AutoText: An End-to-End AutoAI Framework for Text

Arunima Chaudhary, Alayt Issak*, Kiran Kate, Yannis Katsis, Abel Valente, Dakuo Wang, Alexandre Evfimievski, Sairam Gurajada, Ban Kawas, Cristiano Malossi, Lucian Popa, Tejaswini Pedapati, Horst Samulowitz, Martin Wistuba, Yunyao Li

IBM Research AI

{arunima.chaudhary, yannis.katsis, dakuo.wang, sairam.gurajada, martin.wistuba}@ibm.com, aissak21@wooster.edu, valente@ar.ibm.com, acm@zurich.ibm.com, {kakate, evfimi, bkawas, lpopa, tejaswinip, samulowitz, yunyaoli}@us.ibm.com

Abstract

Building models for natural language processing (NLP) tasks remains a daunting task for many, requiring significant technical expertise, efforts, and resources. In this demonstration, we present AutoText, an end-to-end AutoAI framework for text, to lower the barrier of entry in building NLP models. AutoText combines state-of-the-art AutoAI optimization techniques and learning algorithms for NLP tasks into a single extensible framework. Through its simple, yet powerful UI, non-AI experts (e.g., domain experts) can quickly generate performant NLP models with support to both control (e.g., via specifying constraints) and understand learned models.

Introduction

Recent AI advances led to rising desire of businesses and organizations to apply AI to solve their problems (Mao et al. 2019). However, developing performant AI models remains a tedious iterative process that includes data preparation, feature engineering, algorithm/model architecture selection, hyperparameter tuning, training, model evaluation, and comparison to other models. To lower the barrier of entry in AI, the research community has looked into automating this process with *AutoML* (Hutter, Kotthoff, and Vanschoren 2019) or *AutoAI* (Wang et al. 2019, 2020).

We present *AutoText*, an extensible end-to-end AutoAI framework, to democratize the development of Natural Language Processing (NLP) models. Its design considerations include: (a) **Usability**: Non-experts can create NLP models (referred to as *pipelines*) through a GUI, while expert users can fully control the process via a Notebook UI[1]. (b) **Extensibility**: AutoText employs an extensible architecture, where both the components of the pipelines and the pipeline discovery (known as optimization) algorithms are modeled as exchangeable modules. (c) **Comprehensiveness**: AutoText offers a comprehensive approach, combining various learning algorithms (incl. classical machine learning and deep learning approaches) and optimization approaches (incl. algorithm selection, hyperparameter tuning, and neural architecture search) into a single framework. (d) **High Performance:** AutoText is able to learn performant models that, in many cases, are comparable or outperform state-of-the-art models for the corresponding tasks/datasets. A video demonstration of AutoText can be found online[2].

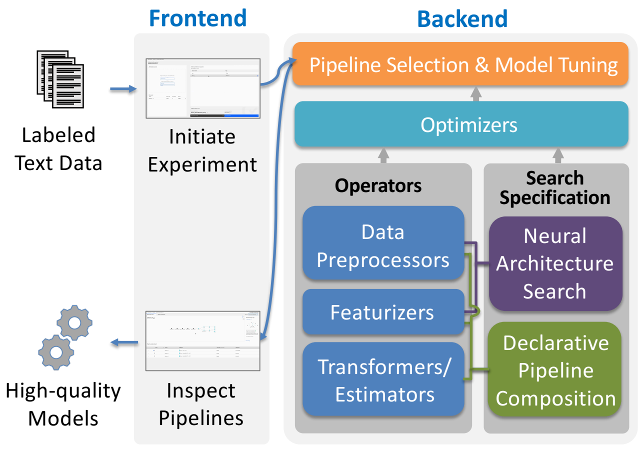

Figure 1: AutoText Architecture

Related Work. Despite increasing popularity of AutoAI systems, we are not aware of any with the comprehensive support for text offered by AutoText. Google AutoML Natural Language[3] does not allow users to select model types, or give insights into the model produced nor into its tuning and hyper-parameter optimization (Weidele et al. 2020). Amazon SageMaker Autopilot[4] offers AutoML capabilities that can be applied on text, but does not provide intuitive visualizations of pipelines and their performance; it also lacks more advanced capabilities for neural architecture search (NAS). Other systems, such as Microsoft AzureML-BERT[5], focus on high-performance model building with BERT pre-training and fine-tuning, but do not offer simpler but faster models. H20[6] is perhaps the closest AutoML framework to ours; however, it supports a limited range of learning al-

*Work done at IBM Research; now at the College of Wooster.

[1]This demo focuses on AutoText's GUI-based interaction.

[2]https://youtu.be/tujB0BrYlBw

[3]https://cloud.google.com/natural-language/automl

[4]https://aws.amazon.com/sagemaker/autopilot/

[5]https://azure.microsoft.com/en-us/services/machine-learning

[6]https://www.h2o.ai/products/h2o-automl

gorithms (e.g., RF, GBM, GLM for classical ML and only MLP for deep learning), a small pipeline search space (only ensembles), and limited features for NLP (Word2Vec). Finally, while NAS is becoming an essential part of AutoML (He, Zhao, and Chu 2019), few systems focus on NLP; moreover, none combine NAS with exploration of alternative pipelines that may include classical ML algorithms.

The AutoText Framework

AutoText comprises a *frontend* for easy interaction and a *backend* for automated model building. We now describe each of them in detail.

Frontend

End users can initiate the building of NLP models by uploading a labeled textual dataset and selecting the target column. AutoText's backend handles the rest of the process, including identifying the task type[7] and suitable evaluation metric, splitting the dataset into train/test, and discovering pipelines. The best performing pipelines are then shown to the user to inspect. This includes both a graphical representation of each pipeline illustrating its components and a tabular leaderboard of pipelines sorted by performance.

Customizations. AutoText allows users to customize through its UI many aspects of the framework, including the types of learning algorithms explored, the number of pipelines returned, as well as constraints, such as a time budget for pipeline discovery. More advanced users can also use its Notebook interface to impose additional controls (e.g., override the default search space for individual operators).

Backend

The backend of AutoText includes: (1) *Operator library:* a set of operators that form basic components of typical NLP pipelines, (2) *Search specification,* defining the pipeline search space, (3) *Optimizers:* a set of methods to perform joint algorithm search and hyperparameter optimization in the defined search space, and (4) *Pipeline selection and model tuning:* the main component that picks the appropriate optimizer and coordinates the pipeline discovery process.

Operator Library. AutoText employs an extensible operator library, with support for three main operator classes:

- *Data Preprocessors* – operators that clean and/or transform data to prepare for later processing. These include tokenization, lemmatization, part-of-speech (POS) tagging, normalization, simplification, spell correction, etc.
- *Featurizers* – non-trainable (or pretrained) operators that compute features for use by downstream operators. These include both *classical* features, such as tf-idf, and *deep learning-based* features, such as embeddings generated by GloVe, Fast-Text, BERT, RoBERTa, and others.
- *Transformers/Estimators* – trainable operators that transform data to a new feature space or make final predictions. These include *classical ML* operators (TfIdfTransformer, SVM, XGBoost, etc.) and *deep learning-based* operators (BiLSTM, CNN, BERT, XLNet, DistilBERT, MLP, etc.).

The library can be easily extended with new operators to further increase the coverage of supported techniques.

Search Specification. AutoText supports two ways to define the pipeline search space: (1) *Declarative Pipeline Composition*: Leveraging the declarative LALE framework (Baudart et al. 2020), AutoText developers can write high-level specifications of the search space for each pipeline operator, as well as operator composition rules. (2) *Neural architecture search (NAS)* enables automatic pipeline composition for deep learning models.

Optimizers. AutoText combines into a single framework multiple optimization techniques presented in the literature, including combined algorithm selection and hyperparameter tuning (CASH) (Thornton et al. 2013) and NAS (Wistuba, Rawat, and Pedapati 2019). This is done by integrating multiple optimizers, each suited to different scenarios, selecting between them based on heuristics. These include Hyperopt (Bergstra et al. 2015), Hyperband (Li et al. 2017), ADMM (CASH optimizer supporting black-box constraints, such as prediction latency or memory footprint) (Liu et al. 2020), TAPAS (Istrate et al. 2019), and NeuRex (Wistuba 2018).

Pipeline Selection and Model Tuning. Given a labeled dataset and optional user options, AutoText employs the appropriate optimizer to iteratively explore different pipeline configurations. During this process, the optimizer considers the search specification, potential user constraints, and the performance of previously explored configurations to determine the next pipeline to explore. This process continues until the user-specified constraints are satisfied or after a predetermined number of iterations. The resulting pipelines are then shown on the frontend for the user to inspect.

Demonstration

We will demonstrate AutoText through a variety of datasets, including standard benchmarks, such as MR (Pang and Lee 2005) and MPQA (Wiebe, Wilson, and Cardie 2005), and additional data, such as the US Consumer Financial Complaints dataset[8]. During the demo the audience will be able to gain the following insights: (a) experience how AutoText allows the quick generation of AI models, (b) understand common components of NLP models and find out which model architectures work best for particular tasks, by inspecting the discovered pipelines, and (c) explore trade-offs made by different estimators (such as the training time/performance trade-off of classical ML vs deep learning algorithms (Li et al. 2020)) by customizing the experiment to only consider specific algorithms. Finally, the audience will also become familiar with AutoText's architecture and understand how different AutoAI techniques presented in the literature have been incorporated into a unified end-to-end AutoAI framework capable of generating performant NLP models.

[7]AutoText currently supports binary and multi-class classification and will be extended in the future with additional NLP tasks, such as entity extraction, relationship extraction, and others.

[8]https://www.kaggle.com/cfpb/us-consumer-finance-complaints

References

Baudart, G.; Hirzel, M.; Kate, K.; Ram, P.; and Shinnar, A. 2020. Lale: Consistent Automated Machine Learning. In *KDD Workshop on Automation in Machine Learning (AutoML@KDD)*. URL https://arxiv.org/abs/2007.01977.

Bergstra, J.; Komer, B.; Eliasmith, C.; Yamins, D.; and Cox, D. D. 2015. Hyperopt: a Python Library for Model Selection and Hyperparameter Optimization. *Computational Science & Discovery* 8(1). URL http://dx.doi.org/10.1088/1749-4699/8/1/014008.

He, X.; Zhao, K.; and Chu, X. 2019. AutoML: A Survey of the State-of-the-Art. *ArXiv* abs/1908.00709.

Hutter, F.; Kotthoff, L.; and Vanschoren, J. 2019. *Automated machine learning: methods, systems, challenges*. Springer Nature.

Istrate, R.; Scheidegger, F.; Mariani, G.; Nikolopoulos, D. S.; Bekas, C.; and Malossi, A. C. I. 2019. TAPAS: Trainless Accuracy Predictor for Architecture Search. In *AAAI*.

Li, J.; Li, Y.; Wang, X.; and Tan, W.-C. 2020. Deep or Simple Models for Semantic Tagging? It Depends on your Data. *Proceedings of the VLDB Endowment* 13(11).

Li, L.; Jamieson, K.; DeSalvo, G.; Rostamizadeh, A.; and Talwalkar, A. 2017. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research* 18(1): 6765–6816.

Liu, S.; Ram, P.; Vijaykeerthy, D.; Bouneffouf, D.; Bramble, G.; Samulowitz, H.; Wang, D.; Conn, A.; and Gray, A. G. 2020. An ADMM Based Framework for AutoML Pipeline Configuration. In *Conference on Artificial Intelligence (AAAI)*, 4892–4899. URL https://aaai.org/ojs/index.php/AAAI/article/view/5926.

Mao, Y.; Wang, D.; Muller, M.; Varshney, K. R.; Baldini, I.; Dugan, C.; and Mojsilović, A. 2019. How Data Scientists Work Together With Domain Experts in Scientific Collaborations: To Find The Right Answer Or To Ask The Right Question? *Proceedings of the ACM on Human-Computer Interaction* 3(GROUP): 1–23.

Pang, B.; and Lee, L. 2005. Seeing Stars: Exploiting Class Relationships for Sentiment Categorization with Respect to Rating Scales. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, 115–124. Ann Arbor, Michigan: Association for Computational Linguistics. doi:10.3115/1219840.1219855. URL https://www.aclweb.org/anthology/P05-1015.

Thornton, C.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 847–855.

Wang, D.; Ram, P.; Weidele, D. K. I.; Liu, S.; Muller, M.; Weisz, J. D.; Valente, A.; Chaudhary, A.; Torres, D.; Samulowitz, H.; et al. 2020. AutoAI: Automating the End-to-End AI Lifecycle with Humans-in-the-Loop. In *Proceedings of the 25th International Conference on Intelligent User Interfaces Companion*, 77–78.

Wang, D.; Weisz, J. D.; Muller, M.; Ram, P.; Geyer, W.; Dugan, C.; Tausczik, Y.; Samulowitz, H.; and Gray, A. 2019. Human-AI Collaboration in Data Science: Exploring Data Scientists' Perceptions of Automated AI. *Proceedings of the ACM on Human-Computer Interaction* 3(CSCW): 1–24.

Weidele, D. K. I.; Weisz, J. D.; Oduor, E.; Muller, M.; Andres, J.; Gray, A.; and Wang, D. 2020. AutoAIViz: opening the blackbox of automated artificial intelligence with conditional parallel coordinates. In *Proceedings of the 25th International Conference on Intelligent User Interfaces*, 308–312.

Wiebe, J.; Wilson, T.; and Cardie, C. 2005. Annotating expressions of opinions and emotions in language. *Language resources and evaluation* 39(2-3): 165–210.

Wistuba, M. 2018. Deep Learning Architecture Search by Neuro-Cell-Based Evolution with Function-Preserving Mutations. In *ECML/PKDD*.

Wistuba, M.; Rawat, A.; and Pedapati, T. 2019. A Survey on Neural Architecture Search. *CoRR* abs/1905.01392. URL http://arxiv.org/abs/1905.01392.