gingado: a machine learning library focused on economics and finance

Douglas Araujo

8 February 2022, draft - Please do not cite

1 Introduction

gingado is an open-source machine learning library[1] that aims to broaden the accessibility of state-of-the-art models to a wide range of practitioners in economics and finance, including beginners. The focus is on promoting good modelling practices and providing an accessible interface to economists, statisticians and other practitioners, some of whom may not be necessarily experts in machine learning. The good modelling practices are encouraged through functionalities that help users establish a quick and reasonable benchmark model, as well as an easy way to document their model. In its turn, the accessibility of *gingado* comes from its design that keeps many of the complexities under the hood - although they are accessible to advanced users that need to customise certain aspects.

Even while accessibility and good practices are priorities for *gingado*, it offers good performance in terms of speed, since the model computations are performed by widely used backend engines that are well known for their performances and high computation efficiency. These libraries are `scikit-learn`, `XGBoost`, `PyTorch` and `TensorFlow`. Most notably, the training of some models *gingado* can be accelerated by graphical processing units (GPU).[2] An overview of *gingado*'s philosophy, license, and high-level description of its design is found in section 2 and the ways the machine learning backends are used by *gingado* are outlined in section 3.

gingado wraps these backend libraries in a way that offers the following main functionalities to support the machine learning modelling workflow:

- Automatic data augmentation

[1] The code will be available on https://github.com/dkgaraujo/gingado after mid-February 2022. Issues, feature requests and pull requests wit code suggestions or corrections are all welcome.

[2] Currently `XGBoost` only supports graphics cards using CUDA (NVidia), `PyTorch` can only accelerate calculations on GPUs using the CUDA and ROCm (AMD cards) libraries, although there are indications from the maintainers that in the near future also the Metal framework (Apple) will be supported, and `TensorFlow` accelerates on these three graphics cards previously mentioned.

- Automatic benchmark models
- Automatic documentation

These functionalities listed above are described in sections 4, 5 and 6, respectively. Together, they should enable quick experimentation cycles, and once the user has landed on a model considered satisfactory (or decided to go with the automatic benchmark), *gingado* will facilitate its replication and documentation.

While *gingado* is targeted for users across different levels of experience, including beginners, this paper itself is not an introduction to machine learning. Other sources cover this material in a very good manner.

2 Overview

2.1 Philosophy

gingado is designed from the principle that a machine learning model is, first and foremost, a *model*: a structured perspective towards an economics or finance problem that

1. has a well-defined question: is it a classification or a regression task? Or is it a clustering or dimensionality-reduction task?
2. has a well-defined way to evaluate the potential responses to this question: the loss function or functions with which competing models are evaluated
3. is trained on a well-defined dataset that; and similar to real life the datasets can sometimes be augmented by other existing data that is pertinent to the problem at hand
4. has a reasonable benchmark established really quickly, so that the user can evaluate whether further work on the model (which is cognitively expensive for the user) is justified, ie if it improves on the performance of that quick, cheaply obtained benchmark
5. allows for the combination of insights from multiple algorithms
6. is reproducible to the extent possible and shareable: thus fomenting the spread of ideas and of solutions beyond the initial use case.

The design of *gingado* largely follows from the philosophy stated above. The main interface with the user is the creation of an instance of the class `GingadoModel`. A `GingadoModel` instance contains the following objects:

- the dataset used to train the model;
- an augmented dataset that adds to the dimensionality of the original dataset;

- the pipeline of transformations (eg, imputation of missing data points; re-scaling of the data, etc);
- a benchmark to help evaluate further development of the model;
- (optional) models defined by the user;
- information for documentation of the model;
- metadata to document model development.

In addition to the points above, *gingado* aims to be of simple use, enabling quick rounds of experimentation. These experimentation rounds should facilitate the user to reach a satisfactory outcome within a reasonable time frame. But, this simplicity does not come at the expense of flexibility, as users maintain the possibility of accessing the underlying objectives (ie, the actual trained model itself, or the datasets). The ability to access these objects created using the backend libraries is important in case users want to access more details about model training outcomes.

2.2 Environment

gingado is written in python, and is installed and managed as a python package, with the command:

Listing 1: Installing *gingado*

```
$ pip install gingado
```

The minimum python version supported is 3.10.2. It can also be used to develop and train models in the context of other programming languages that have bindings with python, such as R through the `reticulate` package (Ushey, Allaire, and Tang 2022). Similarly, *gingado* can be used in association to Stata.[3] In addition, trained *gingado* models can be deployed in a variety of settings (including web apps).

gingado uses semantic versioning,[4] which should help with reproducibility over time, as versions change to add new features and fix any bugs.

2.3 License

The code is made publicly available for free under the Apache 2.0 license. It is a permissive license that allows users to use *gingado* commercially, as well as to use *gingado* in their own projects and subject those projects to a different license if they wish.

[3] See `https://www.stata.com/python/pystata/index.html` for instructions and examples of how to use Stata within a python environment. *gingado* models could either be trained with data manipulatd by Stata routines, and/or the outputs of *gingado* models could be analysed in Stata environments.

[4] https://semver.org contains details of what this means for the author and users of *gingado*

3 Backend engines

The machine learning engines used by *gingado* are `scikit-learn`, `XGBoost`, `PyTorch` with `Fast.ai`, and `TensorFlow`.

`scikit-learn` (Pedregosa et al. 2011) offers a wealth of machine learning algorithms, all accessible with a consistent API (see Buitinck et al. 2013 for considerations on their API design). It is an important component of *gingado*, as it provides the preprocessing functions, the functions used to separate training, validation and testing datasets, the ability to pipe all these steps in a coherent manner, as well as some model algorithms themselves.

`XGBoost` (Chen and Guestrin 2016) is a tree-boosting system,[5] engineered to provide scalability with large datasets. It is also renowned for its stellar performance in many machine learning challenges, as well its ability to handle missing data more smoothly than other algorithms. *gingado* uses `PyTorch` (Paszke et al. 2019) as wrapped by `fast.ai` (Howard and Gugger 2020). `PyTorch` is a deep learning library that combines ease of use, high performance, wide latitude to customise neural networks and an idiomatic syntax (ie, the code is "pythonic" in the sense that it does not depart from the normal flow from a python-based programmer). Similarly, `Keras` is a deep learning library that provides a simple yet flexible API that aims to promote experimentation in deep learning. `Keras` uses `TensorFlow` (Martín Abadi et al. 2015) as its backend.

A word on other backends: users that want to use their data with other backends (eg, ONNX (Bai, Lu, Zhang, et al. 2019)) can do so by simply calling the appropriate methods that extract the data from a `GingadoData` instance (see subsection 8.3.2) and using that data to train the model as normal.

`Fast.ai`/`PyTorch` and `Keras`/`TensorFlow` are deep learning libraries that are not typically associated with tabular data such as the ones used in economics and finance. They are directly supported by *gingado* as backend engines due to the following main advantages of using them:

- Extensive documentation and tutorial examples, that users can refer to when building their models
- Ability to customise functions if needed (ie, it is very "hackable")
- Ability to integrate multi modular data (ie, include texts and images in a tabular file)

Together, these backends should provide users with a compelling combination of a robust experimentation environment that performs well, and offers a wide variety of algorithms to explore. And the automatic benchmarks (section 5) provide users that might otherwise be constrained by the broad availability of choices with a starting point for their journey.

[5] In plain language, boosting algorithms refer to machine learning methods that combine several predictors that achieve-slightly better-than-chance performance to create a well-performing learning algorithm.

4 Automatic data augmentation

Data augmentation, or in plain English, increasing the dataset available to train the model, is one technique used frequently to improve the ability of machine learning models to understand the underlying patterns from data instead of memorising the results.[6]. Data augmentation is frequently done, for example, during the training of machine learning models that take images as inputs. For a given limited dataset of, say, pictures of cats and dogs, commonly used machine learning libraries implement routines that flip some images or distort them. Thus, in addition to the original dataset now the model learns from this "augmented" data that still allow it to learn the main features of dogs vs cats. Inspired by these techniques, *gingado* offers users the possibility of augment their own datasets.

In the context of economics and finance, virtually all of the data is in the "tabular" format, ie numbers, as opposed to images, videos, text, etc. And given the wealth of statistical information available for free from official sources, it can be reasonable to expect that many use cases for *gingado* involve dimensions (either cross-section or over time) that could be better described by the data, if there was a way to include these statistics in the model. For example, a model to predict economic recession of a given country over time could benefit from adding to the original dataset a number of other variables occurring in the same country in the same time period as used for training. This possibility offers an advantage compared to other machine learning tasks that use different types of data as inputs (eg, the dogs vs cats classification model) because usually additional data on the same entities and/or time periods being studied do not exist in these cases.

According to Goodfellow, Bengio, and Courville 2016, data augmentation (as traditionally done, ie with images or speech recognition) is easiest for classification tasks. This is because a classifier model takes as input high-dimensional data x and learns to summarise it by outputting a specific category identity y, and therefore these algorithms tend to be invariant to a broad spectrum of transformations. But in the case of tabular data, data augmentation does not need to involve transformation - instead it can rely on additional datasets that inform the original entities in x. In practice, this means that it is reasonable to expect that the augmented dataset will contribute to the task at hand, be it classification or regression. The claim is that including freely available additional datasets that are relevant for the case at hand could increase the chances that the machine learning model will learn useful patterns that generalise well. This is because this additional datasets help elucidate even more dimensions about the entities being modelled, which tends to have a positive result given that machine learning models usually deal well with high-dimensional settings unlike traditional statistical and econometric techniques.

But of course, the claim above is not a theorem: augmenting the dataset may not be improve performance in all cases. In practice, there is no way currently

[6] In other words, to improve models' ability to generalise better by improving their out-of-sample performance.

to know if the desired result from augmenting the dataset is achieavable across a wide spectrum of machine learning models. For this reason, *gingado* tests two versions of the benchmark model described below in section 5. The user is then informed about their different performances, and is thus in a good position to decide whether or not to use the augmented dataset in the experimentations that ensue.

4.1 Sources

4.1.1 Own data

gingado assumes that the original dataset provided by the user does not have any polynomial transformation (eg, if the dataset is the GDP growth for countries across time, that the GDP growth variables are not squared). Thus, it provides an automatic data augmentation in the form of including polynomial interactions of the variables, in the second degree. In other words, *gingado* adds the square of each variable and their interactions with each other.

4.1.2 SDMX

gingado explores and fetches available datasets from official sources to augment user data using the Statistical Data and Metadata eXchange (SDMX)[7]. SDMX is an initiative by international organisations (Bank for International Settlements, European Central Bank, Organisation for Economic Cooperation and Development, International Monetary Fund, World Bank, Eurostat, and United Nations) that develop and publish statistics from these sources. Since its inception in 2001, SDMX has grown to be used by other sources as well - primarily central banks and statistics agencies - as a standard to disseminate their data.

There are many other official agencies that offer freely (and easily) accessible datasets that would certainly be useful to the proposed data augmentation. For example, the Federal Reserve Bank of St Louis maintains the Federal Reserve Economic Data (FRED)[8] and related APIs, and the Brazilian Institute for Geography and Statistics (IBGE) has a range of APIs[9] that offer datasets that could be of use. While they would indeed be useful, incorporating each of these sources as part of the baseline *gingado* package would not scale given each source's description of the data, encoding of the variables and dimensions, and overall API design. That is where the main advantage of SDMX shows: the SDMX encodes a set of cross-domain code lists that ensures that the values inputted by the users with respect to dimensions of the dataset are consistent across sources.

For example, using SDMX to look across multiple sources for data of quarterly frequency related to the countries of Argentina, Brazil and South Africa

[7] `https://sdmx.org/wp-content/uploads/SDMX_3-0-0_SECTION_1_FINAL-1_0.pdf`
[8] `https://fred.stlouisfed.org`
[9] `https://servicodados.ibge.gov.br/api/docs/`

for the period spanning the first quarter of 2015 to the fourth quarter of 2021 would use the code lists for frequency and for reference area. These are the same across the SDMX sources, which facilitate the process of finding relevant datasets. It also helps to ensure that user data can be safely merged with augmented data, by having a well-defined list of possible values of each of these variables to check the user dataset. For example, if a use case relies on identifying the currency related to each observation, then *gingado* can first ensure that the original dataset is using currency codes just as in the SDMX code list for currency, before any augmentation is performed.

In practice, this is done using the pandasdmx python package. The backend code looks at all listed SDMX sources in this package, and retrieves the dataflows for those it is able to get (some sources timeout). Given that downloading all the relevant data from all sources could be an expensive operation, users are required to define the SDMX source(s) from which to get the data.

4.2 Selecting the relevant additional data

It is reasonable to assume that only some of all the datasets added during the augmentation process are actually relevant for the use case at hand. Because holding datasets in memory could extrapolate memory limits in some use cases, *gingado* uses the following heuristic to decide which of the newly incorporated variables to keep:

1. Keep a separate list with the column names of the original dataset
2. Train a regression tree (different from the benchmark described in section 5) with the full (original + augmented) dataset
3. list the variables by feature importance; all features that are more important than the least important original variable are kept.

The process above ensures that all original dataset is still kept, and potentially new data is aggregated, thus striking a balance between augmenting the dataset in the search of higher performance and the ability to store large datasets in memory. It should be noted, however, that the regression tree mentioned in the list above is not the same automatic benchmark because they serve different objectives. The tree used during the data augmentation part is a quick (although still potentially reasonable) model that enables a transparent conclusion about feature importance. In contrast, the benchmark model described in section 5 serves the purpose of being a good benchmark that has a good chance to be put into production if the user so wishes; for this reason, the benchmark model is not necessarily the fastest to estimate and in fact, its estimation counts with expensive calculations such as fine-tuning.

5 Automatic benchmark models

5.1 The model

The automatic benchmark created is a neural network with fine-tuned parameters.[10] Gorishniy et al. 2021 show that neural networks have comparable, if not better, performance on tabular data compared to other algorithms. (To be sure, the others also perform well). Thus, this type of models was chosen as the default benchmark models in *gingado*, similar also to how they are the default model in `Fast.ai`. Specifically, the neural network parameters are:

- **number of hidden layers** - three options: 1 and 5
- **number of neurons in each hidden layer** - two options: 50 and 100
- **learning rate** - two options: 0.003 and 0.0003

The main fixed parameters (ie, not subject to fine-tuning in the automated benchmark version) are the optimizer itself, the activation function,[11] the number of epochs that the model is trained for (100), and the "patience" of the callback function for early stopping (ie, the number of consecutive epochs that the model will wait without improvement before stopping the training).

6 Automatic documentation

6.1 Importance of model documentation.

Even when machine learning models are not trained for public consumption, documenting their .... adds value to the development process, and can help future users within the same organisation (or a future version of the developer).

Especially for economists that are beginning their machine learning journey, embedding documentation as part of model development aims at facilitating their development in tandem. The automatic documentation adopted by *gingado* also encourages the calculation of performance metrics across breakdowns to better uncover biases or pitfalls of the models.

Thus, the inclusion of an automated documentation module in *gingado* aims to help both the development phase include these different performance evaluations as part of a normal process; and to encourage their use during post-training evaluation of models, or of the use of models in specific use cases.

An additional benefit occurs if the models are themselves put in the public domain, or shared across teams in the same or in different organisations. These third-party users can more easily understand the model and its implications across a reasonable breakdown of potential uses, and therefore make a more

[10] Ie, a version of the model is estimated with each combination of the parameters, and the model with the best performance in a validation sample is chosen.

[11] Ie, the non-linear function that will transform the integral data into another data.

informed judgement about its use (or the need for further development in certain areas).

Finally, thinking about model documentation since the outside helps clarify use cases, and once the model is put to production, minimise usage in contexts for which it was not intended.

6.2 Scope of documentation

The scope is the machine learning model itself.

The dataset to be used can be documented by the developer, but this is not in scope of the *gingado* library. *gingado* users are encouraged to also consider documenting the datasets used to train their machine learning models.

6.3 Process for documentation

So far this is not automated, and thus developers carry a burden of documenting the model. Also, model documentation is done (if at all) in a separate workflow from model development, whereas incorporating both processes could improve both the documentation and the model building itself if done in tandem. It could improve the model because the developers would think about the model at the same time as they are coding the model.

Mitchell et al. 2018 propose the use of "model cards", a form of transparent reporting on the machine learning model that provides benchmark evaluation across different groups. If the model "unit" is at the person- or household-level, then ideally the model should be evaluated across different breakdowns including demographics such as nationality, ethnicity, age bucket, gender and any other that might be relevant. If the model unit is at the firm-level, then firms of different sizes, or different economic sectors could be reasonable benchmarks. When the model unit is countries, then perhaps breakdowns along the lines of advanced economies vs emerging markets could help identify any major differences in performances.

gingado follows Mitchell et al. 2018 in the design of a model card. Although not all aspects in the model card model proposed by these authors is automatable (at least in the current implementation of *gingado*), implementing at least a smaller version in an automated way seeks to help include ethical, inclusiveness, fairness and sustainability considerations as part of model development, as complements to traditional evaluation metrics such as model accuracy.

6.4 Technical aspects of the documentation

The instance of a `GingadoModel` class contains the relevant information in a python dictionary. This dictionary is saved in a JSON file in the local machine whenever a model is trained. (This ensures that the relevant characteristics of the model are accessible and could be recovered easily by the user if they want to revert to a previous version.)

6.5 Documentation details

In this subsection, items in **bold** denote elements that are included in the automatic documentation JSON file.

6.5.1 Model details

Person or organisation developing the model: *gingado* can automatically include the username in the machine in which the model is developed. But, this automatic solution is likely to not work appropriately in the case development is done in the cloud[12]

Model date: This is automatically generated by *gingado*, using the system date (for human-readable documents) and also recording the timestamp behind the scenes.

Model version: This needs to be informed by the user, although it is encouraged only when the model is being saved (since *gingado* assumes saved models will be used for deployment or at least for versioning control).

Model task: if the model is a classification or regression task. THis is automatically fetched by *gingado*.

Model framework: Whether the model is a `PyTorch`, **scikit-learn**, or a `XGBoost` model. This is obtained automatically.

Model type: The model architecture (ie, if the model is a neural network, a tree, random forest, etc.) and parameters are all obtained automatic from the instance of `GingadoModel` created by the user.

Paper or other resource for information: this must be provided by the user.

Citation details: if the model is for public consumption, it would help other users to have a description of how the paper should be cited. This can be, for example, in the BibTex format.

License: when the model is for consumption of third parties (even if they are not for wide dissemination, models might be shared with other parties), it is highly advisable to be explicit about the license determining the conditions of usage of the model.[13] This information is not automatically found by *gingado* and must be provided by the user.

Contact for feedback: the contact information of the developer (or maintainer) of the code is important if the model is made for consumption of third parties. These other users might need to contact someone to solve questions, report bugs, request features and raise other issues that might be important either for future development of the model or for appropriate usage.

[12] For example, running the same code used to identify the username in an instance of Google Colab (`https://research.google.com/colab`) returns `'root'` instead of the username registered in the Google system.

[13] A useful source to help readers choose the appropriate license for their models is `https://choosealicense.com/`.

6.5.2 Intended use

None of the items of this section are captured automatically by *gingado* from the creation of the model. Users are encouraged to add the model's **primary intended uses**, as well as who the **primary intended users** are.

Out-of-scope uses: this section highlights use cases that might be similar to the intended use case, but that are out-of-scope due to the training data not being adequate, or the model itself not being suitable for this. For example, a model used to nowcast GDP could easily be confused with a model that is able to predict GDP several quarters ahead, and even as it is able to output a specific number, it would not be adequate to use it for this purpose because the model was not trained for that.

6.5.3 Factors

This section of the model card should include a description of which factors could impact performance of the model. For example, it might be that the model is substantially more accurate for firms in a given economic sector, so "economic sector" is one factor to be listed (and preferably briefly outlined) as one potential factor in such a model.

This section is only partially filled-in automatically. When training an instance of `GingadoModel`, the user can pass a parameter listing the (categorical) columns in the dataset that contain the relevant factors for which there is data. *gingado* then evaluates the model performance for the different groups.

Relevant factors describe those factors that are likely to influence performance of the model. For example, in a nowcasting model it would probably matter for performance if the model is being used during a period of high volatility / in the middle of a crisis compared to normal times. In a firm-level model, one factor for which performance can be different is the firm nationality. In loan-level or personal-level models, factors like socio-economic status, gender, ethnicity and others might also influence model performance.

Evaluation factors are those for which the model developer actually tested the performance of the model, as opposed to just relying on aggregate performance evaluation. Ideally they should be the same, or very close to, the relevant factors but in practice they might end up varying. For example, while the developer might conjecture that a loan-level credit quality model might yield less confident predictions for one gender vs the other, if the developer does not have access to gender labels for the data than they cannot evaluate the performance across genders. The intention of this subsection is to be explicit about which relevant factors were evaluated and which not, and outline the reasons for this difference.

6.5.4 Metrics

Model performance measures: which metrics were used to evaluate the model? While not specifically enforced by *gingado*, users are encouraged to consider multiple metrics when relevant for their use case instead of relying

on a single metric (or even target variable), as well as consider how to ally the quantitative predictions from the machine learning model with their own qualitative assessments. Users are also urged to consider how these metrics will affect a range of stakeholders, and to the extent possible try to consult them if the model is expected to have an impact on these groups (Thomas and Uminsky 2020).

Benchmark(s): against which baseline model was the model in question tested? *gingado* can get this information automatically from its own benchmark model, but if other benchmarks are used (eg, state-of-the-art performance points on benchmark datasets), then the user needs to include this manually.

Decision thresholds: in cases where thresholds are relevant, what are they and how were they chosen? When applicable, it could also be informative to mention the performance metrics in other thresholds value (ie, a sensitivity analysis of the threshold).

Approaches to uncertainty and variability: *gingado* automatically describes the validation strategy used in the model

6.5.5 Data

Datasets: a description of the dataset used for training and evaluating the model. *gingado* is able to capture automatically if the evaluation dataset is a split of the original dataset used in the model. If other datasets are used to evaluate the model, users should inform them manually. Ideally, if the dataset is publicly accessible, users are encouraged to include the link here.

Preprocessing: a description of the dataset preprocessing. This is automatically included when the instance of `GingadoModel` has a preprocessing step in the pipeline. If the evaluation dataset for some reason necessitates a different preprocessing pipeline, then users are encouraged to report that.

Motivation: A brief description of why that dataset was chosen for training and for evaluation. When the dataset is not publicly available (or available to the audience of the model card), users are encouraged to input here some summary statistics that describe the data.

6.5.6 Ethical considerations

Some machine learning models in economics and finance might be deployed in settings where their usage might materially impact stakeholder outcomes. For example, credit models might direct lending towards one group vs the other (Fuster et al. 2022). For this reason, users developing models for deployment are strongly encouraged to consider the ethical implications, as well as to lay out explicitly where relevant the ethical choices when the model was being trained.

Suggested points include the following:

Data: whenever relevant, ethical aspects of the data collection could be noted in this section. Note that while the bulk of the discussion of ethics in machine learning and related data collection refers to activities such as image classifiers or other tasks that are not traditionally associated with studies in

economics and finance, a prominent case related to a widely-used benchmark dataset shattered the notion that economics and finance use cases can dispense with ethical considerations on data. Namely, serious ethical problems related to systemic racism in the so-called "Boston housing dataset" (Harrison and Rubinfeld 1978) have been shown by Carlisle n.d., and since then its use has been discouraged (unless if deployed to illustrate the ethical shortcomings in question).

The **potential impact on people** are also an encouraged field in the model documentation, along with the **risks**: what is the likelihood of the negative scenarios and their materiality on people if they do occur?

Mitigation: what are the measures that were taken during model training to mitigate any potential unwarranted negative effect stemming from the use of this model? Users are encouraged to list measures spanning from data collection practices to modelling techniques that are undertaken with the purpose of mitigating such ethical risks.

Use cases to avoid: if the model developer can think of any use cases that a third party could reasonably think to deploy the model, but should not do so on ethical grounds, this is the section where the develop could mention that and provide some background.

6.5.7 Caveats and recommendations

Caveats includes any notes from the developer to raise awareness of users around any known issues, situations or use cases that require awareness. **Recommendations** or notes describing any recommended usage can also be helpful to include. Users may also find it useful to add **Miscellaneous notes** to the documentation.

7 Time series functionalities

Since economics and financial datasets often involve the time dimension, it is worthwhile to revisit in this section some of the main functionalities associated with time series models. In addition to these functions, which should be valuable to the user on their own, *gingado* parses the time series with the goal of identifying its frequency, which in turn is important for the data augmentation part described in section 4, since the downloaded datasets should preferably be of the same frequency.

7.1 Date and time

While economics time series tend to work with data measured in days, months, quarters or years, financial datasets sometimes are measured in intraday time, ie hours, minutes or even seconds, miliseconds. These formats are all supported, by virtue of the date/time abilities of the underlying `pandas` package.

7.2 Data validation

gingado allows for choosing a certain date as the cutoff between training data and the validation and/or test datasets. This date can also be defined as the distance to the most recent date, reflecting the fact that some machine learning uses in economics and finance will probably be retrained periodically. It also allows the user to choose a specific percentage of the available *dates*.

7.3 Time fixed effects

In traditional estimation involving time series, many use cases include in the formula an estimation of the fixed effects of each of the time periods in the data. These fixed effects are a scalar representing the average value of the target variable for each unique realisation of the time period. With *gingado*, we leverage a pre-existing function in the `Fast.ai` library to decompose each period into its different components (the year, semester, quarter, month, week, day, day of the week...) as relevant. In turn, for each unique value of these time columns, *gingado* calculates its embedding, ie a n-length vector that describes each time period in more dimensions than the traditional fixed effects. n can be any positive integer larger than one fixed by the user; *gingado* includes a default value of $n = 4$.

8 API

A minimalistic example is shown below.

Listing 2: API example

```
>>> import gingado
>>> import pandas as pd
>>> df = pd.DataFrame("path/to/file.csv")
>>> model = GingadoModel(
...     data=df,
...     y_cols=['gdp_growth'],
...     cat_cols=['country', 'currency'],
...     cont_cols=['employment_growth', 'inflation', 'credit_growth'],
...     time_cols=['quarter'],
...     sdmx_sources=['BIS', 'IMF'],
...     data_commentary="This dataset was obtained from source XYZ"
... )
```

Once an instance of GingadoModel is created, behind the curtains it creates an instance of GingadoData, which contains the dataset, including a version with augmented data (see section 4.)

8.1 Inputting data

Data can be included in an instance of `GingadoModel` in two forms: either as a `pandas` DataFrame or as a path to a comma-separated values (CSV) file.

`pandas` offers multiple possibilities for reading files of many different formats, including more advanced formats associated with big data, such as parquet.

Important is that the dataset going into an instance of `GingadoModel` should be "tidy" (see Wickham 2014 for more details). The user is asked to identify which columns are numerical (ie, continuous), categorical, or denominating time, since they undergo different processing routes. The path for numerical columns is described in subsection 8.2 below; categorical columns are used to calculate embeddings, which are then processed similar to numerical variables; and the time columns are first expanded into the different components of time, as mentioned in subsection 7.3, before going the same route as categorical variables.

8.2 Data processing

gingado supports steps of data processing and feature engineering, after the data is cleaned and structured by the user. This is done via `scikit-learn`'s preprocessing methods.[14] Technically, they are the Python classes within the `preprocessing` module. In addition to the reliability of these preprocessing tools, another convenience is that once they are defined by the user, they can be fit inside a so-called "pipeline": a collection of preprocessing steps that ensures that the training data and the validaton data are dealt with separately, thus avoiding creation of spurious results known as leakage. *gingado* uses by default the following preprocessing steps for the numerical columns with continuous values:

1. `MaxAbsScaler`, which scales the variables to be between -1 and 1.
2. `PolynomialFeatures`, which multiplies variables by themselves and creates iteractions between variables.

Three advantages of using `scikit-learn`'s functionalities for data processing are:

- Ability to automatically use only the training data to calibrate the preprocessing steps (ie, standardisation), and do it in a consistent way, ie avoid leaking training data into the validation or test datasets.
- Ability to save the calibrated preprocessing step (and to automatically identify it, to include in the model documentation)
- Possibility to use a broad range of preprocessing tools, from a library that is widely used in machine learning.

[14] An overview of the preprocessing methods available, as well as instructions to how the user can calculate their own, is found here: `https://scikit-learn.org/stable/modules/preprocessing.html`

8.3 Defining the model

As mentioned in section 5, *gingado* already provides the user a reasonable benchmark model to be used as a baseline. But it can be expected that many users will want to further experiment with models to try to beat the benchmark's performance. They can do that in two ways: "inside" and "outside *gingado*".

8.3.1 Models inside *gingado*

Users can choose to send in one or more model architectures to an instance of `GingadoModel`. After the models are fitted, their performance on the test dataset is compared to that of the benchmark model, and the user is informed of the result. The class `GingadoModel` keeps a record at all times of what is the best model trained so far, so that the user feel safe to experiment with different model architectures or parameters due to the possibility of going back to the best model so far.

The user might also decide to use model ensembles, ie to combine different models. This is also made easy by *gingado*: the user only needs to send in a python dictionary containing each component model's name (it should be unique compared to the other models in the same instance of `GingadoModel`), its backend (ie, whether `scikit-learn`, `XGBoost`, `PyTorch` or `TensorFlow`), parameters (themselves structured as a dictionary where the keys are the parameter names) and a weight for the ensemble. If the value associated with the weight key is empty for all models, they are weighted equally.

8.3.2 Models outside *gingado*

The user can also choose to use an instance of `GingadoModel` to preprocess and augment their original dataset, and use the resulting data to train a model that is not created as part of the instance of `GingadoModel`. This is training a model "outside *gingado*". In this case, the user leverages the data-related functionalities of *gingado*, perhaps in models that are not yet integrated in this library, or in more advanced APIs that require an extreme degree of flexibility and control.

That alternative is obviously completely fine and should work well, since ultimately the data is being output in formats that compatible with the major machine learning libraries. The only warning to users training "outside" is that they be reminded to document their models properly.

9 Deploying models built with *gingado*

Because the underlying models are saved in standard machine learning model formats from the different backends supported (see section 3), *gingado* models can be deployed across a variety of use cases:

- in web apps

- in mobile apps (both Android and iOS platforms support the use of machine learning models; in the case of iOS, the library coremltools can be used to translate between those models and an iOS-compatible trained model.)
- as part of compiled programmes, etc.

Users that seek to deploy models are encouraged to see the instructions for the specific backend that powers their instance of `GingadoModel`

References

Bai, Junjie, Fang Lu, Ke Zhang, et al. (2019). *ONNX: Open Neural Network Exchange*. `https://github.com/onnx/onnx`.

Buitinck, Lars et al. (2013). "API design for machine learning software: experiences from the scikit-learn project". In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122.

Carlisle, M (n.d.). *racist data destruction?* `https://medium.com/@docintangible/racist-data-destruction-113e3eff54a8`. Last accessed: 2020-02-08.

Chen, Tianqi and Carlos Guestrin (2016). "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: Association for Computing Machinery, pp. 785–794. ISBN: 9781450342322. DOI: `10.1145/2939672.2939785`. URL: `https://doi.org/10.1145/2939672.2939785`.

Fuster, Andreas et al. (2022). "Predictably Unequal? The Effects of Machine Learning on Credit Markets". In: *The Journal of Finance* 77.1, pp. 5–47. DOI: `https://doi.org/10.1111/jofi.13090`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1111/jofi.13090`. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1111/jofi.13090`.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Gorishniy, Yury et al. (2021). *Revisiting Deep Learning Models for Tabular Data*. arXiv: `2106.11959 [cs.LG]`. URL: `https://proceedings.neurips.cc/paper/2021/hash/9d86d83f925f2149e9edb0ac3b49229c-Abstract.html`.

Harrison, David and Daniel Rubinfeld (Mar. 1978). "Hedonic housing prices and the demand for clean air". In: *Journal of Environmental Economics and Management* 5, pp. 81–102. DOI: `10.1016/0095-0696(78)90006-2`.

Howard, Jeremy and Sylvain Gugger (2020). "Fastai: A Layered API for Deep Learning". In: *Information* 11.2. ISSN: 2078-2489. DOI: `10.3390/info11020108`. URL: `https://www.mdpi.com/2078-2489/11/2/108`.

Martín Abadi et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: `https://www.tensorflow.org/`.

Mitchell, Margaret et al. (2018). "Model Cards for Model Reporting". In: *CoRR* abs/1810.03993. arXiv: 1810.03993. URL: http://arxiv.org/abs/1810.03993.

Paszke, Adam et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Thomas, Rachel and David Uminsky (2020). *The Problem with Metrics is a Fundamental Problem for AI*. arXiv: 2002.08512 [cs.CY].

Ushey, Kevin, JJ Allaire, and Yuan Tang (2022). *reticulate: Interface to 'Python'*. https://rstudio.github.io/reticulate/, https://github.com/rstudio/reticulate.

Wickham, Hadley (2014). "Tidy Data". In: *Journal of Statistical Software* 59.10, pp. 1–23. DOI: 10.18637/jss.v059.i10. URL: https://www.jstatsoft.org/index.php/jss/article/view/v059i10.