Journal of Machine Learning Research 23 (2022) 1-6 Submitted 9/21; Revised 4/22; Published 5/22

ktrain: A Low-Code Library for Augmented Machine Learning

Arun S. Maiya AMAIYA@IDA.ORG
Institute for Defense Analyses
Alexandria, VA, USA

Editor: Joaquin Vanschoren

Abstract

We present **ktrain**, a low-code Python library that makes machine learning more accessible and easier to apply. As a wrapper to TensorFlow and many other libraries (*e.g.,* `transformers`, `scikit-learn`, `stellargraph`), it is designed to make sophisticated, state-of-the-art machine learning models simple to build, train, inspect, and apply by both beginners and experienced practitioners. Featuring modules that support `text` data (*e.g.,* text classification, sequence tagging, open-domain question-answering), `vision` data (*e.g.,* image classification), `graph` data (*e.g.,* node classification, link prediction), and `tabular` data, **ktrain** presents a simple unified interface enabling one to quickly solve a wide range of tasks in as little as three or four "commands" or lines of code.

Keywords: low-code machine learning, nlp, computer vision, graphs, tabular data

1. Introduction

Machine learning workflows can be quite involved and challenging for newcomers to master. Consider the following steps.

1) Model-Building. The training data may reside in a number of different formats from files in folders to CSVs or *pandas* dataframes. If the data is large, it must be wrapped in a generator. Data must be preprocessed in specific ways depending on different factors such as the language of training texts (*e.g.,* English vs. Chinese) and whether or not transfer learning is being employed. Learning rates, learning rate schedules, number of epochs, weight decay, and many other hyperparameters and settings must be selected or implemented.

2) Model-Inspection. Once trained, a model is inspected in terms of both its successes and failures. This may include classification reports on validation performance, easily identifying examples that the model is getting the most wrong, and Explainable AI methods to understand why mistakes were made.

3) Model-Application. Both the model and the potentially complex set of steps required to preprocess raw data into the format expected by the model must be easily saved, transferred to, and executed on new data in a production environment.

ktrain is a Python library for machine learning with the goal of presenting a simple, unified interface to easily perform the above steps regardless of the type of data (*e.g.,* text vs. images vs. graphs). Moreover, each of the three steps above can be accomplished in as little as three or four lines of code, which we refer to as “low-code” machine learning. **ktrain** can be used with any model implemented in TensorFlow Keras (`tf.keras`).

Unlike automatic machine learning (AutoML) solutions that place a strong emphasis on automating subsets of the model-building process such as architecture search (*e.g.,* He et al. (2019)), **ktrain** instead focuses on either partially or fully automating other aspects of the machine learning (ML) workflow such as data-preprocessing and human-in-the-loop model tuning and inspection. Following inspiration from a blog post[1] by Rachel Thomas of `fast.ai` (Howard and Gugger, 2020), we refer to this as *Augmented Machine Learning.*

Many supported tasks in **ktrain** allow users to either choose from a menu of state-of-the-art models or employ a custom model. With respect to text classification, for example, available models include cutting-edge Transformer models like BERT (Devlin et al., 2018; Wolf et al., 2019) in addition to fast models such as fastText (Joulin et al., 2016) and NBSVM (Wang and Manning, 2012) that are amenable to being trained on a standard laptop CPU. Other features include a learning-rate-finder to estimate an optimal learning rate (Smith, 2018), easy-to-access learning rate schedules like the **1cycle policy** (Smith, 2018) and Stochastic Gradient Descent with Restarts (SGDR) (Loshchilov and Hutter, 2016), state-of-the-art optimizers like AdamW (Loshchilov and Hutter, 2017), ability to easily inspect classifications through Explainable AI and other methods, and a simple prediction API for use in deployment scenarios. **ktrain** is also bundled with pretrained, ready-to-use NER models for English, Chinese, and Russian. **ktrain** is open-source, free to use under a permissive Apache license, and available on GitHub at: `https://github.com/amaiya/ktrain`.

2. Building Models

Supervised learning tasks in **ktrain** follow a standard, easy-to-use template.

STEP 1: Load and Preprocess Data. This step involves loading data from different sources and preprocessing it in a way that is expected by the model. In the case of text, this may involve language-specific preprocessing (*e.g.,* tokenization). In the case of images, this may involve auto-normalizing pixel values in a way that a chosen model expects. In the case of graphs, this may involve compiling attributes of nodes and links in the network (Data61, 2018). All preprocessing methods in **ktrain** return a `Preprocessor` instance that encapsulates all the preprocessing steps for a particular task, which can be employed when using the model to make predictions on new, unseen data.

STEP 2: Create Model. Users can create and customize their own model using `tf.keras` or select a pre-canned model with well-chosen defaults (*e.g.,* pretrained BERT text classifier (Devlin et al., 2018), models for sequence tagging (Lample et al., 2016), pretrained Residual Networks (He et al., 2015) for image classification). In the latter case, the model is automatically configured by inspecting the data (*e.g.,* number of classes, multilabel vs. multi-classification). At this stage, both the model and the datasets are wrapped in a `ktrain.Learner` instance, which is an abstraction to facilitate training.

1. `https://www.fast.ai/2018/07/16/auto-ml2/`

STEP 3: Estimate Learning Rate. Users can employ the use of a learning rate range test (Smith, 2018) to estimate the optimal learning rate given the model and data. Some models like BERT have default learning rates that work well, so this step is optional.

STEP 4: Train Model. The **ktrain** package allows one to easily try different learning rate schedules. For instance, the `fit_onecycle` method employs a 1cycle policy (Smith, 2018). The `autofit` method employs a triangular learning rate schedule (Smith, 2018) with automatic early stopping and reduction of maximal learning rate upon plateau. Thus, specifying the number of epochs is optional in `autofit`. The `fit` method, when supplied with the `cycle_len` parameter, decays the learning rate each cycle using cosine annealing. Users can easily experiment with what works best for a particular problem.

To illustrate ease of use, we provide fully-complete example for text classification. More specifically, we train a Chinese-language sentiment-analyzer on a dataset of hotel reviews.[2]

Fine-Tuning a BERT Text Classifier for Chinese:

```
import ktrain
from ktrain text as txt
# STEP 1: load and preprocess data
trn, val, preproc = txt.texts_from_folder('ChnSentiCorp', maxlen=75,
                                          preprocess_mode='bert')
# STEP 2: load model and wrap in Learner
model = txt.text_classifier('bert', trn, preproc=preproc)
learner = ktrain.get_learner(model,train_data=trn, val_data=val)
# STEP 3: estimate learning rate
learner.lr_find(show_plot=True)
# STEP 4: train model
learner.fit_onecycle(2e-5, 4)
```

Note that there is nothing special we need to do to support Chinese versus other languages like English. The language and character encoding are auto-detected. Moreover, models are configured automatically through data inspection. For instance, the data is automatically analyzed to determine the number of categories, whether or not categories are mutually-exclusive or not, and if targets are numerical or categorical. The model is then auto-configured appropriately. A similar set of steps can be used for a variety of other tasks such as image classification. A unified interface to different and disparate machine learning tasks reduces cognitive load and allows users to focus on more important tasks that may require domain expertise or are less amenable to automation.

3. Non-Supervised ML Tasks

All the examples covered thus far involve *supervised* machine learning. Other tasks such as training *unsupervised* topic models to discover latent themes in document sets or using *pre-trained* NER models follow slightly different steps than those described previously. Despite involving a different pipeline, these non-supervised tasks also employ a low-code API and can be implemented in as little as three lines of code. To illustrate this, we provide a code example for a fully-functional, end-to-end, **open-domain question-answering system**

2. `https://github.com/Tony607/Chinese_sentiment_analysis`

using the well-studied *20 Newsgroups* dataset.[3] We will first load the dataset into a Python list called `docs` using `scikit-learn` (Pedregosa et al., 2011). The basic idea here is to use the document set as a knowledge base that can be issued natural language questions to receive exact answers. In this case, we would like to issue questions about the subject matter buried in the *20 Newsgroups* dataset and receive exact answers. To accomplish this, the following steps are performed:

1. Index documents to a search engine.
2. Use the search index to locate documents that contain words in the question.
3. Extract paragraphs from these documents for use as contexts and use a BERT model pretrained on the SQuAD dataset to parse out candidate answers.
4. Sort and prune candidate answers by confidence scores and returns results.

This entire workflow to build an end-to-end, open-domain question-answering (QA) system can be implemented with a surprisingly minimal amount of code with **ktrain**:

Building an End-to-End Open-Domain QA System in ktrain

```
# build open-domain QA system
from ktrain.text.qa import SimpleQA
SimpleQA.initialize_index('/tmp/myindex')
SimpleQA.index_from_list(docs, '/tmp/myindex', commit_every=len(docs))
qa = SimpleQA('/tmp/myindex')

# ask a question
qa.ask('When did the Cassini probe launch?') # returns "October of 1997"
```

As shown above, upon building the QA system in **only 3 lines of code**, we can submit natural language questions and receive exact answers. In the example shown, we use the `ask` method to submit the question, "**When did the Cassini probe launch?**". The candidate answer with the highest confidence score returned by the `ask` method is the correct answer of "**October of 1997**." Note that, for document sets that are too large to fit into a Python list, one can index documents using `index_from_folder` instead of `index_from_list`. Many additional code examples are available on the GitHub repository.[4]

4. Comparison to Related Libraries

Table 1 compares **ktrain** to popular low-code and AutoML libraries in their out-of-the-box support for a variety of machine learning tasks. Related libraries such as `fastai` (Howard and Gugger, 2020) and `ludwig` (Molino et al., 2019) in addition to AutoML tools like `AutoKeras` (Jin et al., 2019) and `AutoGluon` (Erickson et al., 2020) lack some key "pre-canned" features in **ktrain**, which has the strongest support for natural language processing and graph-based data. Support for additional features is planned for the future.

5. Conclusion

This work presented **ktrain**, a low-code platform for machine learning. **ktrain** currently includes out-of-the-box support for training models on `text`, `vision`, `graph`, and `tabular`

3. `http://archive.ics.uci.edu/ml/datasets/Twenty+Newsgroups`
4. `https://github.com/amaiya/ktrain/tree/master/examples`

Table 1: A comparison of ML tasks supported out-of-the-box in popular low-code and AutoML libraries for tabular, image, audio, text and graph data.

Task	**ktrain**	**fastai**	**Ludwig**	**AutoKeras**	**AutoGluon**
Tabular: Classification/Regression	✓	✓	✓	✓	✓
Tabular: Causal Machine Learning	✓				
Tabular: Time Series Forecasting			✓	✓	
Tabular: Collaborative Filtering		✓			
Image: Classification/Regression	✓	✓	✓	✓	✓
Image: Object Detection	prefitted*	✓			✓
Image: Image Captioning	prefitted*		✓		
Image: Segmentation		✓			
Image: GANs		✓			
Image: Keypoint/Pose Estimation		✓			
Audio: Classification/Regression			✓		
Audio: Speech Transcription	prefitted*		✓		
Text: Classification/Regression	✓	✓	✓	✓	✓
Text: Sequence-Tagging	✓		✓		
Text: Unsupervised Topic Modeling	✓				
Text: Semantic Search	✓				
Text: End-to-End Question-Answering	✓*				
Text: Zero-Shot Learning	✓				
Text: Language Translation	prefitted*		✓		
Text: Summarization	prefitted*		✓		
Text: Text Extraction	✓				
Text: QA-Based Information Extraction	✓*				
Text: Keyphrase Extraction	✓				
Graph: Node Classification	✓				
Graph: Link Prediction	✓				

*Pre-fine-tuned models that can be applied with no training required are denoted as *prefitted*.
Question-answering models can also be applied with no training or may be further fine-tuned.

data. As a simple wrapper to TensorFlow Keras, it is also sufficiently flexible for use with custom models and data formats, as well. Inspired by other low-code (and no-code) open-source ML libraries such as `fastai` (Howard and Gugger, 2020) and `ludwig` (Molino et al., 2019), **ktrain** is intended to help further democratize machine learning by enabling beginners and domain experts with minimal programming or data science experience to build sophisticated machine learning models with minimal coding. It is also a useful toolbox for experienced practitioners needing to rapidly prototype deep learning solutions.

References

CSIRO's Data61. Stellargraph machine learning library. `https://github.com/stellargraph/stellargraph`, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505*, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *arXiv preprint arXiv:1908.00709*, 2019.

Jeremy Howard and Sylvain Gugger. Fastai: A layered api for deep learning. *Information*, 11(2):108, Feb 2020. ISSN 2078-2489. doi: 10.3390/info11020108. URL `http://dx.doi.org/10.3390/info11020108`.

Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. *arXiv preprint arXiv:1806.10282*, 2019.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Piero Molino, Yaroslav Dudin, and Sai Sumanth Miryala. Ludwig: a type-based declarative deep learning toolbox. *arXiv preprint arXiv:1909.07930*, 2019.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Leslie N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018.

Sida Wang and Christopher D. Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2*, ACL '12, page 90–94, USA, 2012. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.