Journal of Machine Learning Research 23 (2022) 1-6 Submitted 3/22; Revised 8/22; Published 8/22

Deepchecks: A Library for Testing and Validating Machine Learning Models and Data

Shir Chorev SHIR@DEEPCHECKS.COM
Philip Tannor PHILIP@DEEPCHECKS.COM
Dan Ben Israel DANB@DEEPCHECKS.COM
Noam Bressler NOAM@DEEPCHECKS.COM
Itay Gabbay ITAY@DEEPCHECKS.COM
Nir Hutnik NIR@DEEPCHECKS.COM
Jonatan Liberman JONATAN@DEEPCHECKS.COM
Matan Perlmutter MATAN@DEEPCHECKS.COM
Yurii Romanyshyn YURII@DEEPCHECKS.COM
Deepchecks Ltd.
Derech Menachem Begin 14,
Ramat Gan, 5270002, Israel

Lior Rokach, LIORRK@BGU.AC.IL
Department of Software and Info. Sys. Eng.,
Ben-Gurion University of the Negev.
Beer-Sheva 8410501, Israel

Editor: Joaquin Vanschoren

Abstract

This paper presents `Deepchecks`, a Python library for comprehensively validating machine learning models and data. Our goal is to provide an easy-to-use library comprising many checks related to various issues, such as model predictive performance, data integrity, data distribution mismatches, and more. The package is distributed under the GNU Affero General Public License (AGPL) and relies on core libraries from the scientific Python ecosystem: `scikit-learn`, `PyTorch`, `NumPy`, `pandas`, and `SciPy`. Source code, documentation, examples, and an extensive user guide can be found at `https://github.com/deepchecks/deepchecks` and `https://docs.deepchecks.com/`.

Keywords: Supervised Learning, Testing Machine Learning, Concept Drift, Python, Data Leakage, MLOps, Bias, Explainable AI (XAI)

1. Introduction

Machine learning (ML) models are becoming increasingly popular in a variety of fields, including healthcare, finance, biology, and others. Complex models can now be easily trained using modern software packages and yield high predictive performance on test sets. Nevertheless, models are often challenged when deployed outside the lab.

As indicated in previous works (e.g., (Xie et al., 2011)), detecting faults in machine learning models can be difficult. This is especially true when models are used in sensitive decision-making processes, where mistakes can have serious consequences. Despite the sensitivity of the matter, many ML models are put into production unmonitored and without proper testing, and thus they are prone to significant risks.

Running ML models in the real world poses a variety of challenges, which may cause degradation in its predictive performance, e.g.:

- Data integrity issues: Data pipelines are often complex, and the data format may change over time. Fields may be renamed, categories may be added or split, and more. Such changes can have a significant impact on your model's performance.
- Data drift and concept drift: Data in the real world constantly changes. This, in turn, may affect the distribution of the data being fed to the model or that of the desired target prediction. Thus, over time, the data used to train the model becomes less and less relevant. The data format is still valid, but the model will become unstable, and its predictive performance will deteriorate over time (especially in the case of out-of-distribution (Geirhos et al., 2020)).

Table 1 presents the existing open-source packages that help ML practitioners to validate their processes. Specifically, `deequ` library [1] enforces data schema, such as column names and properties. The `auditor` library [2] is an R package that generates model agnostic visual plots for model verification, validation and error analysis. The `TDDA` library [3] provides tests for validating the data and data manipulation processes. The `FastDup` tool [4] provides an efficient method for detecting clusters of duplicate or similar images within a dataset such as CIFAR. It also helps to identify anomalous images that might need to be excluded before training the model. The `erroranalysis.ai` library [5] aims to identify cohorts with high error rate versus benchmark and visualize how the error rate distributes. From Table 1 we can conclude that `Deepchecks` is the most comprehensive open-source library which covers data integrity, model evaluation and train-test distribution checks. In addition, `Deepchecks` is the only package that supports checks for both tabular and image datasets.

2. The Deepchecks Library: An Overview

The `Deepchecks` library provides a framework of data, models, checks, suites, and conditions that enable customizable and extensible testing for ML. It suggests a unified framework with a corresponding application programming interface (API) for testing models and data. It also allows the user to concatenate several checks to be executed later as a single test command.

In addition, The `Deepchecks` library comes with many built-in checks and suites that can help validate various points throughout the machine learning development process.

Of course, every process has its unique steps and challenges, and therefore all checks and suites can be easily customized. In particular, the user can add new checks and their results to support the validation of various phases in the pipeline. Alongside that, we have

1. `https://github.com/awslabs/python-deequ`, visited 2022-07-16
2. `https://cran.r-project.org/web/packages/auditor`, visited 2022-07-16
3. `https://tdda.readthedocs.io/en/latest/`, visited 2022-07-16
4. `https://github.com/visualdatabase/fastdup`, visited 2022-07-16
5. `https://erroranalysis.ai/`, visited 2022-07-16

Table 1: Existing open-source tools for ML-related validation

Package Name	**Data Integrity**	**Model Evaluation**	**Train-Test Distribution**	**ML Domain**	**Platform Support**
deequ	✓			Tabular	Based on Apache Spark, python interface
auditor		✓		Tabular	cross platform, R
TDDA	✓			Tabular	cross platform, python
fastdup			✓	Vision	linux & MacOS, python
erroranalysis.ai	✓	✓		Tabular Data	cross platform, python
Deepchecks	✓	✓	✓	Tabular Vision	cross platform, python

identified several recurring scenarios, each of which has its own needs and characteristics. In particular, `Deepchecks` includes predefined suites for the following scenarios:

- New Data (Data Integrity) - When a user starts working on a new task, `Deepchecks` helps to validate data's integrity (For example, detecting duplicate samples, problems with string or categorical features, significant outliers and inconsistent labels)
- After Splitting the Data (Train-Test Validation) - When splitting the data (e.g., to train, validation or test) and just before training the model, `Deepchecks` ensures that the splits are representative. For example, it verifies that the classes are balanced similarly, that there is no significant drift in the distributions between the features or labels in each of the datasets, and that there is no potential data leakage that may contaminate the model.
- After Training a Model (Model Evaluation) - Once a trained model is available, `Deepchecks` examines several performance metrics, compares them to various benchmarks, and provides a clear picture of the model's performance. `Deepchecks` attempts to find sub-spaces where the model underperforms and provide insights that may be used to improve its performance.

While the above-mentioned scenarios are predefined, a common use case is also to run specific checks on an "on-demand" basis. This is particularly useful when the user is looking into a problem, such as over-fitting.

3. Deepchecks' Building Blocks

Depending on the checks that the user wishes to execute, some of the following objects should be provided as input: Raw data (before pre-processing) with optional labels, the training data (after pre-processing) with the target attribute, test data (which the model is not exposed to), with optional labels, and the model to be checked.

For tabular data, the `Deepchecks` library requires that the model have a `predict` method for regression tasks and in addition `predict_proba` method for classification tasks, both of which should be implemented by the `scikit-learn` API conventions (Pedregosa et al., 2011). Checks may attempt to use additional model methods if they exist. For example, it uses the built-in feature importance property if it exists, and if it does not, it calculates the feature importance through the permutation importance procedure (Breiman, 2001). Note that built-in scikit-learn classifiers and regressors, along with many additional popular models types (e.g. XGBoost (Chen and Guestrin, 2016) and LightGBM (Ke et al., 2017)) implement these methods and are thus supported. The three main building blocks of the `Deepchecks` library are checks, conditions, and suites.

3.1 Checks

A check aims to inspect a specific aspect of the data or model. Checks can cover all kinds of common issues, such as data leakage and concept drift. To date, the `Deepchecks` library contains 62 checks, 42 of which support classification and regression models trained on tabular data[6]. The remaining checks support computer vision models trained on image data. The checks available to users are listed in the online documentation[7]. The library includes three main categories of checks (called modules):

1. `Data Integrity` - This module provides meta-information regarding the dataset, such as: the role and logical type of each column. Additionally, this module contains all data integrity checks, including: checking for duplicate samples; detecting a small amount of a rare data type within a column, such as a few string samples in a mostly numeric column and checking if there are columns which have only a single distinct value in all rows.

2. `Train-Test Validation` - This module contains various checks for estimating if the training and test sets have different distributions. This module includes checks such as: Calculating the drift between the train set and test set per each input feature (in particular, Earth Movers Distance for numerical variables and Population Stability Index for nominal features); Comparing the model's trust score (Jiang et al., 2018) of the train and test sets. Additionally, this model contains checks for detecting leakage, such as: Calculating the difference between the train set and test set and detecting samples in the test set that also appear in the training set; Calculating the Predictive Power Score[8] of all features, in order to detect features whose ability to predict the target is due to leakage.

3. `Model Evaluation` - Module that contains checks of model performance metrics, such as: Comparing the predictive performance of a given model to that of a relatively simple baseline model (e.g., a single tree model); Calculating the calibration curve with Brier score for each class; Checking the distribution of errors; Finding features that best split the data into segments of high and low model error [9]. This module

6. Deepcheck library supports NumPy ndarray, Pandas Dataframe object and Pandas Series object. The Series object is used for checking leakage over time by defining the time column. Time series models are still not supported but are planned to be added in forthcoming versions.
7. `https://docs.deepchecks.com/stable/checks_gallery/tabular.html`, visited 2022-08-12
8. `https://github.com/8080labs/ppscore`, visited 2022-07-16
9. The check WeakSegmentsPerformance performs error analysis by training a shallow decision tree on the test set. However, instead of using the original target attribute (y), it uses the prediction error as the

also contains checks for methodological flaws in the model building process, such as: Checking for overfitting caused by using too many iterations in a gradient boosted model; Detect features that are nearly unused by the model; Additionally, this module provides meta-information about the model, such as the model's hyperparameters and inference time.

Each check can have two types of results: A) A visual result meant for display (e.g., a figure or a table); B) A return value that can be used for validating the expected check results. These values can be further used as predicates in conditions (validations are typically done by adding a "condition" to the check, as explained below).

3.2 Condition

A condition is a function that can be added to a check to validate if the check's return value complies with a predefined threshold or logic. A condition returns a status of either pass, fail, or warning result, as well as a statement that describes the status (e.g. "found 7% duplicate samples"). The last two results indicate that a flaw may exist, and further investigations are required.

For example, a condition attached to the check of `DataDuplicates` may return the status fail if there are more than 5% duplicate samples in the dataset. This may be valid if the duplicates are intentional (e.g., because of intentional oversampling or because the dataset's nature has identical-looking samples). Still, if this is a hidden issue that is not expected to occur, it may indicate a problem in the data pipeline (Barz and Denzler, 2020).

3.3 Suites

A suite is an ordered collection of checks that can have conditions added to them. Once a suite is executed, a summary report is generated, which consists of high-level results and detailed results. The suite mechanism efficiently runs a large group of checks with a single call and displays a concluding report for all of the checks that ran. The library comes with a list of common predefined suites for tabular data. The user can build a customized validation scenario adapted to the pipeline's models, domains, and timing. These suits can be shared with the community and re-used, serving as a framework for methodological testing.

4. Conclusion and Future Work

Here, we presented `Deepchecks`, a library for validating machine learning models and their corresponding datasets. The library currently supports classification and regression models for tabular data and computer vision models trained on images. We are continuously adding new checks and improving usability, documents, and tutorials. Specifically, we plan to add checks for models that support time-series and Natural Language Processing (NLP) tasks. Finally, we welcome contributors to help us at `https://github.com/deepchecks/deepchecks`.

target attribute. This allows the user to focus on the least performing subcohort. This method was recently tested in rating prediction of recommender systems (Misztal-Radecka and Indurkhya, 2022) and it was also adopted by other libraries such as erroranalysis.ai.

References

Björn Barz and Joachim Denzler. Do we train on test data? purging cifar of near-duplicates. *Journal of Imaging*, 6(6):41, 2020.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, 2020.

Heinrich Jiang, Been Kim, Melody Guan, and Maya Gupta. To trust or not to trust a classifier. *Advances in Neural Information Processing Systems*, 31, 2018.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems*, 30, 2017.

Joanna Misztal-Radecka and Bipin Indurkhya. A bias detection tree approach for detecting disparities in a recommendation model's errors. *User Modeling and User-Adapted Interaction*, pages 1–37, 2022.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of Machine Learning research*, 12:2825–2830, 2011.

Xiaoyuan Xie, Joshua WK Ho, Christian Murphy, Gail Kaiser, Baowen Xu, and Tsong Yueh Chen. Testing and validating machine learning classifiers by metamorphic testing. *Journal of Systems and Software*, 84(4):544–558, 2011.