Language and Computing

journal homepage: www.ksiresearch.org/jvlc/

Implementing BERT With Ktrain Library For Sentiment Analysis

Hasnan Othman[a,*], Mohd Ridzwan Yaakub[b]

[a] Malaysian Administrative Modernisation and Management Planning Unit (MAMPU),Prime Minister Department,Putrajaya, 62502 Malaysia
b.Centre for Artificial Intelligence Technology, Universiti kebangsaan Malaysia, Bangi 43600, Malaysia

ARTICLE INFO

Article History:
Submitted 10.1.2022
Revised 10.2.2022
Second Revision 10.5.2022
Accepted 10.6.2022

Keywords:
Bi-directional Representation
Bi-directional Encoder
BERT
ktrain library
Sentiment Analysis
Transformers

ABSTRACT

Building the Sentiment Analysis (SA) model for classification tasks is challenging to boost computational efficiency while writing Machine Learning (ML) applications with the Python Library package. Its implementation with the pre-trained Bi-directional Encoder From Transformers (BERT) will produce better results for the experiment. However, one of the implementation issues is choosing library resources to develop the SA model in solving the Natural Language Processing (NLP) tasks. Implementing BERT with the Ktrain framework can make such jobs more accessible by allowing domain experts to further democratize ML with minimum code. The Ktrain Library package's experiment in implementing the BERT model on IMDB movie datasets obtained an accuracy of 92.8 percent compared to based line models using Support Vector Machine (SVM) and Logistic Regression (LR). Future works will apply the Ktrain library package with other ML models on diversified domains of datasets.

1. Introduction

Sentiment analysis(SA) on the text is the form of indirect assessment based on information gathered in a text which can be classified as positive, negative, or neutral. Three approaches to conducting SA on text are machine learning, lexicon-based, and hybrid. The machine learning strategy uses machine learning algorithms to classify text, whereas the lexicon-based approach requires the construction of manual lexicon dictionaries, which is more time-consuming [1].

The difficulties in building a model for SA inherent in mastering machine learning coding workflows might make it challenging for beginner programmers to determine appropriate library resources to be used. Machine learning workflows were composed of several components, such as building the model, inspection, and application, to ensure that the models become operational [2]. Python's ability to be used as a programming language is significantly more efficient than other programming languages in performing matrix operations and SA tasks [3]. Ktrain library lets you use many pre-trained deep learning architectures in Natural Language Processing (NLP), such as BERT. Choosing the model coherently with NLP SA tasks can be improved by getting an efficient programming language and supporting resources library. Numerous studies have been conducted on sentiment analysis over the years, and most studies accurately classify data as positive, neutral, or negative polarity. Despite this, various studies have shown the diverse applications of SA research in security, tourism, and business intelligence. The approach can expand the need to use SA as part of the business strategy to increase revenues [4].

Ktrain is a Python machine learning package library that provides a simple, unified interface for performing the ML programming workflow in building the model regardless of the input type (i.e., text vs. images vs. graphs). It is a lightweight TensorFlow Keras wrapper package that can be useful in developing deep learning and ML projects. The wrapper is used to create, train and deploy deep learning and ML models. There are three phases of developing the project: building a model, inspecting, and applying it that can be completed in as few as three or four lines of code, a technique known as "low-code" ML [2]. The ktrain library package currently supports several data types and tasks: text, vision, graph, and tabular data.

Ktrain also aims to further democratize machine learning by allowing novices and domain experts with minimal programming or data science knowledge to construct advanced machine learning models with

DOI reference number: 10-18293/JVLC2022-N2-024

minimal coding. It is also handy for seasoned practitioners who need to develop deep learning solutions rapidly[2].

A deep learning transformer solution like BERT is the first fine-tuning-based representation model that outperforms numerous task-specific designs on many sentence and token-level challenges. As a state-of-art model, BERT was designed to pre-train deep bidirectional representation from an unlabeled text by conditioning each layer on the left and right context. BERT can be fine-tuned to various task questions and language inference in NLP without significant task-specific architecture modifications. BERT is advanced in eleven NLP tasks and distinctive features unified are architectured across different tasks [5].

The hyperparameters setting using the ktrain library used a simple lines code to describe what batch size to be used and what learning rate parameter to specify. The ktrain-BERT has a pre-training length of 75 words, a full feature set to consider, and a batch size of words to embed. It features a three and n-gram range for word token sequencing. Model loading also establishes column labels so the model can determine if the job is binary or multi-label categorization[6]. Fig 1. shows the codes line to set the training model with one cycle learning rate policy. The fit method will be applied with the cycle_len parameter. The hyperparameters setting using the Ktrain Library used a simple line that described what batch size to use and what learning rate parameter to specify. Figure 1 shows the line of code to set the training model with one cycle learning rate policy. The function will process the learning rate schedule and maximal learning rate reduction.

```
hist = learner.fit_onecycle(1e-5, 2)
```

Figure 1: Method to set one cycle policy rate schedules

Text classification is a common problem in SA. It is distinct from text mining and employs text classification to identify document subjects. SA classifies material depending on the writer's mood on various positive or negative themes [4]. For determining the sentiment of a text, the machine learning approach uses algorithms such as Naive Bayes, S Vector Machine, Decision Tree, Logical Regression, etc. In contrast, the lexicon-based approach relies on sentiment lexicons (i.e., a dictionary of opinion words and phrases with assigned polarities and intensities) [7].

2. Research Objectives

The primary goal of this research is to enhance the model development process using programming languages such as Python by choosing an adequate library package to increase the model's performance. Doing this can make the process more efficient with less code and improve performance on processing data to meet challenged NLP tasks in SA. The scope of work for this research is to fulfill the following objectives :

(i) Enhance the capability of a model building using the library packages to accommodate the need for NLP tasks, particularly on SA domain datasets.

(ii) Better managed the language program tools using transformer BERT correlated with the python package using ktrain library.

The works of literature related to the ktrain Library, BERT pre-trained Model, ML model workflow, and buildings of the SA model in these subsections are as follow:

2.1 BERT Model

The BERT language model has two approaches: feature-based, unsupervised, and fine-tuning. Feature-based unsupervised techniques, in general, are like word or word embedding [8]–[10]. The process to train the word representation based on previous work was to place the possibilities of the following sentence [9], the generation of left-to-right emphasis words of the last sentence [8], and the derivative of automatic denoising encoding [11]. The unsupervised approach based on fine-tuning, on the other hand, works on word embedding parameters trained from the unlabeled text. Sentence or document encoders that produce contextual token representations have been trained from unlabeled text and refined for supervised downstream tasks [12]–[14]. Figure 2 shows the pre-training and fine-tuning for the BERT model in which the output architecture is the same. In pre-training, the BERT model will use the masked sentence to train the model on how to predict the surrounding words[5]. As for fine-tuning, the model can be configured to apply a feature selection algorithm using hybrid approaches to building the sentiment analysis model.

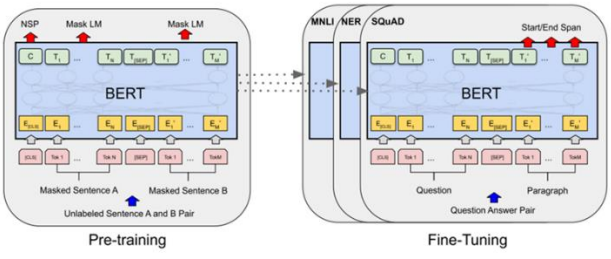

Figure 2: BERT model Pre-training and Fine-Tuning [5]

The machine learning model development consists of 3 phases: model development, inspection, and

development [15]. Recent improvements in deep learning-based recommender systems have piqued researchers' interest by overcoming traditional models' limitations and reaching excellent suggestion quality. Deep learning can capture non-linear and non-trivial user-item connections and codify more sophisticated abstractions as data representations in higher layers. Furthermore, it captures the subtle relationships within the data from readily available sources like contextual, textual, and visual information. The BERT model can semantically pick up a relevant field associated with documents classified in scientific publications[16].

2.2 Ktrain Library

Ktrain is a low code library for Python that can make accessibility and application to ML tasks easier. It is straightforward for beginners and expert practitioners to build, train, inspect, and apply sophisticated, state-of-the-art machine learning models. Ktrain is also designed to make deep learning, such as BERT and Artificial Intelligent (AI), more accessible and easier for newcomers and experienced practitioners. It also includes modules that deal with textual data. Ktrain is used in the machine learning model implemented in TensorFlow Keras (TF. Keras) and supports the following data types and tasks, as illustrated in Fig. 3[2]. Four categories for the ktrain library are text, vision, graph, and tabular data. This category will relate the ktrain library with a classifying model for performing the relevant task. Tasks for text data include task classification, such as auto categorizing documents, text regression, such as predicting numerical values, and sequence tagging, which extracts sequences of words representing Named Entity Recognition. The tasks on vision data include image classification, such as auto-categorizing images across various dimensions, and image regression to predict numerical values such as a person's age from photos. The graph data types category is a node classification that auto-categorizes nodes in a graph, such as social media accounts, and predicts missing links in a social network. Lastly, tabular data includes classification and regression that store data in tables.

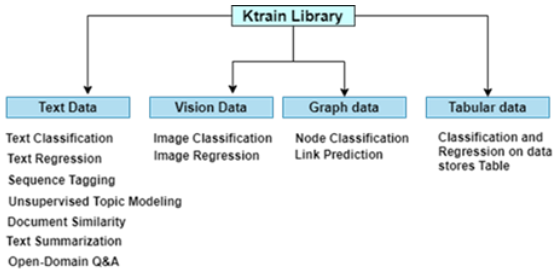

Figure 3: Ktrain data types and task [2]

Additionally, ktrain offers a straightforward and user-friendly prediction API for making predictions on fresh and unstudied samples. The model (i.e., the underlying tf. Keras model) and the preprocessing procedures (i.e., a Preprocessor instance) necessary to convert raw data into the format anticipated by the model are both included in a Predictor instance. It is simple to save and reload the Predictor instance to deploy production environments [2].

Ktrain simplifies the SA model workflow, from input curation and preprocessing (i.e., ground-truth-labeled training data) to model training, tuning, troubleshooting, and application. Libraries that are a core of Ktrain's explainable AI are powered by other libraries, such as Shapley Additive Explanations (SHAP) (Kiros et al. 2015) and ELI5 with LIME [9]. The prediction using both libraries significantly impacted the result by trusting a prediction or testing the model. We characterize "explaining a prediction" as displaying textual or visual artifacts that provide a qualitative comprehension of the relationship between the instance's components (e.g., words in a text, patches in a picture) and the model's prediction [9]. The SHAP library will assign each feature an essential value for a particular prediction. Hence, the local Interpretable Model-agnostic Explanations (LIME) technique interpreted individual model predictions based on a local approximation model around a given prediction [15].

The equation prediction using SHAP by additive feature attribution methods is :

$$g(z') = \emptyset_0 + \sum_{i=1}^{M} \emptyset_i \; z_i' \qquad (1)$$

Where $z' \in \{0,1\}^M$, M is the number of simplified input features, and $\emptyset_i \in \mathbb{R}$.

This method matches the attribute with the effect of $\emptyset_i$ to each feature and sums up all the attribute features to the output *f(x)* of the original model.

As for using LIME, the equation prediction interprets the model predictions individually based on approximating the model for the given prediction. It is to simplify input by mapping the value of $x = h_x(x')$ to convert it into a binary vector of interpretable input to the original input space.

$$\xi = \arg\min L(f, g, \pi_{x'}) + \Omega(g) \qquad (2)$$

The explanation of the argument is the minimum ξ =arg min, and it will enforce the loss of L over a set of samples in simplified input space weighted by the local kernel $\pi_{x'}$.Ω penalizes the complexity of g. The equation is solved using penalized linear regression [15].

Figure 4 is a sentiment analysis pipeline by applying SHAP in IMDB positives datasets for two rows of data. It shows how to score positive based on the reviewer's dataset comment.

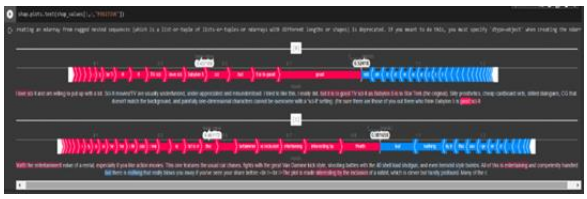

Figure 4: SHAP explained in an IMDB positives review on two sets of data

2.3 SA Model

Acquiring SA models and modifying them according to the task requirements is time-consuming and error-prone. It is because process knowledge is typically disseminated across many people, and workflow modeling is an arduous task that requires the expertise of a modeling specialist [17]. There are three fundamental problems in developing the SA method: feature selection, senti-word identification, and sentiment classification. This requirement is needed to prepare the developing model to meet SA requirements and focus on the given task. The data validated the sentiment model's integrity by developing new SA frameworks and components under SA-related issues [18]. Building models using the ktrain standard for supervised tasks described are as described in Figure 5 [2], [17].

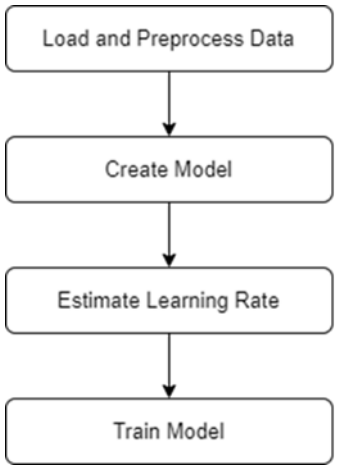

Figure 5: Phases in building model [2]

Phases step based on Figure 5 that will be applied using ktrain library packages :

- Load and preprocess Data comprises a step for loading data from various sources and preparing it according to the model's specifications. The process for language-specific preparation will include lemmatization and tokenization for text classification. Image data may involve auto-normalizing pixel values based on the model, attributes of nodes, and the link to the networks. All ktrain preprocessing techniques return a Preprocessor instance that incorporates all the preprocessing stages for a particular task. This instance can be used when utilizing the model to make predictions on new, unknown data.

- Create a Model using tf.Keras for model development and be wrapped in a ktrain. The model will be configured based on configuring the Learner instance wrapped in ktrain. Learner instance as an abstraction to facilitate training. The customization also can be made by the users whether to customize it or use defaults.

- Estimate Learning Rate can be employed to estimate the optimal learning rate for the given model and data. This step is optional for some models, such as BERT, with default learning rates that function well.

- The training model will be using the fit-one-cycle method and autofit method. Fit-one-cycle is the method that employs a one-cycle policy, whereas autofit utilizes a triangular learning rate schedule by specifying the number of epochs. After training a model, ktrain provides a simple interface to evaluate the technique and calculates extensive validation (or test) metrics.

3. Methodology

The proposed methodology using the ktrain Library and BERT model are as per Figure 6. The building frameworks for this methodology will be conducted in various steps starting from initiating the library package for ktrain library resources, processing the data, and splitting data for training and evaluation models. We will use ktrain to easily and quickly build, train, inspect, and evaluate the model. The visualization of the experience will be assessed based on a comparison will baseline models and to produce the result based on the finding.

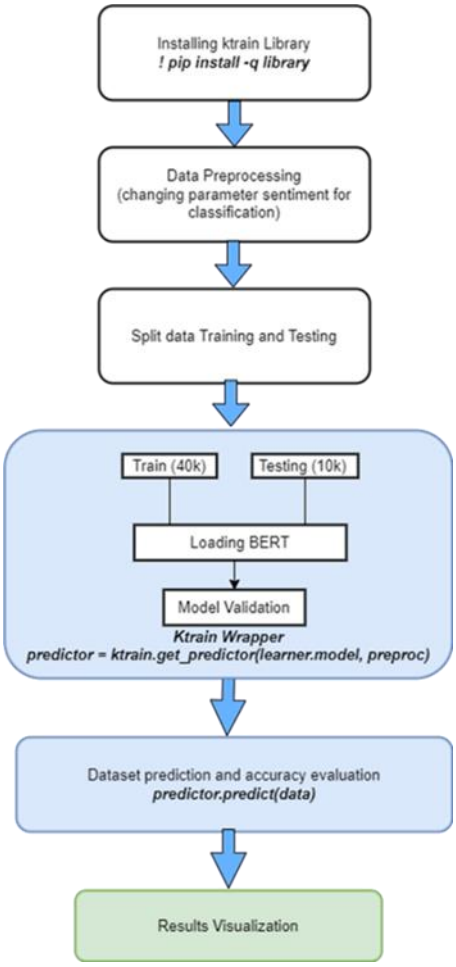

Figure 6: Proposed Methodology Using the Ktrain Library and BERT Model

Firstly, the building of the model BERT will be using the ktrain library by implementing based on hyperparameter tuning on the model structure. Firstly the pre-trained BERT will determine the column labels so that the model can know whether the task is binary or multi-label classification. The transformers class using the ktrain library will be instantiated by providing the model name, the length sequence, and the target names' populating classes. The training metrics were fit to 2 epochs using the one-cycle policy with a max LR of 1e-05. Total trainable parameters are 109,336 322.

Next is the preprocessing data in the Keras model, where the preprocessor instances are loaded with the encapsulated feature set, pre-training length, and model range. The preprocessing steps include tokenization, normalization, and stop word removal [19].

After that, we create a classifier with pre-trained weights, and fine-tuneable final layers initialized randomly. The model will be wrapped in a ktrain Learner object, which will make it simple to train, analyze, and use the model to generate predictions on new data. Dataset will be split into training and testing datasets in a ratio of 40,000 data to trained and 10,000 for testing. We will use the Learning Rate Finder in ktrain to estimate a good learning rate for our model and dataset. Then the model will perform sentiment classification by using the learner.validate() method to produce testing data results (learner.validate(val_data=(x_test, y_test)).

The following method is to inspect the model by examining the classifying datasets. This process includes loading the BERT model using the ktrain wrapper by implementing the function that predicts the classifying sentiment by declaring operating predictor=ktrain.get_predictor(learner. model, preproc).

Lastly, the prediction of the dataset and evaluation for accuracy will be using the predictor.predict(data) function. The result will be visualization by evaluating the accuracy, precision, recall, and F1 score. The visualization will illustrate the comparison with other baseline models, such as SVM and LR.

4. Results and Discussion

The experiment was conducted on the Google Colab platform with GPU using an IMDB movie review [20] with total data of 50,000 lines data. Dataset was split into 40,000 for the training dataset and 10,000 for the test dataset. It is a balanced dataset with polarity negatives and positives. The experiment reviewed the ktrain lightweight wrapper for TensorFlow Keras, which streamlines the development of the model transformer BERT. The execution of model activities through the ktrain wrapper will use the library's resources that initialize from the ktrain library. The task was done with minimal program code to initialize the

BERT model classifier. This function will process the splitting data process with eight batches based on the parts of program codes shown in Figure 7.

```
model = text.text_classifier('bert',
                    train_data=(x_train, y_train),
                    preproc=preproc)

learner = ktrain.get_learner(model,
                    train_data=(x_train, y_train),
                    batch_size=8)
```

Figure 7: Line Codes In Training The Model Using The Ktrain Library

The classifier model used the method of ktrain.get_predictor to evaluate data from the learner model to predict the sentiment outcome using predictor instances predictor.predict(data). The predictor can be implemented on individual phrases of data to check the sentiment classifier outcome. The method of predictor object is to help the model understand how those predictions were made. Figure 8 is the result based on the unigram classifier. The Unigram word is 'bad,' and the result gives 0 negative value.

```
predictor =
ktrain.get_predictor(learner.model, preproc)
data=["bad"]
predictor.predict(data)
```

Figure 8: Results based on unigram

The calculation was based model classifier for True Positive, False Negative, True Negative, and False Positive. Table 1 shows the results based on learner.validate() method for the testing of 10,000 data (0.2 from datasets of 50,000 with 40,000 for training dataset). The results are based on classification Table I, which then shows the detailed results compared with the baseline models.

Table 1. Evaluation results

Model	True Positive	False negative	True Negative	False Positive
BERT	4612	339	4676	373

Details calculations are based on the measurement equation below [21]:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} => \frac{9288}{10000}$$

$$=> \mathbf{0.928}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} => \frac{4612}{4612 + 373}$$

$$=> \frac{4612}{4985} => \mathbf{0.925}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} => \frac{4612}{4612 + 339}$$

$$=> \frac{4612}{4951} => \mathbf{0.931}$$

$$\text{F1 Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$=> 2 * \frac{0.861}{1.856} => \mathbf{0.927}$$

Compared with the baseline model summarized, model BERT gets a higher accuracy of 92.8 compared to other baseline models using Support Vector Machine(SVM) and Logistic Regression(LR). For precision, BERT obtained 92.5%, Recall 93.1% dan f1-score 92.7%. This result shows that the BERT model was better than SVM and LR. Detail as described in Table 2.

Table 2. Comparison of results with baseline models

	Accuracy (%)	*Precision* (%)	*Recall* (%)	*F1-Score* (%)
BERT	**92.8**	**92.5**	**93.1**	**92.7**
LR	88.90	88.90	90.80	88.90
SVM	89.80	89.20	90.80	90.0

The discussion on the experimental process is an initialization of the ktrain library that provides more options for the general sentiment analysis process. Ktrain package and sub-modules comprehensively cover most of the tasks in classifying the SA requirements. The experiment reviewed the ktrain lightweight wrapper for TensorFlow Keras, which streamlines the development of transformer models such BERT model.

The prediction instances show that making a predictor of the current data only needs simple code to call the

predictor instances, declare it based on the data set, and thus, predict the data polarity. The predictions can be easily defined by utilizing the method predictor objects to understand how the prediction is made. An experimental sample using the unigram n bigram shows that the predictor was accurate based on the predicted results. Figure 9 shows the ktrain library's predictions using the BERT model based on bigram phrases.

```
# calling instances
predictor = ktrain.get_predictor(learner.model, preproc)
#line of data set
data=["bad movie"]
# predict the data
predictor.predict(data)
#Results 0 – negative 1 - positive
['0']
```

Figure 9: Make predictions using the ktrain library and BERT model

Once all models are trained, they can be managed using TensorFlow or the transformer Library. The predictor also can be saved than to be reloaded later in making other deployment classifications by applying the predictor. save method.

Additionally, the study has shown that the task can be done with minimal program code and demonstrated how to do the tasks with less program code. This post explained how to execute most modeling activities entirely through the wrapper. Developing ktrain Library to BERT classifier also used a few line codes for examples method such learner.validate. That will produce the result evaluation that uses a validation set by default, as shown in Figure 10.

```
learner.validate(val_data=(x_test, y_test))
              precision    recall  f1-score   support

           0       0.92      0.93      0.92      5049
           1       0.91      0.92      0.91      4951

    accuracy                           0.92     10000
   macro avg       0.92      0.92      0.91     10000
weighted avg       0.91      0.92      0.91     10000
```

Figure 10: Evaluation Results of the BERT Model

The comparison within the finding of True positives (TP), True Negatives (TN), False positives (FP), and False Negatives (FN) are illustrated in Figure 11. The result shows that the predicted TP and TN were high, and FP and FN were low, resulting in better accuracy.

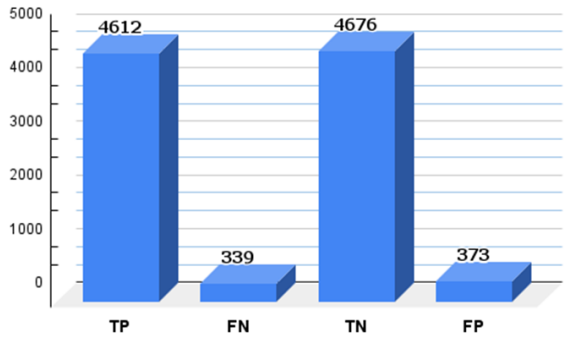

Figure 11. Evaluation results

The comparison of the BERT model to other baseline models resulted in Figure 12. The results show that the BERT classifier results better than other SA models like LR and SVM. The predictions of TP dan TN showed a higher true result. FN and FP also showed lower false results.

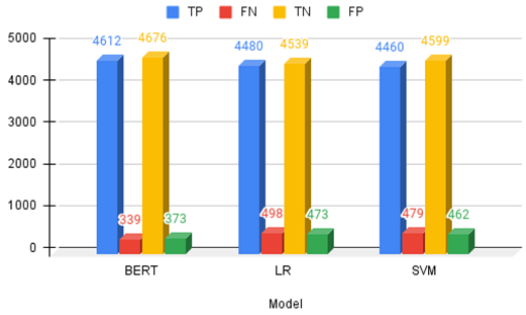

Figure 12: Comparison with baseline models

5. Conclusion

This paper shows that the BERT model works better with the ktrain library resources modules. It successfully processed the function or method to expedite programming capability in delivering results. The proposed model achieved a better result with an accuracy of 92.8%, a precision of 92.5%, Recall 93.1%, and F1-Score with 92.7% compared to Support Vector Machine and Logistics Regression. Additionally, it demonstrated that the library provided by ktrain requires less code and can produce the required task based on NLP needs. Nevertheless, further research needs to work on the limitations. For future work, we advocate wrapping the ktrain library to more pre-trained models and testing them on diverse domains of datasets.

Acknowledgement

The authors gratefully acknowledged Universiti Kebangsaan Malaysia and RHB Bank through grant code RHB-UKM-2020-001 (Geran RHB-UKM) for supporting this research. The authors also thank the Public Service Department of Malaysia(PSD) and the Malaysian Administrative Modernisation and Management Planning Unit (MAMPU) for this research opportunity.

This material was made initially and has never been published elsewhere.

References

[1] W. Kaur, V. Balakrishnan, and B. Singh, "Improving teaching and learning experience in engineering education using sentiment analysis techniques," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 834, no. 1, 2020, doi: 10.1088/1757-899X/834/1/012026.

[2] A. S. Maiya, "ktrain: A Low-Code Library for Augmented Machine Learning," vol. 10703, 2020, [Online]. Available: http://arxiv.org/abs/2004.10703.

[3] P. Ksieniewicz and P. Zyblewski, "Stream-learn — open-source Python library for difficult data stream batch analysis," *Neurocomputing*, vol. 478, pp. 11–21, 2022, doi: 10.1016/j.neucom.2021.10.120.

[4] I. S. Ahmad, A. Abu Bakar, M. R. Yaakub, and M. Darwich, "Beyond sentiment classification: A novel approach for utilizing social media data for business intelligence," *Int. J. Adv. Comput. Sci. Appl.*, vol. 11, no. 3, pp. 437–441, 2020, doi: 10.14569/ijacsa.2020.0110355.

[5] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *NAACL HLT 2019 - 2019 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. - Proc. Conf.*, vol. 1, pp. 4171–4186, 2019.

[6] S. Das, P. Mandal, and S. Chatterji, "Probabilistic Impact Score Generation using Ktrain-BERT to Identify Hate Words from Twitter Discussions," 2021.

[7] W. Kaur and V. Balakrishnan, "Improving sentiment scoring mechanism: a case study on airline services," *Ind. Manag. Data Syst.*, vol. 118, no. 8, pp. 1578–1596, 2018, doi: 10.1108/IMDS-07-2017-0300.

[8] R. Kiros *et al.*, "Skip-thought vectors," *Adv. Neural Inf. Process. Syst.*, vol. 2015-Janua, no. 786, pp. 3294–3302, 2015.

[9] L. Logeswaran and H. Lee, "An efficient framework for learning sentence representations," *6th Int. Conf. Learn. Represent. ICLR 2018 - Conf. Track Proc.*, pp. 1–16, 2018.

[10] C. Republic and T. Mikolov, "Statistical Language Models Based on Neural Networks," *Wall Str. J.*, no. April, pp. 1–129, 2012, doi: 10.1016/j.csl.2015.07.001.

[11] F. Hill, K. Cho, and A. Korhonen, "Learning distributed representations of sentences from unlabelled data," *2016 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. NAACL HLT 2016 - Proc. Conf.*, pp. 1367–1377, 2016, doi: 10.18653/v1/n16-1162.

[12] A. M. Dai and Q. V. Le, "Semi-supervised sequence learning," *Adv. Neural Inf. Process. Syst.*, vol. 2015-Janua, pp. 3079–3087, 2015.

[13] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," *ACL 2018 - 56th Annu. Meet. Assoc. Comput. Linguist. Proc. Conf. (Long Pap.*, vol. 1, pp. 328–339, 2018, doi: 10.18653/v1/p18-1031.

[14] A. Radford, K. Narasimhan, T. Salimans, and I. Sustskever, "Improving Language Understanding by Generative Pre-Training," *Comput. Sci.*, vol. 9, no. 1, 2018.

[15] S. M. Lundberg and S. I. Lee, "A unified approach to interpreting model predictions," *Adv. Neural Inf. Process. Syst.*, vol. 2017-Decem, no. Section 2, pp. 4766–4775, 2017.

[16] A. Garcia-Silva and J. M. Gomez-Perez, "Classifying Scientific Publications with BERT - Is Self-attention a Feature Selection Method?," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 12656 LNCS, pp. 161–175, 2021, doi: 10.1007/978-3-030-72113-8_11.

[17] J. Herbst, "A machine learning approach to workflow management," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 1810, pp. 183–194, 2000, doi: 10.1007/3-540-45164-1_19.

[18] S. R. Ahmad, A. A. Bakar, and M. R. Yaakub, "A review of feature selection techniques in sentiment analysis," *Intell. Data Anal.*, vol. 23, no. 1, pp. 159–189, 2019, doi: 10.3233/IDA-173763.

[19] N. Omar, M. Albared, T. Al-Moslmi, and A. Al-Shabi, "A comparative study of feature selection and machine learning algorithms for arabic sentiment classification," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8870, pp. 429–443, 2014, doi: 10.1007/978-3-319-12844-3_37.

[20] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," *ACL-HLT 2011 - Proc. 49th Annu. Meet. Assoc. Comput. Linguist. Hum. Lang. Technol.*, vol. 1, pp. 142–150, 2011.

[21] M. Hasnain, M. F. Pasha, I. Ghani, M. Imran,

M. Y. Alzahrani, and R. Budiarto, "Evaluating Trust Prediction and Confusion Matrix Measures for Web Services Ranking," *IEEE Access*, vol. 8, pp. 90847–90861, 2020, doi: 10.1109/ACCESS.2020.2994222.