ML.NET: Machine Learning Toolkit for Software Developers

Matteo Interlandi [1] **Sergiy Matusevych** [1] **Markus Weimer** [1]

Abstract

Machine Learning is transitioning from an art and science into a technology available to every developer. In the near future, every application on every platform will incorporate trained models to encode data-based decisions that would be impossible for developers to author. This presents a significant engineering challenge, since currently data science and modeling are largely decoupled from standard software development processes.

In this demo we present ML.NET: a recently open sourced ML toolkit enabling .NET developers to author, train, evaluate, and deploy ML models. We showcase the life-cycle of model development and deployment through a Github issue classification problem.

1 Introduction

We are witnessing an explosion of new frameworks for building Machine Learning (ML) models (Abadi et al., 2016; cnt; pyt; Pedregosa et al., 2011; h2o). This profusion is motivated by the transition from ML as an art and science into a set of technologies readily available to every developer. An outcome of this transition is the abundance of applications that rely on trained models for functionalities that evade traditional programming due to their complex statistical nature. This unfolding future, where most applications make use of at least one model, profoundly differs from the current practice in which data science and software engineering are performed in separate and different processes. Furthermore, in current practice, models are routinely deployed and managed in completely distinct ways from other software artifacts. While typical software packages are seamlessly compiled and run on a myriad of heterogeneous devices, machine learning models are often relegated to be run as web services in relatively inefficient containers (Lee et al., 2018a;b). This pattern severely limits the kinds of applications one can build with machine learning capabilities.

At Microsoft, we have encountered this phenomenon early on and across a wide spectrum of applications and devices, ranging from services and server software to mobile and desktop applications running on PCs, Servers, Data Centers, Phones, Game Consoles and IOT devices. A ML toolkit for such diverse use cases, must satisfy several constraints but, most importantly, it has to capture the full prediction pipeline that takes a test example from a given domain (e.g., an email with headers and body) and produces a prediction that can often be structured and domain-specific (e.g., a collection of likely short responses). The requirement to encapsulate predictive pipelines is of paramount importance because it allows for effectively decoupling application logic from model development. Carrying the complete train-time pipeline into production provides a dependable way for building efficient, reproducible, production-ready models (Zinkevich).

In this demo we will present ML.NET (mld; Interlandi et al., 2018): an open source machine learning framework allowing .NET developers to author and deploy complex ML pipelines composed of data featurizers and state of the art machine learning models. Pipelines implemented and trained using ML.NET can be seamlessly embedded into .NET applications without any modification. Training and prediction, in fact, share the same code paths, and (as we will show) deploying a model into an application is as easy as importing ML.NET runtime and binding the inputs/output data sources. ML.NET's ability to capture full, end-to-end pipelines has been demonstrated by the fact that 1,000s of Microsoft's data scientists and developers have been using ML.NET over the past decade, infusing 100s of products and services (among which Windows, Bing, PowerPoint and Excel) with machine learning models used by hundreds of millions of users worldwide. In the demo, we will show how the Visual Studio team employs ML.NET for the automatic classification of Github issues. A simplified version of the demo can be found online on the ML.NET tutorials page. [1]

[1]Microsoft

Proceedings of the 2^{nd} SysML Conference, Palo Alto, CA, USA, 2019.

[1]https://docs.microsoft.com/en-us/dotnet/machine-learning/tutorials/

2 DEMONSTRATION DETAILS

Github allows developers to fill issues against an online repository when a bug is found, to request a missing feature, to track new ideas, etc. It is common in large repositories to have hundreds of issues running concurrently. To better organize the work, Github allows developers to label issues under categories usually referred to as *areas*. This labelling process is however manual, and it requires developers to sift through all the (eventually hundreds of) issues. The Visual Studio team employs ML.NET to automatically attach labels to issues, and therefore decrease the manual processing.

The demonstration will guide the attendants in the process of creating an application for the automatic classification of Github issues. The process can be summarized in 4 steps: (1) *authoring the data ingestion and featurization pipeline*; (2) *building and training the model*; (3) *evaluating the model*; and finally (4) *model deployment* in the application. The demo will be carried on a MacOS laptop machine with Visual Studio Code and .NET Core pre-installed. We plan to provide an additional Windows machine for attendants interested in creating and testing their own ML.NET pipelines.

Data Ingestion and Feauturization Pipeline. ML.NET is a .NET machine learning library that allows developers to build complex machine learning pipelines. Pipelines are often composed of multiple transformation steps that featurize and transform the raw input data. For this specific demo, we will use a publicly available Github issues dataset [2] as our input training data. This dataset is composed by a list of textual columns (Title of the issue, Area and a Description): we will show which options ML.NET provides to developers for text featurization, and we will compose a simple pipeline translating the input data into feature vectors understandable by ML models. We will also exploit this first step to introduce the main concepts representing data, data transformations, and trainable operators in ML.NET: namely the IDataView abstraction, and the concepts of *Transformer* and *Estimator*.

Building and Training the Model. Featurization pipelines are commonly followed by one or more ML models, either stacked or forming ensembles. Predicting the area in which an issue belongs to is a multi-class classification problem. In this step we will describe which models are available in ML.NET to solve this task, and we will use the previously generated featurization pipeline to train a model for automatically predicting issues' area. We will also show how commonly used techniques such as data caching can improve training performance.

Model Evaluation. ML.NET allows developers to directly evaluate the quality of models by using a test dataset and proper *evaluator* modules. In this step, we will use a multi-class evaluator to generate and print metrics related to the authored pipeline. We will also show how ML.NET makes easy to iterate over model parameters and evaluations until a "good enough" model is generated.

Model Deployment. In the final step we will save the final model and show how this can be loaded and embedded into an application reacting to Github issues. The model will be used to automatically predict the area the newly posted issues should be in. ML.NET facilitates the process of surfacing models into applications since the framework takes care of serializing / deserializing pipelines and related dependencies, without any user intervention.

[2] https://raw.githubusercontent.com/dotnet/samples/master/machine-learning/tutorials/GitHubIssueClassification/Data/issues_train.tsv

ACKNOWLEDGMENT

We would like to thank the ML.NET team for the hard work and suggestions on early versions of this demo.

REFERENCES

CNTK. https://github.com/Microsoft/CNTK.

H2O. https://github.com/h2oai/h2o-3.

ML.NET. http://github.com/dotnet/machinelearning.

PyTorch. https://pytorch.org/.

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., and et al. Tensorflow: A system for large-scale machine learning. In *OSDI 16*, pp. 265–283, 2016.

Interlandi, M., Matusevych, S., Bilenko, M., Amizadeh, S., Zahirazami, S., and Weimer, M. Machine Learning at Microsoft with ML.NET. *NIPS SysML Workshop*, 2018.

Lee, Y., Scolari, A., Chun, B., Santambrogio, M. D., Weimer, M., and Interlandi, M. PRETZEL: opening the black box of machine learning prediction serving systems. In *OSDI 18*, pp. 611–626, 2018a.

Lee, Y., Scolari, A., Chun, B.-G., Weimer, M., and Interlandi, M. From the Edge to the Cloud: Model Serving in ML .NET. *IEEE Data Ing. Bull.*, 2018b.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., and et al. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12:2825–2830, November 2011. ISSN 1532-4435.

Zinkevich, M. Rules of machine learning: Best practices for ML engineering. https://developers.google.com/machine-learning/rules-of-ml.