

Automated Dataset Tuning - Hub (ActiveLoop) Proposal

Personal Details

- Name: Daniel Gareev
- Timezone: CEST – Central European Summer Time
- Links: [Github](#), [LinkedIn](#)

Project Abstract

The goal of this project is to create a set of high-level data preprocessing APIs that allow Hub users to improve overall dataset quality with minimal parameters. A machine-learning algorithm may perform differently on datasets with different characteristics (e.g., it might perform better on a dataset with continuous attributes rather than with categorical attributes). Taking into account all the possible pre-processing strategies, there exists an extremely large number of options and non-experienced users become overwhelmed. A lot of research has been done for providing help and an overview of the different steps of data analysis. The focus, however, has usually been on the model training step, and data pre-processing has generally been overlooked. This problem can be addressed by an automated approach, leveraging ideas from multiple research papers.

Project Description

Data preprocessing is a multi-step process of transforming and encoding the raw data so that it may be easily parsed by the machine. Data preprocessing is so important that about 50-80% of the data scientist's time is spent on data preparation tasks. The main criteria for a model to be accurate and precise in predictions is that the algorithm should be able to easily interpret the data's features. Typically, data are preprocessed before being fed into the model. It is almost always necessary to transform (e.g. scaling, binarizing, one-hot encoding, etc.) into a more suitable representation for the downstream model. Most kinds of ML models take only numeric data as input, so we must at least encode the non-numeric data. The majority of the real-world datasets are highly susceptible to missing, mislabeled, inconsistent, and noisy data. Applying machine learning algorithms on that data would not give quality results as they would fail to identify patterns effectively. Data processing is, therefore, important to improve the overall quality of the data pipeline. Based on initial discussions with the Hub community, I believe that automated dataset optimization can transform the way developers and data scientists prepare data for machine learning pipelines.

Objectives

In an effort to improve the data preprocessing for non-experienced users and help the Hub community to build ready-to-production pipelines faster, I will work towards extending the functionality of Hub in the following ways (along with a few principles I have in mind):

- **Easy-to-use and Clear APIs.** Simplicity in user-facing API has been a central component of many successful machine learning libraries, such as `scikit-learn`. To transform a dataset, a minimal number of parameters needs to be specified. This is especially valuable for non-experienced users. This is also important to alleviate the issue of the black-box in the machine learning area, where models are currently being used without much understanding of their internal behavior. I believe that developing APIs for various dataset transformations that are clear in the first place will make it easier for users to understand what's happening under the hood. I will first work towards making basic transformation strategies available for numerical data, text and images. As an example, I will first work on developing transformations for numerical data (e.g., normalizing, scaling, one-hot encoding, etc.). Then I will make specific improvements for each area and extend the collection of available pre-processing strategies if necessary.
- **Automating Preprocessing for Raw Data.** Preprocessing is usually written and performed separately, before building and training the model. We fit some transformers, transform the whole dataset(s) and train the model on the processed data. For a complex task, a single-step preprocessing would not be enough. Raw data initially collected can be very noisy, contain useless columns or splitted into different dataframes or tables sources. The first data processing is usually performed on an original raw dataset without considering any kind of model data eventually will be fed in. The dataset is cleaned and the following additional processing and the model are built considering only the cleaned data. The preprocessing step, on the contrary, should be considered closely tied with the downstream ML model and adapted to its particular "needs". However, even plugging in a raw dataset in a specific model might be an issue for non-experienced users. For example, a specific model might only accept RGB colorspace, but the user has given a number of images in grayscale. Given a specific model, it would be useful to be able to automatically apply preliminary transformations to raw data. If we have a raw dataset, transformations targeted on completeness of the dataset, consistency in features and absence of duplicates are more important than more advanced data transformations. While keeping this in mind, I will design and build the functionality (or a toolkit) to (1) automatically apply basic required preprocessing for a specific model on a raw dataset (e.g., resizing all of the images to the same resolution that the model requires) (2) find a set of optimal processing strategies for a dataset that has already passed some basic pre-processing to further increase dataset quality (i.e., in terms of the increasing predictive performance for a specific scoring function of a model).
- **Experimenting and Researching Novel Automated Pre-processing Methods.** There is no defined methodology to find the set of strategies that would improve the overall dataset quality for a downstream model. Automated pre-processing is still a novel area of research: only a few open-source tools are available and only a few papers are

published. Therefore, I will spend significant time researching and experimenting with various approaches that are leveraged in the literature, such as meta-learning.

Methodology

From my quick preliminary research I found out that the approaches to finding a set of preprocessing parameters can be classified into four main categories:

1. **Manual:** select pre-processing parameters based on intuition/experience/guessing, train the model, and score on the validation data.
2. **Grid Search:** set up a grid of pre-processing parameter values and for each combination, train a model and score on the validation data.
3. **Random search:** set up a grid of pre-processing parameter values and select random combinations to train the model and score.
4. **Automated Search:** use methods such as gradient descent, Bayesian Optimization, or evolutionary algorithms to conduct a guided search for the best parameters (i.e., continuously pick the most statistically promising parameters).
5. **Automated Search via meta-learning:** use meta-learning to predict the impact of transformations on the final performance of models on the corresponding datasets. [This paper](#) is a great example of this approach.

The difference between the last two approaches is that when using methods such as gradient descent or Bayesian Optimization, we have to continuously train and validate the model to be able to rank the impact of the transformations. On the contrary, the meta-learning approach attempts to predict the impact of the transformations without training the model on the validation set. With this approach, the ranking of the transformations according to their impact on the final result can be available without actually training the model. Meta-learning is a promising approach, but I think that it might be too challenging for me to implement this as meta-learning for pre-processing is not yet widely used and discussed only in a few research papers. As a starting point, I will use a more naive approach. I will first create a set of parameters for a specific data transformation (e.g., normalization for numerical data, augmentation for image data, etc.), apply the transformation and train the model with that dataset. I will repeat this over n times and pick the parameters that yielded the best results. I will first implement a search of data transformation parameters via grid search and then further improve it by using more advanced methods such as Bayesian Optimization. I will also start with simple transformations (e.g., one-hot encoding for numerical data, cropping for image data, etc.) before working on more advanced ones. I will also use only a single model (e.g. [Resnet18](#)) as a baseline to develop a general pipeline first before adding support for more models.

From a technical standpoint, the majority of the development (e.g. data transformations) will be done with the tools already used by many for data pre-processing. Many of the new features I plan to implement, such as transformations for numerical data, can be accomplished by using existing pre-processing modules in popular machine learning libraries. For example,

a large number of the data preprocessing transformations for numerical data such as one-hot encoding, discretization, imputation and normalization are available in `sklearn.preprocessing` package. Pytorch's `torchvision.transforms` are also used as a part of the transformation pipeline that performs operations such as normalization and image augmentation. However, a couple of features cannot be implemented using only existing libraries: a general algorithm that can find the most optimal parameters for a specific data transformation. I will implement this algorithm by taking a look at `GridSearchCV` and `RandomizedSearchCV` classes taking a model, a search space, and a cross-validation configuration. The benefit of these classes is that the search procedure is performed automatically, requiring minimal configuration. I have an intuition that hyperparameter tuning of a machine learning model should be similar to hyperparameter tuning for a specific data transformation. Similarly, the `scikit-optimize` library provides a similar interface for performing a Bayesian Optimization of model hyperparameters via the `BayesSearchCV` class.

Taking into account all the possible pre-processing strategies for each modality is extremely challenging. Identifying the correct preprocessing and augmentation steps for increasing model performance requires a firm understanding of the problem, data collected, and production environment. For an image dataset, changing the size of an image sounds trivial, but there are considerations to take into account. Many model architectures require square input images, but few devices capture perfectly square images. Altering an image to be a square calls for either stretching its dimensions to fit to be a square or keeping its aspect ratio constant and filling in newly created "dead space" with new pixels. Moreover, input images may be various sizes, and some may be smaller than the required input size. Therefore, preserving scale is not always required, filling in dead pixels with reflected image content is often preferred and downsampling large images to smaller images is often safest. Randomly mirroring an image forces the model to recognize that an object need not always be read from left to right or up to down. Flipping may be illogical for order-dependent contexts, like text recognition on an image. However, for the most real world objects, flipping is a strong way to improve performance. The pre-processing steps for images include many more steps including orientation, random rotations, random noise, random exposure and/or brightness. There is no right way or the order to preprocessing. Therefore, I will experiment with various data collection and model inference contexts to make the tool that can scale to a larger number of datasets.

Deliverables

- A set of high-level APIs for data transformations. The set of specific pre-processing transformations will be refined later with the mentors.
 - Image data: Rotation, augmentation, cropping, mirroring, segmentation
 - Text data: TF-IDF, stemming, lemmatization
 - Numerical data: Normalization, scaling, one-hot encoding

- A general algorithm that can find the most optimal parameters for a specific data transformation. This algorithm should be able to rank transformations according to the improvement of the final result of the model training.
- Docs on how to use data transformation APIs for various data types.
- Quick overview and comparison of similar open-source solutions such as <https://github.com/albumentations-team/autoaugment> and <https://github.com/DeepVoltaire/AutoAugment>.

API Overview

The following is a short description of the high-level API for this project. This is a high-level schema idea of how users would interact with the API rather than a complete API description.

```
ds.optimize(estimator, transform, search_spaces, strategy)
```

`ds.optimize()` takes a model, transformation type and a set of optional parameters for data transformation, evaluates it and returns a score for the data transformation with the parameters that maximize the training metrics (e.g. training loss/accuracy, confusion matrix, etc.).

Args

`estimator` - Estimator baseline model (e.g., `'Resnet18'`).

`transform` - A specific data transformation that needs to be applied to the dataset (e.g. `'augmentation'`)

`search_spaces` - An optional parameter to provide a specific range of parameters for a specific data transformation type.

`strategy` - An optional parameter to provide a strategy (i.e., optimization algorithm) to use when looking for the best set of parameters (e.g., `'grid_search'` , `'randomized_search'` or `'bayes_search'`)

Returns

`results` - A dict with applied sets of parameters and their ranks

`best_params` - A set of best parameters fetched

`best_score` - Best score fetched

The `scikit-learn` offers useful methods for cross-validation, model selection, pipelining, and grid search abilities. I would like to use or adapt `scikit-learn` simple methods that are available in the library such as cross-validation and grid search.

I will use the existing `hub.core.transform` module to implement different data-preprocessing strategies for various data types format as this module already supports high-level API for defining a pre-processing function that would be applied to a dataset.

Benefits to Community



First, Hub users who are just starting out with machine learning will be able to build data pipelines faster. They will be able to shift their focus towards working with data (e.g. data visualization, feature engineering and data modelling), as opposed to completing basic data pre-processing (e.g. imputing missing values). Currently, it is a complex task to review all of the possible pre-processing transformations as there exists an extremely large number of options. Finding the most optimal set of these parameters that improves the prediction quality of the model is even more difficult.


Second, In multimodal machine learning, we aim to build models that can process and relate information from multiple modalities. Learning from multimodal sources offers the possibility of capturing relations between modalities. An example can be images that are usually associated with tags and text explanations. Multimodal representation is the task of representing data from multiple modalities in the form of a vector or tensor. Since data from multiple modalities often contain both complementary and redundant information. A first fundamental challenge is learning how to represent and summarize multimodal data in a way that exploits the complementarity and redundancy of multiple modalities. The heterogeneity of multimodal data makes it challenging to construct such representations. However, high-quality representations that have similarity in the representation space are important for the performance of machine learning models. This project will be a first step towards tackling some of these challenges including dealing with different noise levels, missing data, and combining data from different modalities.

Finally, the area of automated data pre-processing is not yet widely studied among the researchers and not often used in the implementation of data pipelines. The project will help both researchers and developers to better understand the state of the field and identify directions for future research.

Development Timeline

The following is a timeline for this project.

 Time Period	 Task
<u>Proposal Deadline (April 19th - May 20th)</u>	Project Proposal Submission Deadline • Read Hub docs and study the code base.
<u>Project Announced to Public (May 20th - June 13th)</u>	Community Bonding • Get involved with the Hub community. • Establish feature requirements with the mentors. • Get to know the mentors better.
<u>Week 1 (June 13th - June 19th)</u>	Coding Period Start • Create a generic structure and schema of <code>ds.optimize</code> API.

<u>Aa</u> Time Period	 Task
<u>Week 2 (June 20th - June 26th)</u>	<ul style="list-style-type: none"> • Add support to create data transformations for numerical data.
<u>Week 3 (June 27th - July 3rd)</u>	<ul style="list-style-type: none"> • Add support to create data transformations for text data.
<u>Week 4 (July 4th - July 10th)</u>	<ul style="list-style-type: none"> • Add support to create data transformations for image data.
<u>Week 5 & Week 6 (July 11th - July 24th)</u>	<ul style="list-style-type: none"> • Add support for a grid search to <code>ds.optimize</code> for finding pre-processing parameters. For each of the sets, train a model and score on the validation data.
<u>Week 7 & Week 8 (July 25th - August 7th)</u>	<ul style="list-style-type: none"> • Implement scoring functions for each model • Add support for storing the results in <code>results</code> and <code>best_score</code>.
<u>Week 9 (August 8th - August 14th)</u>	<ul style="list-style-type: none"> • Add support for specific parameters for each data transformation (e.g. a set of parameters that will be used in grid search for <code>'augmentation'</code> transform). • Add support to provide <code>search_spaces</code> • Fix various unpredictable cases specific to each data transform.
<u>Week 10 (August 15th - August 21st)</u>	<ul style="list-style-type: none"> • Add support to provide more <code>estimator</code> models
<u>Week 11 (August 22nd - August 28th)</u>	<ul style="list-style-type: none"> • Add support for a method such as Bayesian Optimization to improve the <code>ds.optimize</code> to be able to continuously pick the most promising transform parameters
<u>Week 12 (August 29th - September 4th)</u>	<p>Final Submission</p> <ul style="list-style-type: none"> • Write docs for the feature • Verify that the feature is operating as expected and tidy up any loose ends. • Submit the final code with complete documentation to the Git repository.

I would prefer a large project (350 hours) as I believe I can make the most impact if I familiarize myself well with the codebase and get better involved in the community. It is supposed that I will have no commitments during this summer except for my student job which takes about 10 hours per week. Therefore, I will be committing around 25-30 hours per week to the project until the end of the coding period. However, I'm definitely willing to make more efforts if the project is behind schedule. I should also be able to extend the coding period if needed.

Future Goals

If there is enough time, I would like to contribute it to some of the ideas I have about extending the feature:

- When a user wants to preprocess a dataset, they select an algorithm to be used for the analysis and then the system automatically recommends transformations to be applied. This will probably be only possible by using the meta-learning approach.

- Users should be able to see what transformations have been applied to a dataset to improve the prediction metrics. Even though the APIs should be working with minimal parameters from an end-user, more experienced users should be able to get an overview of the specific details of the transformations.
- Write a short research paper provided that the results of the project are significant or the approach that is implemented is novel. Although I expect that this won't add value to Hub in terms of the source code, this can be a logical wrap-up for the project as I believe this project has a strong scientific component. This new taxonomy will also help researchers to better understand the state of the field and identify directions for future research.

About Myself

I am a computer science student at the University of Luxembourg, graduated in July 2021. In terms of programming experience, I have programmed in Python for 4 years, and in JavaScript for 3 years. As a result, I have thorough experience with the languages and technologies used by the Hub community. I am also well-versed with machine learning and with existing ML libraries. I have used various data types (text, images, numerical data) for building ML pipelines. Therefore, I'm familiar with most of the pre-processing techniques used for various types of data. I'm also familiar with the concepts such as grid search, cross-validation and autoML. For my computer science coursework, I have taken object-oriented design and data structures, analysis of algorithms, and discrete structures. I have also taken extensive machine learning and data science coursework.

While I have not contributed to other open-source projects, I have contributed to Git repositories for more than 2 years while working for various internships and student jobs. Additionally, I have also made a number of relevant projects on my own, which I have linked here with descriptions:

- [Creating a Music Video of a Song with StyleGAN2](#). I am in charge of the initial research, planning and managing the project, and designing an end-to-end data pipeline to create generative images and music videos for the machine learning exhibition at the University of Luxembourg. I built a data-collection pipeline in Python and implemented an algorithm for clustering images and deployed the Docker containers for training 15+ StyleGAN2 (TensorFlow) models to High-Performance Computing at the University of Luxembourg.
- [Depression Classification Using Machine Learning](#). I implemented a model that classifies whether a person is depressed based on the number of speech and non-speech-related features. Experimented with various machine learning models (KNN, SVM, Keras NN, Decision Tree, XGBoost, Naive Bayes). Visualized data using Seaborn and Matplotlib.
- [Airbnb Price Prediction using Machine Learning](#). I developed a machine learning model that uses a wide range of listing data points to predict optimal prices for rental. I experimented with linear and ridge regressions, gradient boosting framework, and grid search to improve the accuracy of the model.

- Generating Music with Recurrent Neural Network (RNN). Used Recurrent Neural Network (RNN) to generate music using Tensorflow and open-source tools from the Magenta project. My responsibilities included collecting the MIDI datasets, modeling sequence data using the Long Short-Term Memory (LSTM) network, and developing a Javascript web application to interact with the model.
- Automatic Spot of Security Fixes in Source Code Repositories. I mined data from 150+ open-source git repositories using Python scripts, designed the dataset and implemented an SVM machine learning classification model that automatically spots security-fix commits. Implemented data classification and visualization using Scikit-learn, Numpy, Pandas, and Matplotlib.
- Predictive Maintenance of Aircraft with Machine Learning. I developed a machine learning algorithm with time series forecasting that helps engineers to decide the best moment to proceed with the maintenance of the aircraft. I experimented with various machine learning models and implemented anomaly detection algorithms with time-series forecasting. I implemented data preprocessing and data modeling using Pandas, NumPy, Statsmodels, Plotly and Matplotlib.

My resume (in the first section) provides more details about the projects, internships and technologies that I have worked with.

I am excited and enthusiastic about working on this project and look forward to writing code that can be used by data scientists and developers far and wide!