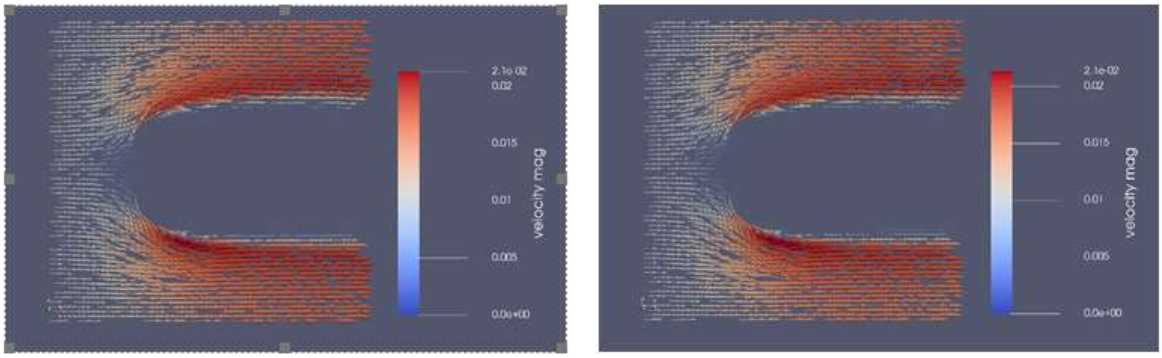
2019 UberCloud Compendium of Case Studies

on

Artificial Intelligence



Simulated flow field (left) and the 1000x faster predicted flow field (right) generated by AI.

Sponsored by:

Hewlett Packard Enterprise

intel

HPC wire

June 25, 2019

UberCloud

https://www.TheUberCloud.com

UberCloud AI Case Studies

More than 210 HPC cloud experiments, 100 case studies, and a ton of hands-on experience gained in High Performance Computing (HPC) in the Cloud, that's the harvest of six years of UberCloud* HPC Experiments, enabling us to measure cloud computing progress, objectively. Looking back these six years, at our first 50 cloud experiments in 2012, 26 of them failed or didn't finish, and the average duration of the successful ones was about three months. Six years later, in 2019, looking at our last 50 cloud experiments, none of them failed anymore; and the average duration of these experiments is now just about three days. That includes defining the application use case, preparing and accessing the engineering application software in the cloud, uploading the engineer's data, running simulation jobs (interactive and batch), evaluating the data via remote visualization, transferring the results back on premise, writing and publishing a case study.

The goal of the UberCloud* Experiment is to perform engineering simulation experiments in the HPC cloud with real engineering applications, in order to understand the roadblocks to success and how to overcome them, and to help **educate our engineering community** about the relatively young field of HPC in the Cloud. Our UberCloud* Compendiums of Case Studies (10 e-books so far) are a way of sharing these results with our broader community of engineers and scientists and their service providers.

Our community of engineers and scientists is currently joining the new wave of artificial intelligence (AI). According to IDC the expected world-wide investment in AI is 19.1 billion USD in 2018, an increase of 54.2% compared to 2017. Industry segments that invest aggressively in projects that use AI software capabilities will drive the investments to 52.2 billion USD by 2021, an average CAGR of 46.2% in the time interval between 2016 and 2021. In addition, the three largest use cases for AI in discrete manufacturing are expected to be predictive maintenance, quality management, and recommendations systems.

This 20th UberCloud* Compendium is dealing with AI, in an **introductory and educational** way. For helping to educate our engineering community in this relatively young field of AI in Engineering we have selected four use cases, about predictive maintenance, and deep learning and computational fluid dynamics.

We are very grateful for all the support for our UberCloud* AI Experiments by our technology partners **Hewlett Packard Enterprise and Intel**, and our Media Sponsor **HPCwire**.

Wolfgang Gentzsch, Burak Yenier, and Joseph Pareti
The UberCloud, Los Altos, CA, June 2019

**) UberCloud is the online community, marketplace, and application container factory, where engineers discover, try, and buy HPC as a Service, on demand. Engineers can explore how to use this additional computing power to solve their demanding problems, better, faster, cheaper, and identify roadblocks and solutions jointly with our engineering and scientific community. Learn more about the UberCloud and our services at: http://www.TheUberCloud.com*.

Table of Contents

2 **Foreword:** UberCloud AI Case Studies

4 **Team 211:** Deep Learning for Steady-State Fluid Flow Prediction in the Advania Data Centers Cloud

10 **Team 212:** Demonstrating a Machine Learning Model for Predictive Maintenance on Microsoft Azure, Part I

20 **Team 213:** Deep Learning in Computational Fluid Dynamics in the Microsoft Azure Cloud

28 **Team 214:** Demonstrating a Machine Learning Model for Predictive Maintenance on Microsoft Azure, Part II

37 **Join the UberCloud Experiment**

Project support and dissemination of this compendium has been generously sponsored by:

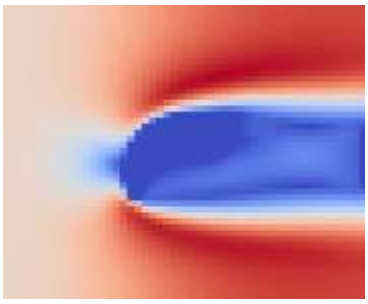
Hewlett Packard Enterprise

intel

HPC wire

Team 211

Deep Learning for Steady-State Fluid Flow Prediction in the Advania Data Centers Cloud



"The overhead of creating high volumes of samples can be effectively compensated by the high-performance containerized computing environment provided by UberCloud and Advania."

1 MEET THE TEAM

End-User: Jannik Zuern, Renumics GmbH, Karlsruhe, Germany
Software Provider: OpenFOAM open source CFD software
Resource Provider: Advania Data Centers Cloud, Iceland
HPC and AI Experts: Stefan Suwelack, Markus Stoll, and Jannik Zuern, Renumics; Joseph Pareti, AI Consultant; and Ender Guler, UberCloud Inc.

2 USE CASE

Solving fluid flow problems using Computational Fluid Dynamics (CFD) is demanding both in terms of computer power and in terms of simulation duration. Artificial neural networks (ANN) can learn complex dependencies between high-dimensional variables. This ability is exploited in a data-driven approach to CFD that is presented in this case study. An ANN is applied in predicting the fluid flow given only the shape of the object that is to be simulated. The goal of the approach is to apply an ANN to solve fluid flow problems to significantly decrease time-to-solution while preserving much of the accuracy of a traditional CFD solver. Creating a large number of simulation samples is paramount to let the neural network learn the dependencies between simulated design and flow field around it.

This project between Renumics GmbH and UberCloud Inc. explores the benefits of additional cloud computing resources that can be used to create a large amount of simulation samples in a fraction of the time a desktop computer would need to create them. In this project, we want to explore whether the overall accuracy of the neural network can be improved when more samples are being created in the UberCloud Container und then used during the training of the neural network. UberCloud kindly provided the cloud infrastructure, a CentOS Docker container with an OpenFOAM installation, and additional tech support during the project development.

3 WORKFLOW OVERVIEW

In order to create the simulation samples automatically, a comprehensive workflow was established.

As a **first step**, random two-dimensional shapes are created. These shapes have to be diverse enough to let the neural network learn the dependencies between different kinds of shapes and their respective surrounding flow fields.

In the **second step**, these shapes are meshed and added to an OpenFOAM simulation case template (Fig. 1). This template is simulated using the steady-state solver OpenFOAM solver simpleFOAM.

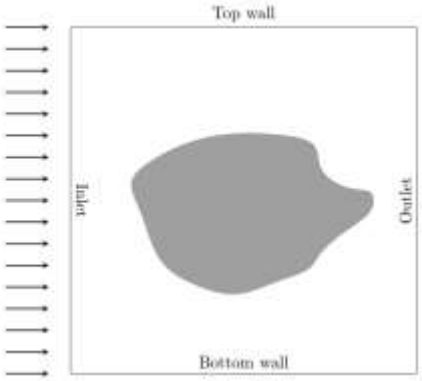


Figure 1: Simulation setup. The flow enters the simulation domain through the inlet, flows around the arbitrarily shaped obstacle (grey shade) and leaves the simulation domain through the outlet.

The **third step** post-processed the simulation results using open-source visualization ParaView. The flow-fields are resampled on a rectangular grid to simplify information processing by the neural net.

In the **fourth step** the simulated design and the flow fields are fed into the input queue of the neural network which, after training, is able to infer a flow field merely from seeing the to-be-simulated design.

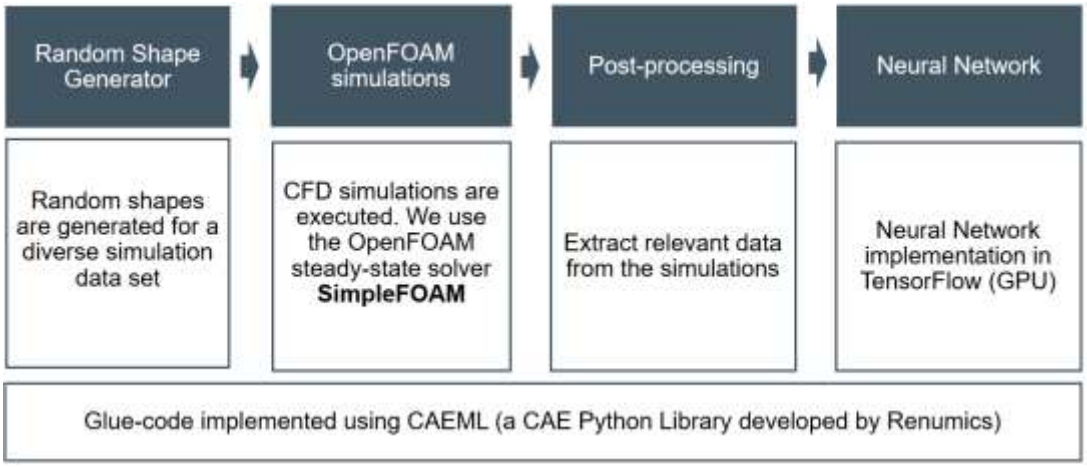


Figure 2: Four-step Deep Learning workflow.

Hardware specs

The hardware specs of the Advania Data Centers compute node hosting UberCloud's container are:

- 2 x 16 core Intel Xeon CPU E5-2683 v4 @ 2.10 GHz and 251 GB memory, no GPU

The hardware specs of the previously used desktop workstation are as follows:

- 2 x 6 core Intel i7-5820K CPU @ 3.30 GHz, and 32 GB memory
- GPU: GeForce GTX 1080 (8GB GDDR5X memory)

4 RESULTS

Time needed to create samples

As a first step, we compared the time it takes to create samples on the desktop workstation computer with the time it takes to create the same number of samples on the UberCloud container. Figure 3 illustrates the difference in time it took to create 10,000 samples. On the desktop computer

it took 13h 10min to create these 10,000 samples. In the UberCloud OpenFOAM container in the Advania Data Centers Cloud, it took about 2h 4min to create 10,000 samples, which means that a speedup of 6.37 could be achieved using the UberCloud container.

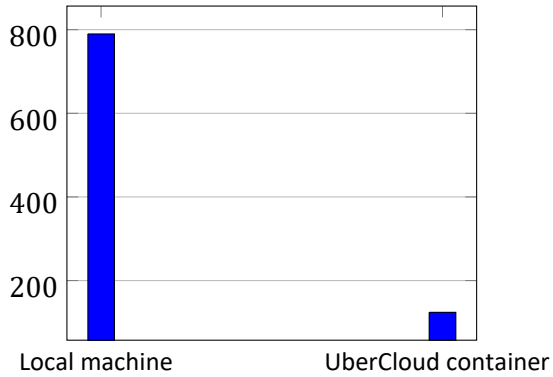


Figure 3: Comparison (in minutes) between Local machine and UberCloud container.

Neural Network performance evaluation

A total of 70,000 samples were created. We compare the losses and accuracies of the neural network for different training set sizes. In order to determine the loss and the accuracy of the neural network, we first must define, what these terms actually mean.

- Performance for generating the flow field data set and Tensorflow training

Setup	2-D external flow	3-D internal flow
Time for 10.000 simulations	13.2 h	152.5 h
Time for training	23.7 h	48.5 h

- Neural network prediction of flow field

Setup	2-D external flow	3-D internal flow
Time for CFD solver	4.7 s	55.0 s
Time of neural network prediction	3 ms	120 ms
Speedup factor with deep leaning	**1566**	**458**

Figure 4: Performance and speedup of flow simulations with neural network prediction.

Definitions

Loss: The loss of the neural network prediction describes how wrong the prediction of the neural network was. The output, or prediction, of the neural network in our project is a $N \times M \times 2$ tensor since the network tries to predict a fluid flow field with N elements in x-direction, M elements in y-direction, and two flow velocity components (velocity in x-direction and velocity in y-direction). A mean-squared-error metric was used to calculate the loss l:

$$l = \frac{1}{2} \sum_{d=0}^{1} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} (v_{dij} - \bar{v}_{dij})^2 \quad (1)$$

where v_{dij} denotes the ground-truth velocity component in dimension *d* at the grid coordinates (*i,j*), $\overline{v}_{dij}$ denotes the predicted velocity component at the same position and in the same dimension. The goal of every machine learning algorithm is to minimize the loss of the neural network using numerical optimization schemes such as Stochastic Gradient Descent. Thus, a loss of 0.0 for all samples would mean that every flow velocity field in the dataset is predicted perfectly.

Accuracy: In order to be able to make sensible statements about the validity of the prediction of the neural network, metrics have to be defined that describe the level of accuracy that the neural network achieves. In general, the accuracy of a neural network describes how accurate the prediction of the neural network was. While the loss of a neural network is the metric that is being minimized during training, a small prediction loss does not necessarily mean that the corresponding prediction is physically meaningful. In general, however, a small prediction loss usually corresponds with a high accuracy. Different measurements of how accurate the outputs of the neural network are needed to express the validity of the predictions. A highly accurate prediction should have high values for all formulated accuracy measurements and a low loss at the same time. These accuracies can have values between 0.0 and 1.0, where an accuracy of 0.0 indicates that the prediction of the neural network does not at all coincide with the ground truth flow metric that is examined, and an accuracy of 1.0 means that the prediction coincides perfectly with the ground truth flow metric. Bear in mind that a low loss does not necessary cause high accuracy and vice versa. However, the two measurements are usually correlated.

In this study, two different accuracies were evaluated: Divergence accuracy and Drag accuracy:

- **Divergence accuracy:** Numerical CFD solvers aim to find a solution to the continuity equation and the momentum equation. For an incompressible fluid, the continuity equation dictates that the divergence of the velocity vector field is zero for every point in the simulation domain. This follows the intuition that at no point in the simulation domain fluid is generated (divergence would be greater than zero) or ceases to exist (divergence would be smaller than zero). By design, the Finite Volume Method preserves this property of the fluid even in a discretized form. A data-driven approach should as well obey this rule.

- **Cell accuracy:** The number of correctly predicted grid cells in the two- or three-dimensional grid yields an intuitive metric for how well the neural network predicts fluid flow behavior. As the network will never be able to predict the fluid flow velocity down to the last digit of a floating-point number, the following approach is proposed: If the relative error between the network prediction and the actual flow velocity is smaller than 5%, the respective grid cell is declared as predicted correctly. The cell accuracy can be calculated by counting the number of correctly predicted grid cells and dividing the results by the total number of grid cells.

5 TRAINING RESULTS

The generated samples are divided into the training and validation datasets. The training- and validation loss for different numbers of training samples was evaluated. Concretely, the neural net was trained three times from scratch with 1,000, 10,000, and 70,000 training samples respectively.
The following training parameters were used for all neural network training runs:

- Batch size: 32
- Dropout rate: 0.5
- Learning rate: 5×10^{-4}

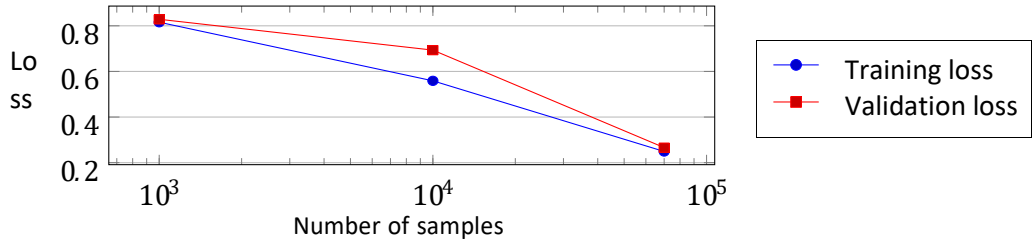


Figure 5: Loss after 50,000 training steps.

It can be observed that both training- and validation losses are lowest for the 70k samples training and are highest for the 1k training samples. The more different samples the neural network processes during the training process the better faster it is able to infer a flow velocity field from the shape of the simulated object suspended in the fluid. The validation loss tends to be higher than the training loss for all tested numbers of samples, which is a typical property of machine learning algorithms. Figure 6 shows the loss after 300,000 training steps:

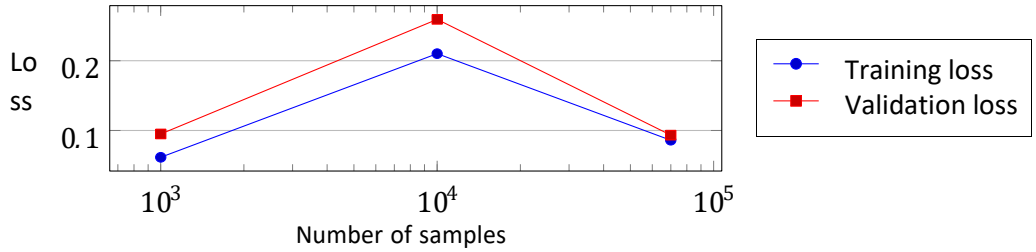


Figure 6: Loss after 300,000 training steps.

Surprisingly, the final training- and validation losses for the 70k samples training session are as low as the losses for the 1k samples training session. Generally speaking, no clear tendency towards lower losses when increasing the set of the training samples could be observed. This result is somewhat surprising since we expected the final losses at the end of the training process to show a similar tendency towards lower losses for higher numbers of samples. We assume that the number of samples does not heavily influence the final loss for extensive training sessions with many hundreds of thousand training steps. Finally, in Figure 7 the divergence and grid accuracies are visualized.

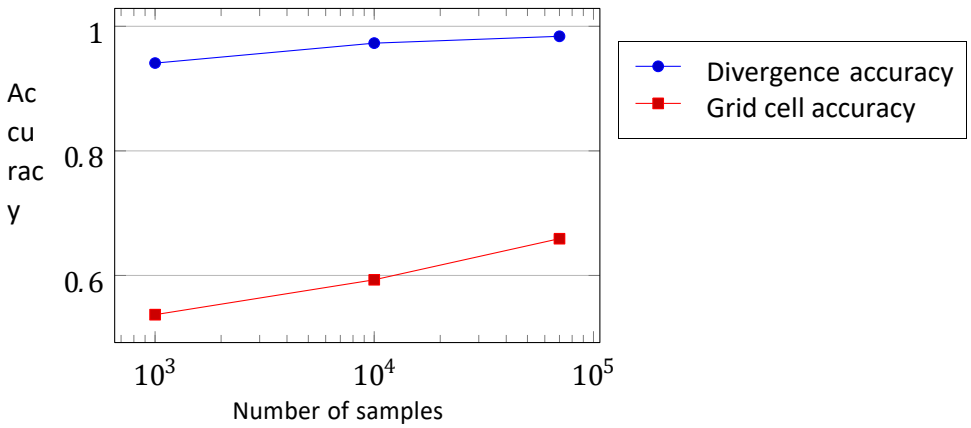


Figure 7: Validation accuracies after training.

Both the divergence accuracy and the grid cell accuracy show higher values for larger numbers of samples. While the divergence accuracy shows overall high values going from 0.94 for 1,000 samples to 0.98 for 70,000 samples, the grid cell accuracy also increases from a value of 0.53 for 1,000

samples to a value 0.66 for 70,000 samples. To recap: a grid accuracy of 0.66 means that approximately two thirds of all velocity grid cells were predicted correctly within 5% relative error to the correct value.

Figure 8 illustrates the difference between the ground truth flow field (left image) and the predicted flow field (right image) for one exemplary simulation sample after 300,000 training steps. The arrow direction indicates the flow direction and the arrow color indicates the flow velocity. Visually, no difference between the two flow fields can be made out.

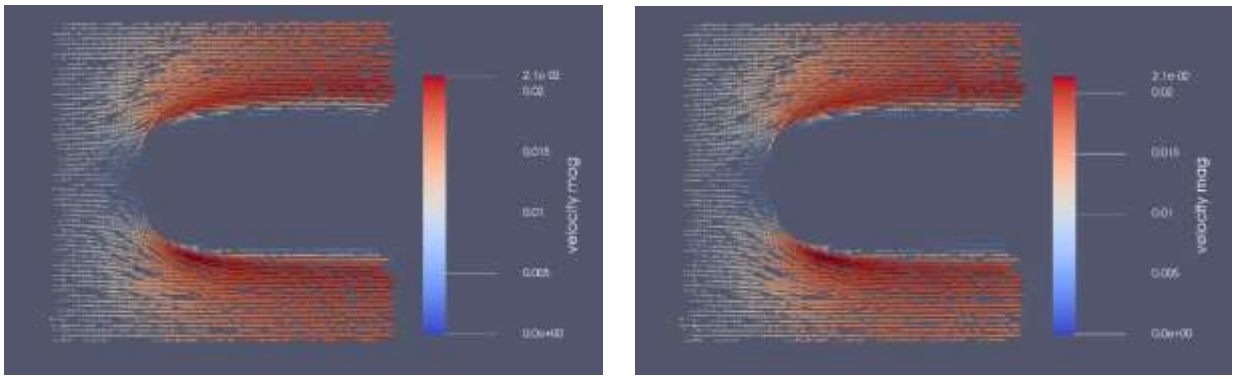


Figure 8: Exemplary simulated flow field (left image) and predicted flow field (right image).

CONCLUSION

We were able to prove a mantra amongst machine learning engineers: *The more data the better*. We showed that the training of the neural network is substantially faster using a large dataset of samples compared to smaller datasets of samples. Additionally, the proposed metrics for measuring the accuracies of the neural network predictions exhibited higher values for the larger numbers of samples. The overhead of creating high volumes of additional samples can be effectively compensated by the high-performance containerized (based on Docker) computing node provided by UberCloud on the Advania Data Centers Cloud. A speed-up of more than 6 compared to a state-of-the-art desktop workstation allows creating the tens of thousands of samples needed for the neural network training process in a matter of hours instead of days.

In order to train more complex models (e.g. for transient 3D flow models) much more training data will be required. Thus, software platforms for training data generation and management as well as flexible compute infrastructure will become increasingly important.

Case Study Author – Jannik Zuern, Renumics GmbH

Team 212

Demonstrating a Machine Learning Model for Predictive Maintenance on Microsoft Azure

Figure 1: Infographics: https://goo.gl/cv7vLp

> *"This Machine Learning study is for predictive maintenance demonstration and learning purposes, and may be used as a baseline for the reader's custom applications."*

MEET THE TEAM

End-User: Joseph Pareti, Artificial Intelligence Consultant
Software Provider: Open source Predictive Maintenance template provided by the Microsoft Azure Team
Resource Provider: Microsoft Azure Cloud
UberCloud Support: Wolfgang Gentzsch, President, The UberCloud
Microsoft Support: Yassine Khelifi, Support Escalation Engineer at Microsoft.

INTRODUCTION: HOW TO USE THIS DOCUMENT

The main purpose of this work is to create awareness on what is possible with machine learning in the manufacturing industry, and hence we picked a predictive maintenance use case. Next, there are 2 classes of users:

1. ***Developers need to follow through all steps described in the next paragraphs.*** You will do data ingestion, features engineering and train a Machine Learning model that can then be used to predict a machine likelihood of failure within a specified time window. If you are a developer, you need to work **all** notebooks. You will also need a front-end system (like a Windows PC running the Azure Machine Learning Workbench) and a back-end system such as a Data Science Virtual Machine in the Azure Cloud.
2. ***End-users who want to just consume the application as a service*** will need the assets files and a Docker image running on e.g. an Ubuntu server. Please send a request for the assets files to joepareti54@gmail.com or 0049 1520 1600 209.

USE CASE

The Predictive Maintenance model described in this report is open source and can be applied to different equipment types for which telemetry data and maintenance data records are available. A demo version is described in the following paragraphs and it is

based on specific prerequisites. Customization of the model to real customers' case[1] is out of scope for this report but can be done on a project basis, explained in the conclusion/ recommendations paragraph.

The demo version is based on the following assumptions: 4 machine types are considered, each machine has 4 components, and there is *telemetry* data available on voltage, vibration, speed, etc. as well as maintenance records (indicating when components were replaced on what machine), error logs (not necessarily implying failure), machine characteristics, and how long each machine has been in service. The model is built in 4 stages each of which is a Jupyter Notebook:

1. Data ingestion
2. Feature engineering
3. ML model
4. Operationalization

Notebook 1, Data ingestion, is about accessing datasets and converting data into py-spark data-frames that are stored in Azure storage container (AKA blob storage), so that the data is accessed in the next notebook. The demo version uses pandas as source data from a SQL server in github, but it can obviously be replaced with customer's data.

Notebook 2, Feature engineering, loads the data sets created in Notebook 2 from an Azure storage container and combines them to create a single data set of features (variables) that can be used to infer a machine health condition over time. The notebook steps through several feature engineering and labeling methods to create this data set for use in the predictive maintenance machine learning solution.
The goal is to generate a single record for each time unit within each asset. The record combines features and labels to be fed into the machine learning algorithm.
Predictive maintenance takes historical data, marked with a timestamp, to predict current health of a component and the likelihood of failure within some future window of time. These problems can be characterized as a *classification method* involving *time series* data. Time series, since we want to use historical observations to predict what will happen in the future. Classification, because we classify the future as having a likelihood of failure.

Notebook 3, The ML model, uses the labeled feature data set constructed in Notebook 2, it loads the data from the Azure Blob container and splits it into a training and test data set. We then build a machine learning model (a decision tree classifier or a random forest classifier) to predict when different components within our machine population will fail. Two different classification model approaches are available in this notebook:

- **Decision Tree Classifier**: Decision trees and their ensembles are popular methods for the machine learning tasks of classification and regression. Decision trees are widely used since they are easy to interpret, handle categorical features, extend to the

[1] Customization may mean working with different datasets (this is the easiest step), or changing the data structure and ML model according to customers' needs.

multiclass classification setting, do not require feature scaling, and are able to capture non-linearities and feature interactions.

- **Random Forest Classifier**: A random forest is an ensemble of decision trees. Random forests combine many decision trees in order to reduce the risk of overfitting. Tree ensemble algorithms such as random forests and boosting are among the top performers for classification and regression tasks.

Notebook 4, Operationalization is about deploying the model into a Docker container for users to consume it as-a-web service. We propose an Ubuntu implementation.

Finally, some test cases have been run, and the relevance of input data on machine failures is shown.

SYSTEM ARCHITECTURE

The Predictive Maintenance application consists of three components**:**

1. A front-end application, Azure Machine Learning Workbench (AML), that runs on a local system, in my case a Windows 10 laptop

2. A back-end system, a Data Science Virtual Machine (DSVM) in the Azure cloud. The main advantage of working with a DSVM is that all required software components for ML are pre-installed and maintained by Microsoft. Moreover, you can configure the DSVM to fulfill your needs, in my case I selected the minimum required configuration, minimum cost, in terms of number of vCPUs, memory, local storage since the exercise was not intended to measure any performance data

3. A blob-storage account (in Azure) to host the intermediate data that are transferred from one notebook to the other.

The front-end and back-end applications interact over the internet; a stable standard configuration Wireless LAN has proven to be adequate for the job. If your network is unstable, you may face some issues that are described in Figure 4 in the Appendix of this report.

DEVELOPER VIEW

If you are a developer, you need to go through all steps starting with data ingestion, next you do feature engineering, and finally you train the model. The basic tool that will guide you all along is the Jupyter Notebook that will be started in AML.

You create a new project in AML, and then select the *General Predictive Maintenance* scenario[2]; AML comes with on-line documentation to guide you through the steps. You need to first ensure that the DSVM is up and running so that the front-end can attach to it via remote Docker. On the front-end, you need a CLI: you can use power shell (which I recommend), or the DOS command interface to do the basic startup commands:

[2] Microsoft provides 2 Predictive Maintenance templates: the other one is Deep Learning for PdM which is however oversimplified.


The CLI command starts a local Jupyter notebook server and opens the default browser tab pointing to the project root directory. The example notebooks are stored in the `code` directory. The predictive maintenance example runs these notebooks sequentially as numbered, starting with the Data Ingestion process in the `code\1_data_ingestion.ipynb` notebook. Whe you first open a notebook, the server will prompt you to connect to a kernel. Use the kernel associated with the docker container under [Project_Name]_Template (Connection_Name).

The example notebooks are broken into separate chunks of work:

- `code/1_data_ingestion.ipynb` download and prepare raw data
- `code/2_feature_engineering.ipynb` create model features and target label
- `code/3_model_building.ipynb` build and compare machine learning model
- `code/4_operationalization.ipynb` deploy a model for production scenario

Each notebook will store intermediate results in an Azure Blob storage container to facilitate a seamless workflow. In order to do this, we require you're storage container access keys to be copied into each notebook. You can select a storage container in the https://portal.azure.com. Search for a `storage account` you'd like to use. Select the `access keys` item, and copy the `[ACCOUNT_NAME]` and one of the `[ACCOUNT_KEYS]` into the notebook code chunk:

```
# Enter your Azure blob storage details here
ACCOUNT_NAME = "<your blob storage account name>"

# You can find the account key under the _Access Keys_ link in the
# [Azure Portal](portal.azure.com) page for your Azure storage container.
ACCOUNT_KEY = "<your blob storage account key>"
```

Figure 1: Preparation steps on the AML front-end application (please enlarge your document).

RESULTS

The result of this study is a demo version that runs in a Docker container. Please contact me (joepareti54@gmail.com) if you want to test it yourself. In addition, I can provide workshops on AI applications for CAE and manufacturing.

PERFORMANCE BENCHMARKING

This is out-of-scope for the current project. On my DSVM, the by far most time-consuming part of the application is Notebook 2 (feature engineering), which takes 71 minutes to complete: table joins and label construction run on all 4 vCPUs in the DSVM. Notebook 3, model training, takes approximately 10 minutes. If you plan to use a larger data-set we will need to size the system, preferably with GPUs.

BENEFITS

In general, the benefits of avoiding down time of costly equipment, avoiding loss of business due to poor response time, or damage to a provider's reputation due to service outages, are self-explanatory. Machine Learning is an effective technology to accomplish those goals. According to Forbes[3], improving preventative maintenance and Maintenance, Repair and Overhaul (MRO) performance with greater predictive accuracy to the component and part-level is *one of the 10 ways ML is revolutionizing manufacturing.*

There are several implementations of Predictive Maintenance (PdM) using ML techniques out there. The following lists some PdM applications I came to know in a short space of time:

1. Rapidminer
2. Mathworks for Predictive Maintenance
    - Toolbox for tasks such as RUL estimation, condition parameters design, statistical methods, ML, features extraction from data, label construction for supervised training, etc.
    - *Toolbox* means the user must build the application
    - Use cases for rolling bearings, gearbox, pumps, etc.
    - Ability to simulate failure data using number crunching code from Mathworks

[3] https://www.forbes.com/sites/louiscolumbus/2016/06/26/10-ways-machine-learning-is-revolutionizing-manufacturing/#12fa1af328c2

3. Vargroup
4. Microsoft
5. General Electric

The benefits of working with the PdM model presented in this report instead of starting from scratch are:

(i) open source code,
(ii) extensive debugging and testing on the demo version, and
(iii) a core team of experts has been identified.

CONCLUSION & RECOMMENDATIONS

The subject matter of this report is an open source implementation of PdM, as such customers have free access to the code. Users are encouraged to approach us, and test the demo version "as-is" to evaluate how well it models their environment.

Next, we could work in a team (including the customer, UberCloud, myself, Microsoft and Nvidia) to scope the project and create a Statement of Work (SOW): this will also specify what data, data structures and interfaces are needed. In some cases, it may be possible to just acquire and integrate customer's data[4]. In the SOW, we will also investigate what ML models are more suitable to the customer's use case. The next step will be a proof of concept prior to production-ready applications.

FINAL UPDATE

Microsoft has released an updated version of its predictive maintenance demo version as part of their GitHub open source samples initiative: this version promises the following new features:

- Training and operationalization using Azure Machine Learning Service
- Model training on Azure Databricks cluster (gaining 1 order of magnitude in speed)
- Prediction Serving using Azure container instance
- Minor changes include:
  - Replacement of pandas data frame manipulation to spark data
  - Addition of a features (variables) importance plot
  - Slight modification to existing Exploratory Data Analysis (EDA) plots.

While our plan going forward calls for testing this new version, at this time we cannot publish any usage experience on it. If you – the reader – are interested in a joint exploration of this new version please contact us.

From an initial look at the new repository, it appears that the functional parts of the application are aligned with what we have already reported in this case study.

Case Study Authors – Joseph Pareti and Yassine Khelifi

[4] Some advice here: https://gallery.azure.ai/Collection/Predictive-Maintenance-Modelling-Guide-1

APPENDIX: Implementation notes: Azure resources

This Appendix is mostly useful for developers and end-users who want to work with this application, and for technical staff that plan to use Azure services for data science. We'll review the details of the configuration in use. First, if not yet done, you need to subscribe with Azure. Next, you will access the Azure dashboard, and then your Data Science Virtual Machine (DSVM) and Azure blob storage; Microsoft will bill you monthly on a usage basis:[5]

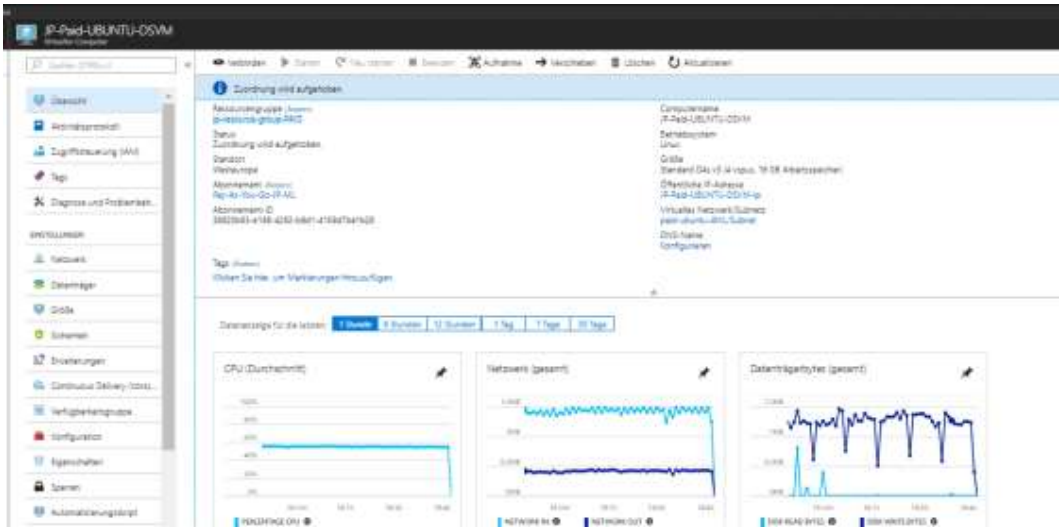


Figure 2: Overview of the DSVM in the Azure cloud

The next diagram shows the resources available in the DSVM:

```
joepareti54@JP-Paid-UBUNTU-DSVM:~$
joepareti54@JP-Paid-UBUNTU-DSVM:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 16.04.4 LTS
Release:        16.04
Codename:       xenial
joepareti54@JP-Paid-UBUNTU-DSVM:~$ grep MemTotal /proc/meminfo
MemTotal:       16404012 kB
joepareti54@JP-Paid-UBUNTU-DSVM:~$ cat /proc/cpuinfo | grep processor | wc -l
4
joepareti54@JP-Paid-UBUNTU-DSVM:~$ df
Filesystem     1K-blocks     Used Available Use% Mounted on
udev             8186556        0   8186556   0% /dev
tmpfs            1640404     8864   1631540   1% /run
/dev/sda1       50758760 38260240  12482136  76% /
tmpfs            8202004        0   8202004   0% /dev/shm
tmpfs               5120        0      5120   0% /run/lock
tmpfs            8202004        0   8202004   0% /sys/fs/cgroup
/dev/sdc1      103080224 20947340  76873672  22% /data
/dev/sdd1       52412420 43828928   8583492  84% /backupdata
/dev/sdb1       32895792    49084  31152660   1% /mnt
tmpfs            1640404        0   1640404   0% /run/user/1003
joepareti54@JP-Paid-UBUNTU-DSVM:~$
```

Figure 3: Putty terminal on the DSVM running Ubuntu

[5] To save money, it's a good idea to shut down your DSVM when you don't use it ☺ My monthly fees so far range from $60-$300 depending on consumption. More info at https://azure.microsoft.com/en-us/pricing/

Implementation notes: code changes to build the model

This is about code and configuration changes that were necessary to achieve a fully functional demo version. The figure below is relevant when you build the model:

Error message	Occurring in	Changes where	Code changes/notes
http 503 first byte timeout	Stage1, data ingestion in telemetry: exception depends on the internet connection between client and server	Stage 1 notebook	import socket\n, socket.setdefaulttimeout(300)\n,
As above	As above	AML Config file: *conda_dependencies.yml*	***tornado==4.5.3*** *(issue with tornado 5.0.2 is reported at https://github.com/jupyter/notebook/issues/3397)*
"code": "ServiceError", "message": "Received 404 from a service request",	Power shell	No changes needed	Please ignore as long as there are no errors in the notebooks

Figure 4: Code and configuration changes required to build the Predictive Maintenance (PdM) model

Implementation notes: how to run the model

If you are an end-user, you just need to consume the application as-a-service. At the end of Notebook 4, a zip file is created that contains the asset files needed to deploy the application as a service. This file can be provided by me. To run the application, you need a supported platform such as a system running Windows or Linux Ubuntu and with Docker installed in it. First create, then run the service on your system. To set the service up, follow the guidelines in "*how to operationalize the model*". Finally, a study (done by Microsoft support) shows important features that influence the likelihood of machine failure.

The following diagram shows the command to create the service and highlights the input "asset" files:

Figure 5: Command to create the service in Ubuntu; the input files are highlighted

After that you run the service: with the input variables defined below the system predicts no failure:
az ml service run realtime -i amlworkbenchpdmwebservice -d "{\"input_df\":

[{\"rotate_rollingmean_36\": 450.0384342542265, \"age\": 9, \"rotate_rollingstd_12\": 0.0, \"volt_rollingstd_36\": 0.0, \"volt_rollingstd_12\": 0.0, \"volt_rollingstd_24\": 0.0, \"pressure_rollingstd_36\": 0.0, \"error1sum_rollingmean_24\": 0.0, \"rotate_rollingmean_12\": 445.7130438343768, \"machineID\": 27, \"vibration_rollingmean_24\": 40.302192663278625, \"comp4sum\": 399.0, \"error4sum_rollingmean_24\": 0.0, \"pressure_rollingmean_36\": 99.1626730910439, \"pressure_rollingstd_12\": 0.0, \"vibration_rollingmean_12\": 39.69610732198209, \"comp3sum\": 444.0, \"error2sum_rollingmean_24\": 0.0, \"error5sum_rollingmean_24\": 0.0, \"pressure_rollingmean_24\": 100.42784289855126, \"pressure_rollingmean_12\": 103.46853199581041, \"vibration_rollingstd_36\": 0.0, \"vibration_rollingstd_12\": 0.0, \"rotate_rollingstd_36\": 0.0, \"vibration_rollingstd_24\": 0.0, \"volt_rollingmean_36\": 166.5072079613422, \"vibration_rollingmean_36\": 39.86004229336383, \"rotate_rollingstd_24\": 0.0, \"comp2sum\": 564.0, \"pressure_rollingstd_24\": 0.0, \"volt_rollingmean_24\": 166.69782028530955, \"comp1sum\": 504.0, \"volt_rollingmean_12\": 162.37456132546583, \"rotate_rollingmean_24\": 444.92430808877185, \"error3sum_rollingmean_24\": 0.0}, {\"rotate_rollingmean_36\": 452.58602482190344, \"age\": 9, \"rotate_rollingstd_12\": 7.358009183124642, \"volt_rollingstd_36\": 1.2113288898088435, \"volt_rollingstd_12\": 1.7162303092954838, \"volt_rollingstd_24\": 1.1327450423992658, \"pressure_rollingstd_36\": 0.360813923769749, \"error1sum_rollingmean_24\": 0.0, \"rotate_rollingmean_12\": 448.82482383859184, \"machineID\": 27, \"vibration_rollingmean_24\": 39.8762193116053, \"comp4sum\": 399.0, \"error4sum_rollingmean_24\": 0.0, \"pressure_rollingmean_36\": 99.18126302139088, \"pressure_rollingstd_12\": 1.3059590035299573, \"vibration_rollingmean_12\": 40.534215611846555, \"comp3sum\": 444.0, \"error2sum_rollingmean_24\": 0.0, \"error5sum_rollingmean_24\": 0.0, \"pressure_rollingmean_24\": 98.84197839575184, \"pressure_rollingmean_12\": 100.13428527324218, \"vibration_rollingstd_36\": 0.12802019423837702, \"vibration_rollingstd_12\": 0.5581845837178677, \"rotate_rollingstd_36\": 1.3063227195446807, \"vibration_rollingstd_24\": 0.26866456414969686, \"volt_rollingmean_36\": 166.8633264221902, \"vibration_rollingmean_36\": 39.83194043387068, \"rotate_rollingstd_24\": 6.2252625510326345, \"comp2sum\": 564.0, \"pressure_rollingstd_24\": 0.5506261833397947, \"volt_rollingmean_24\": 168.8315798036505, \"comp1sum\": 504.0, \"volt_rollingmean_12\": 169.6342364499553, \"rotate_rollingmean_24\": 455.68853459771736, \"error3sum_rollingmean_24\": 0.0}, {\"rotate_rollingmean_36\": 452.6366978657443, \"age\": 9, \"rotate_rollingstd_12\": 12.545522310840685, \"volt_rollingstd_36\": 0.4066137169118576, \"volt_rollingstd_12\": 1.9026812928919759, \"volt_rollingstd_24\": 1.388783538126311, \"pressure_rollingstd_36\": 0.40800640702349306, \"error1sum_rollingmean_24\": 0.0, \"rotate_rollingmean_12\": 462.5522453568429, \"machineID\": 27, \"vibration_rollingmean_24\": 39.48080284488274, \"comp4sum\": 398.0, \"error4sum_rollingmean_24\": 0.0, \"pressure_rollingmean_36\": 99.92595364177775, \"pressure_rollingstd_12\": 0.30398746497620055, \"vibration_rollingmean_12\": 39.21822301136402, \"comp3sum\": 443.0, \"error2sum_rollingmean_24\": 0.0, \"error5sum_rollingmean_24\": 0.0, \"pressure_rollingmean_24\": 98.70475189546528, \"pressure_rollingmean_12\": 97.5496715182615, \"vibration_rollingstd_36\": 0.06733738203927228, \"vibration_rollingstd_12\": 0.33150005427630586, \"rotate_rollingstd_36\": 0.726203655443797, \"vibration_rollingstd_24\": 0.2757178837764945, \"volt_rollingmean_36\": 164.9839282666808, \"vibration_rollingmean_36\": 39.16084871098736, \"rotate_rollingstd_24\": 2.2615583783043336, \"comp2sum\": 563.0, \"pressure_rollingstd_24\": 0.43573594568766316, \"volt_rollingmean_24\": 165.47787140830766, \"comp1sum\": 503.0, \"volt_rollingmean_12\": 168.0289231573457, \"rotate_rollingmean_24\": 454.4666253135592, \"error3sum_rollingmean_24\": 0.0}]}"

The following diagram shows the input variables to the model leading to a component-2 failure:

Figure 6: Sample run predicting machine failure (please enlarge your document)

Implementation notes: how to operationalize the model

To be able to run the application as a service on an Ubuntu server as described before, you must first deploy it according to the documentation. In addition, the following needs to be addressed:

1. You must register with all services per user's guide, in particular Microsoft.ContainerRegistry. In my initial attempt, I registered this service *after* having created the model management: this caused a conflict, and the solution was to delete and re-create the model management.
2. The sequence of commands that led to creation and deployment of the service on an Ubuntu server is as follows, and it differs from the documentation:
   $ az ml env delete -n pdmmodelmanagement1 -g jp-resource-group-PAID **# delete pre-existing model management**
   $ unzip o16n.zip **# the kit contains the asset files produced in notebook 4, and must be unzipped in the current working directory**
   $ az ml env setup --location westeurope -g jp-resource-group-PAID --name pdmmodelmanagement9 **# assign new model management name**
   $ az ml env show -g jp-resource-group-PAID -n pdmmodelmanagement9
   $ az ml env set -g jp-resource-group-PAID -n pdmmodelmanagement9
   $ az ml account modelmanagement show
   $ az ml service create realtime -f pdmscore.py -r spark-py -m pdmrfull.model -s service_schema.json --cpu 0.1 -n amlworkbenchpdmwebservice
   pdmrfull.model **# the files specified as arguments are from the asset kit o16n.zip**

After that, you can run the service as shown in Figure 5.

Note the following steps above differ from the documentation:

1. SKIP az ml experiment prepare -c docker
2. Use
   a. az ml env set -g jp-resource-group-PAID -n pdmmodelmanagement
   b. In lieu of
   c. az ml env set --resource-group <RESOURCE_GROUP> --cluster-name pdmmodelmanagement

Features influence study

One may ask what are the most important variables leading to a failure. For this purpose, Microsoft has provided the following parametric study:

Plot features importances

In [43]:
```
importances = model_pipeline.stages[2].featureImportances


rf = model_pipeline.stages[2]
df = pd.DataFrame.from_dict(importances.values).transpose()
df.columns = input_features

df.plot(kind='bar', figsize =(13,11),
        xlim = (0, max(importances.values)),
             title = "Model features importance",\
             fontsize = 24)
```

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8a84234080>

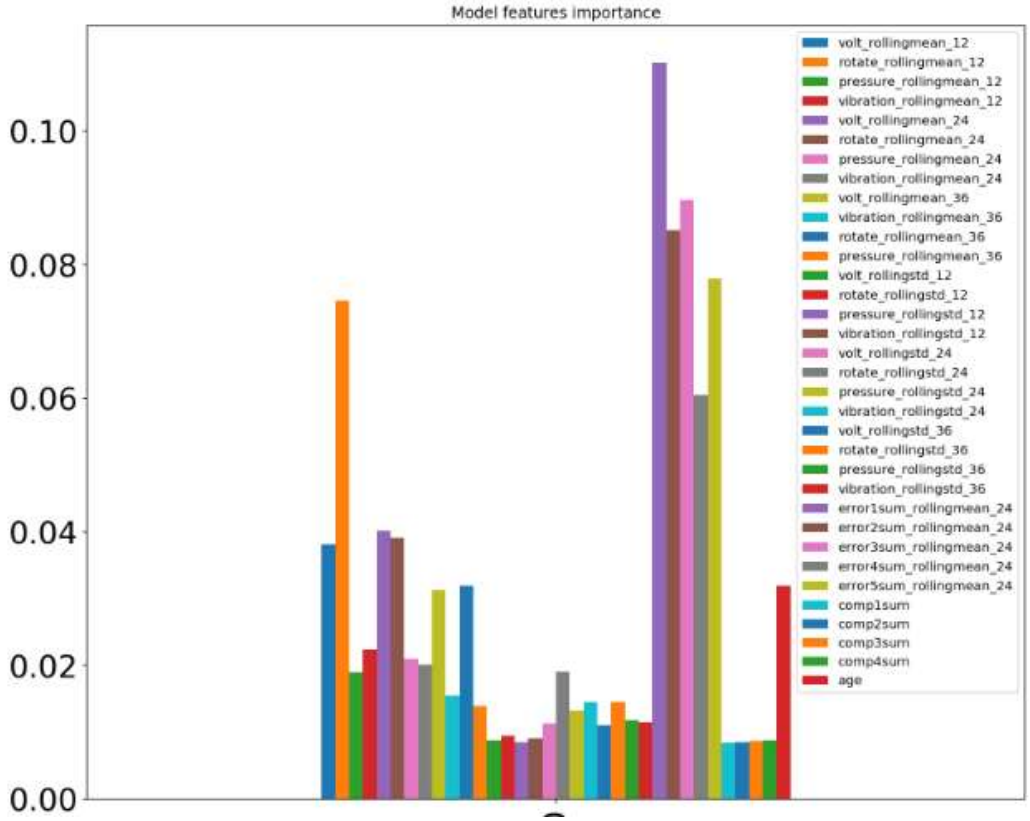


Figure 7: Impact of input variables on likelihood of failure (courtesy of Microsoft)

Team 213

Deep Learning in Computational Fluid Dynamics in the Microsoft Azure Cloud

Courtesy, Thurey Group, TUM

"Applying Deep Learning to highly complex compute intensive CFD by creating a database of synthetic flow descriptors that correlate with a given complex problem, so that the accurate solution could be inferred by a coarse-grid and much less compute intensive solution is one of the great benefits of this method."

MEET THE TEAM

End-User: Joseph Pareti, Artificial Intelligence Consultant
Software Provider: Nils Thurey and Mengyu Chu, Technical University of Munich, Germany
Resource Provider: Microsoft Azure Cloud
UberCloud Support: Wolfgang Gentzsch, President, The UberCloud
Microsoft Support: Yassine Khelifi, Support Escalation Engineer at Microsoft.

INTRODUCTION: USING ARTIFICIAL INTELLIGENCE TO SPEED UP A CFD APPLICATION

In this work, Convolution Neural Networks (CNN) are used for computing feature descriptors for density and velocity fields in smoke clouds. The CNN learns from a repository of computed results using the MANTAFLOW application. The CNN training, using TensorFlow, determines flow descriptors for density and velocities and a flow similarity score. In addition, the model includes a deformation limiting patch advection with anticipation module which enhances the stability and performance.

When given a coarse simulation, the model associates with a high degree of confidence a more accurate simulation that is retrieved from a database of computed results. When compared with a traditional approach, the CNN-based approach provides much faster time to solution with comparable accuracy as when using a sufficiently fine grid.

There is an interest among CAE users to accelerate the time-to-solution of numerically intensive applications, in order to run a sufficient number of cases for product optimization using a parametric approach. Hence it is expected that more applications will become available that use deep networks to either replace compute intensive modules, such as FluidNet, or replace the entire computation as described. Another reason to select this application for an UberCloud case study was the interest by the developer at TUM to contribute to our work in the form of Q&A, troubleshooting, etc.

Finally, we've made use of Microsoft Azure support for questions related to the configuration of the Data Science Virtual Machine (DSVM). If you are interested to run the application and you need more details than explained in this report, please send us a request at joepareti54@gmail.com or call us at +49 1520 1600 209.

USE CASE

Predicting smoke flows is a common task in computer graphics using CFD models. This can be done in 2D or 3D. Because an accurate calculation of flow properties is time consuming, researchers at the TUM developed new algorithms that benefit from deep learning technologies to dramatically reduce time-to-solution. While this paper is about a specific flow problem, with a specific solver (MANTAFLOW), it should be regarded as a feasibility study, and hence we encourage general purpose CFD users to look *beyond the box*, and come back to us with their specific CFD (or CAE) use case that could also benefit from deep learning or machine learning approaches.

At TUM, Nils Thurey and Mengyu Chu, in their work about Data-Driven Synthesis of Smoke Flows with CNN-based Feature Descriptors, take the following perspective to efficiently realize high-resolution flows (see Figure 1): they propose to use a fluid repository, consisting of a large collection of pre-computed space-time regions. From this, they synthesize new high-resolution volumes. In order to very efficiently find the best match from this repository, they propose to use novel, flow-aware feature descriptor. they ensure that L2 distances in this feature space will correspond to real matches of flow regions in terms of fluid density as well as fluid motion, so that they can very efficiently retrieve entries even for huge libraries of flow datasets.

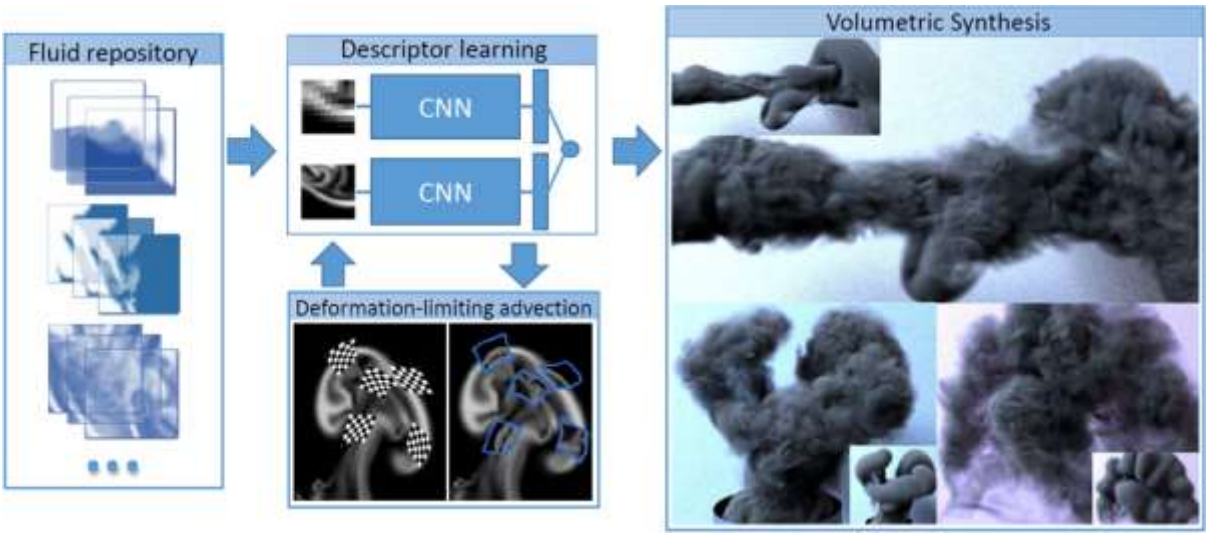


Figure 1: Realizing high-resolution flows efficiently by using a fluid repository, consisting of a large collection of pre-computed space-time regions.

SYSTEM ARCHITECTURE

In this application, we first ***built the MANTAFLOW application*** to calculate the exact data that will be used to train the deep network. There are 2D and 3D datasets. Next, we ***ran MANTAFLOW***. The output data of MANTAFLOW will be fed in ***TensorFlow*** to train the deep network. Once the network is trained we were able ***to make inferences*** and compare "exact" results (i.e. from MANTAFLOW with a fine grid) and results from the deep network. For the interested reader, all steps listed above are explained in detail in the Appendix. The paragraph on Results shows the exact and approximated output for the *smoke_tiled* example.

RESULTS

We achieved the following results:

1. An Azure-based demo version of the application that uses deep learning to significantly reduce time-to-solution in a CFD application.
2. A demo version that was employed for a specific example called the *smoke_tiled problem.*

In Figure 2 we compare the exact solution (right) with the approximate solution (left) for 1 sample:

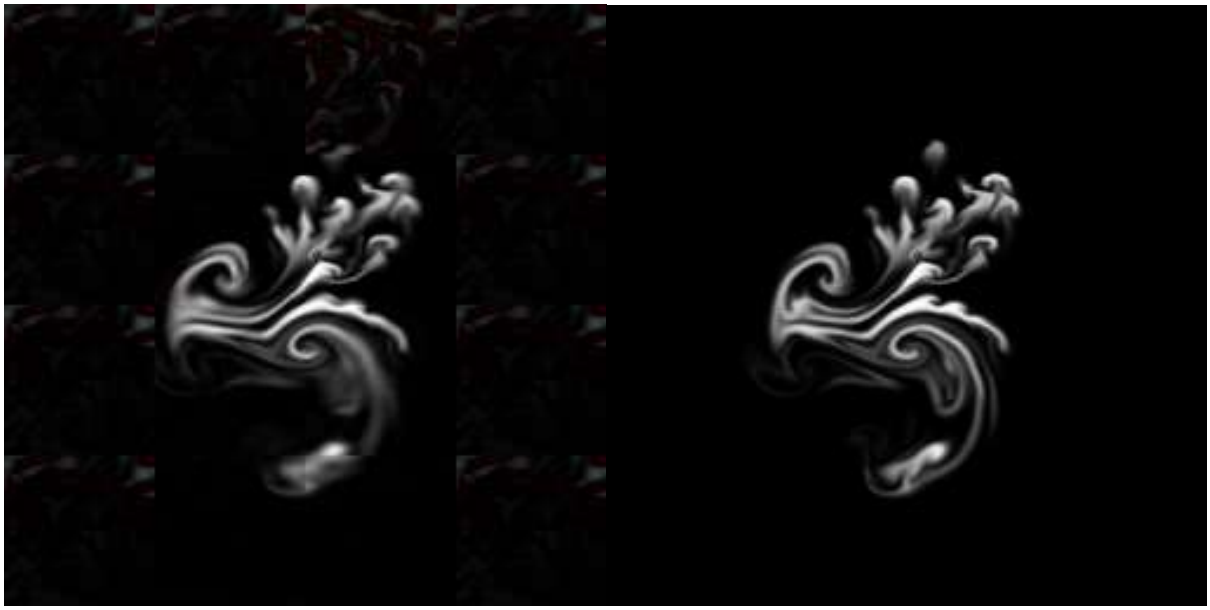


Figure 2: Smoke flow approximated by deep learning (left) vs. accurate MANTAFLOW calculation (right).

PERFORMANCE BENCHMARKING

This is out-of-scope for the current project. On our DSVM, the test cases presented in this report run in a few minutes; the exercise was just intended to validate the application and not to record any performance data.

BENEFITS

There are several R&D organizations working on deep learning tools to accelerate time-to-solution of CAE applications. In CFD, the deep learning approach varies:

1. One could aim at creating a database of synthetic flow descriptors that correlate with a given problem, so that the accurate solution could be inferred by a coarse grid solution (which is obviously less compute intensive). This is the approach taken by TUM in this report.
2. Another way to exploit the power of deep learning is by replacing parts of the computation by means of a deep network that simulates the exact behavior; this is the case of FluidNet where the non-linear Navier-Stokes partial differential equations are solved numerically, while the Poisson pressure correction term is simulated by a deep network instead of being calculated using traditional algorithms like sparse matrix solvers.
3. Or one could accurately calculate a number of cases, and then train the network using those results so that the network can predict the fluid flow based on geometry alone; this is the approach taken by Renumics in a recent UberCloud case study.[6]

If you are a CAE engineer, you may want to follow the methodology presented in this report and then possibly adapt it to your real problem, or work with us to design a deep learning-based solution.

CONCLUSION & RECOMMENDATIONS

The subject matter of this report is a demonstration how engineering simulations like computational fluid dynamics can benefit from Deep Learning with TensorFlow. Engineers are encouraged to

[6] Team 211: Deep Learning for Steady-State Fluid Flow Prediction in the Advania Data Centers Cloud

approach us, get inspired for their work, and test this demo version “as-is”, to evaluate how their own environment can benefit from this or a similar approach.

As a next step for interested readers, we suggest to work together in a team (including the customer, Joseph Pareti, UberCloud, Microsoft, and Nvidia) to identify an internal application, scope the project, and create a Statement of Work (SOW): this will specify what algorithm can be used that takes advantage of Deep Learning / Machine Learning. The next step will be a Proof of Concept prior to production ready applications.

Case Study Author – Joseph Pareti and Wolfgang Gentzsch

APPENDIX: Implementation notes: Azure resources

In this paragraph, we’ll review the details of the configuration in use on Azure, if you are relatively new to the field of Deep Learning and are interested in working with this or a similar project.

First, you need to subscribe with Azure, access the Azure dashboard, and then your Data Science Virtual Machine (DSVM) and Azure blob storage. DSVM is a family of Azure Virtual Machine images published by Microsoft on the Azure marketplace, specially built for Machine learning, deep learning and analytics. It contains a comprehensive set of popular tools used in data analytics, machine learning and AI development – all pre-installed, configured and tested so your data science environment is ready to go. Microsoft will bill you monthly on a per usage basis:[7]

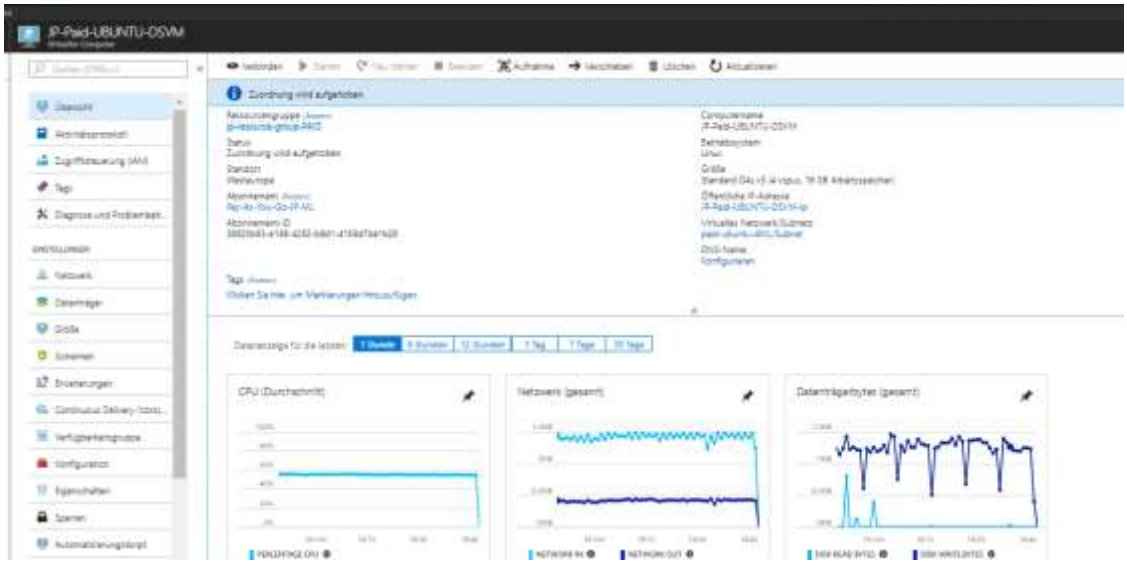


Figure 3: Overview of the DSVM in the Azure cloud

[7] To save money, it’s a good idea to shut down your DSVM when you don’t use it ☺ My monthly fees so far range from $60-$300 depending on consumption. More info at https://azure.microsoft.com/en-us/pricing/

The following diagram shows the resources we had available in the DSVM:

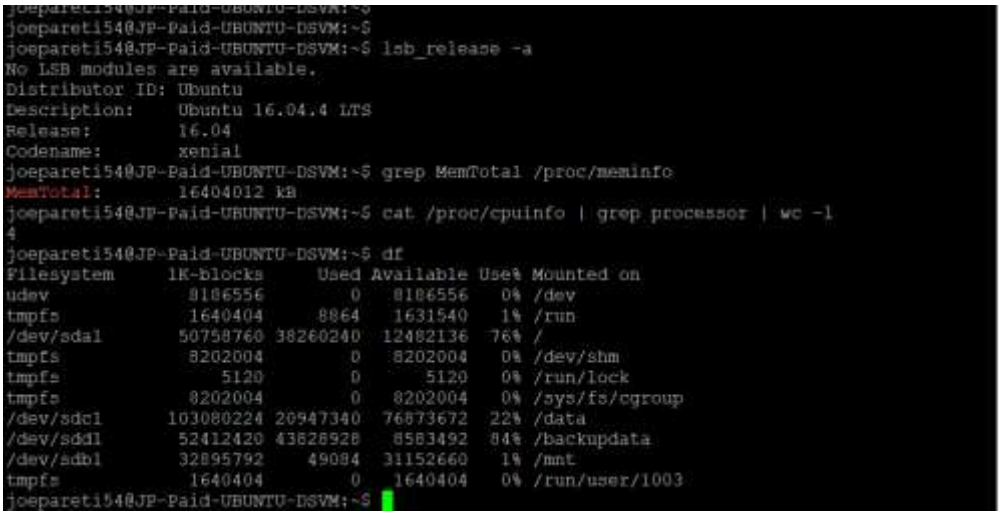


Figure 4: Putty terminal on the DSVM running Ubuntu

Building and running the MANTAFLOW application, training the network and calculating the smoke_tiled problem

In this paragraph we'll present the log file from the DSVM.

Setup environment

$ pwd
/home/joepareti54/TUM-validate

$ sudo apt-get install cmake g++ git python3-dev qt5-qmake qt5-default
Reading package lists... Done
Building dependency tree
Reading state information... Done
g++ is already the newest version (4:5.3.1-1ubuntu1).
python3-dev is already the newest version (3.5.1-3).
cmake is already the newest version (3.5.1-1ubuntu3).
git is already the newest version (1:2.7.4-0ubuntu1.4).
qt5-qmake is already the newest version (5.5.1+dfsg-16ubuntu7.5).
qt5-default is already the newest version (5.5.1+dfsg-16ubuntu7.5).
0 upgraded, 0 newly installed, 0 to remove and 110 not upgraded.
$ git clone https://bitbucket.org/mantaflow/manta.git
Cloning into 'manta'...
remote: Counting objects: 7350, done.
remote: Compressing objects: 100% (3131/3131), done.
remote: Total 7350 (delta 5113), reused 6275 (delta 4188)
Receiving objects: 100% (7350/7350), 2.92 MiB | 3.87 MiB/s, done.
Resolving deltas: 100% (5113/5113), done.
Checking connectivity... done.
joepareti54@JP-Paid-UBUNTU-DSVM:~/TUM-validate$

Build manta executable

$ pwd
/home/joepareti54/TUM-validate/manta/build
$cmake .. -DGUI=ON -DOPENMP=ON 2>&1 | tee -a cmake-July-27-2018.log
$make -j4 2>&1 | tee -a make-July-27-2018.log

Start virtual frame buffer because I cannot route graphic display from Azure

$ Xvfb :99 -screen 0 640x480x24 &
[1] 9098
$ export DISPLAY=:99
$

Build the training dataset for the *smoke tiled* problem:

$ pwd
/home/joepareti54/TUM-validate/manta/tensorflow/example1_smoke_tiled
$ ls
manta_genSimData.py runTests.sh tf_train_keras.py tilecreator.py
README.txt tf_genManySims.py tf_train.py

Set path to the manta build directory:

$ export PATH=/home/joepareti54/TUM-validate/manta/build/:$PATH

Follow the instruction in README.txt, build the training dataset:

$ manta manta_genSimData.py 2>&1 | tee -a manta-July27-2018.log

Version: mantaflow 0.12 64bit fp1 omp commit 15eaf4aa72da62e174df6c01f85ccd66fde20acc from Jul 27 2018, 09:28:45
Loading script 'manta_genSimData.py'
Noise tile loaded from file waveletNoiseTile.bin
Build info: mantaflow 0.12 64bit fp1 omp commit 15eaf4aa72da62e174df6c01f85ccd66fde20acc from Jul 27 2018, 09:28:45
Current time t: 0.0

ICP/mICP pre-conditioning only supported in 3D for now, disabling it.
Current time t: 0.5

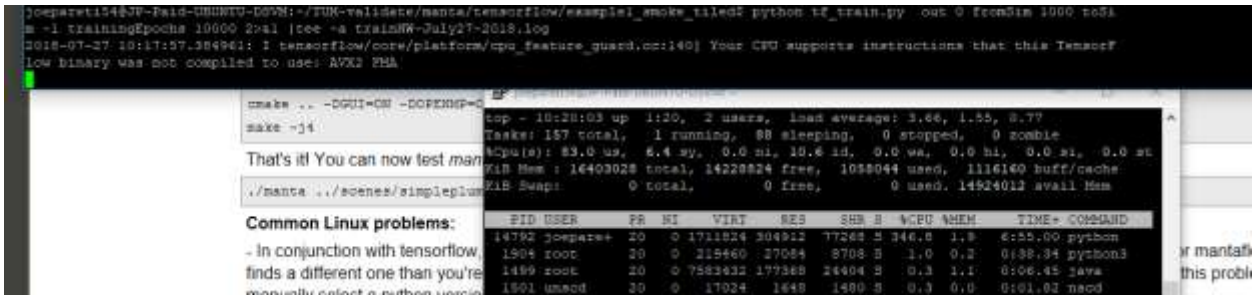
Current time t: 1.0
….
Writing grid density to uni file ../data/sim_1001/frame_0199/density_low_1001_0199.uni
Writing grid vel to uni file ../data/sim_1001/frame_0199/vel_low_1001_0199.uni
Writing grid xl_density to uni file ../data/sim_1001/frame_0199/density_high_1001_0199.uni
Script finished.
0 centre [+82.057129,+106.976028,+0.500000] radius 13.14271983426271 other [+0.992315,+1.022525,+1.000000]
1 centre [+159.852737,+121.765602,+0.500000] radius 17.06393042010118 other [+1.029323,+1.004425,+1.000000]
2 centre [+132.451920,+178.964081,+0.500000] radius 9.418921693160279 other [+1.033331,+1.021694,+1.000000]

3 centre [+112.929214,+81.441391,+0.500000] radius 17.09621443848919 other [+1.006732,+1.024593,+1.000000]
4 centre [+147.330032,+157.783569,+0.500000] radius 10.310567918335074 other [+1.048792,+0.965142,+1.000000]
5 centre [+78.095718,+95.134720,+0.500000] radius 11.093868114591041 other [+1.019041,+1.040684,+1.000000]
6 centre [+169.415039,+109.276825,+0.500000] radius 9.889896197742017 other [+0.992716,+0.957080,+1.000000]
7 centre [+153.345779,+77.587715,+0.500000] radius 15.978795998555917 other [+0.999475,+0.968100,+1.000000]
8 centre [+82.063469,+131.189926,+0.500000] radius 12.422082145402 other [+0.978788,+0.955976,+1.000000]
9 centre [+151.872528,+122.567413,+0.500000] radius 14.995933928095454 other [+1.019633,+0.989048,+1.000000]
10 centre [+95.823654,+117.936523,+0.500000] radius 9.356802719393698 other [+1.000962,+1.048095,+1.000000]
11 centre [+113.804543,+142.279907,+0.500000] radius 13.028546502477788 other [+0.993923,+1.049784,+1.000000]
Using sim dir '../data/sim_1001/'

Finally, train the network using TensorFlow:

There is a typo in the README.txt file: the script to be used is tf_train.py

$ python tf_train.py out 0 fromSim 1000 toSim -1 trainingEpochs 10000 2>&1 |tee -a trainNW-July27-2018.log



…
Epoch 10000/10000 - Avg. cost= 254.125065002 - Avg. validation cost= 1135.869055176
200 epochs took 6.60 seconds.

*****TRAINING FINISHED*****

Training needed 5.51 minutes.

To apply the trained model, set "outputOnly" to True, add "out 1 loadModelTest 1" to script call
Use the trained Network to generate the output

$ python tf_train.py out 1 fromSim 1000 toSim -1 loadModelTest 1 2>&1 |tee -a GenOutput-July27-2018.log

2018-07-27 10:30:07.319304: I tensorflow/core/platform/cpu_feature_guard.cc:140] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
/anaconda/envs/py35/lib/python3.5/site-packages/h5py/__init__.py:36: FutureWarning: Conversion of the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.
from ._conv import register_converters as _register_converters

Team 214
Demonstrating a Machine Learning Model for Predictive Maintenance on Microsoft Azure Part II *

Picture, Courtesy: VIZIYA Corporation

"This Machine Learning study (update from Team 212) is for predictive maintenance demonstration and learning purposes, and may be used as a baseline for the reader's own custom application."

MEET THE TEAM

End-User: Joseph Pareti, Artificial Intelligence Consultant
Software Provider: Open source Predictive Maintenance template by the Microsoft Azure Team
Resource Provider: Microsoft Azure Cloud
UberCloud Support: Wolfgang Gentzsch, President, The UberCloud
Microsoft Support: Yassine Khelifi, Support Escalation Engineer at Microsoft.

USE CASE

The Predictive Maintenance model described in this report is open source and can be applied to different equipment types for which telemetry data and maintenance data records are available. **The implementation described herein assumes an Azure cloud subscription and some operating knowledge on Azure.**

Four machine types are considered, each machine has four components, and there is *telemetry* data available on voltage, vibration, speed, and pressure, as well as maintenance records (indicating when last a component was replaced on what machine), error logs (not necessarily implying failure), machine characteristics, and how long each machine has been in service. The model is built in four stages each of which is implemented in a Jupyter notebook running Python version3:

1. Data ingestion
2. Feature engineering
3. ML model
4. Operationalization

*) Note: This document is the continuation of our case study 212, about a recent release of the predictive maintenance software by Microsoft. The functionality of the model, and predictive precision and accuracy are the same as in the first release, however the current implementation is based on Azure Machine Learning Services and databricks that significantly reduces time-to-solution, while simplifying the end-user's administration task, since the entire application runs in Azure. Therefore, the desk-top front end, "Azure Machine Learning Workbench" which was a required component in the first release, has been removed.

Notebook 1: Data ingestion is about accessing the datasets from blob storage, cleaning the data, and storing the data as a SPARK dataframe in cluster for further processing by the next notebooks.

Notebook 2: Feature engineering loads the data sets created in the **Data Ingestion** notebook and combines them to create a single data set of features (variables) that can be used to infer a machine health condition over time.
The goal is to generate a single record for each time unit within each asset. The record includes features and labels to be fed into the machine learning algorithm.

Predictive maintenance takes historical data, marked with a timestamp, to predict current health of a component and the probability of failure within some future window of time. These problems can be characterized as a *classification method* involving *time series* data. Time series, since we want to use historical observations to predict what will happen in the future. Classification, because we classify the future as having a probability of failure.

Note that this step requires significant data science and programming skills, however it is separate from classical Machine Learning tasks. It also requires domain expertise in order to focus on relevant features for the task at hand.

Notebook 3: The ML model uses the labeled feature data set constructed in notebook 2, it loads the data and splits it into a training and test data set. We then build a machine learning model (a decision tree classifier or a random forest classifier) to predict when different components within our machine population will fail.

Two different classification model approaches are available in this notebook:

- **Decision Tree Classifier**: Decision trees and their ensembles are popular methods for the machine learning tasks of classification and regression. Decision trees are widely used since they are easy to interpret, handle categorical features, extend to the multiclass classification setting, do not require feature scaling, and are able to capture non-linearities and feature interactions.
- **Random Forest Classifier**: A random forest is an ensemble of decision trees. Random forests combine many decision trees in order to reduce the risk of overfitting. Tree ensemble algorithms such as random forests and boosting are among the top performers for classification and regression tasks.

Notebook 4: Operationalization, loading the model from the Code/3_model_building.ipynb Jupyter notebook and the labeled feature data set constructed in the Code/2_feature_engineering.ipynb notebook in order to build the model deployment artifacts. The notebook is used to deploy and operationalize the model and is built on the Azure Machine Learning service SDK.

An appendix at the end of this report provides additional detail on the 4 notebooks. **If you whish a complete input/output set for this use case, e.g. to compare with your own work, please send us e-mail to joepareti54@gmail.com.**

AZURE MACHINE LEARNING SDK ON AZURE DATABRICKS

The ML SDK and databricks are options to implement custom AI which require custom data and model training. The Azure ML SDK defines a workspace that contains compute and storage resources, as well as models, experiments (i.e. all attempts with different parameters), and deployment services such as Docker images.

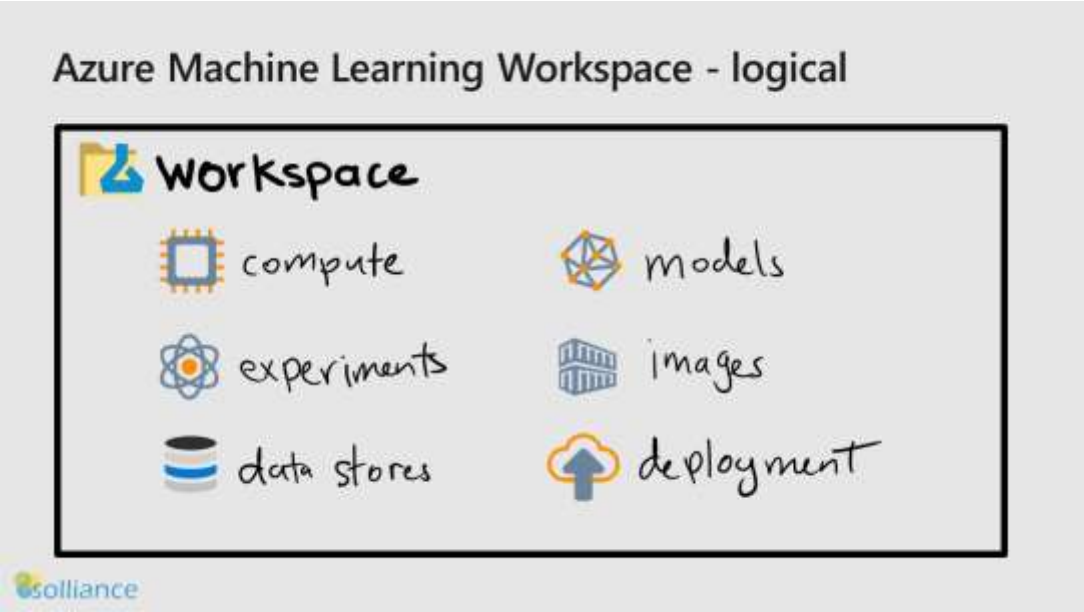


Figure 1: Azure Machine Learning Workspace.

Databricks is an architecture for Spark environments that provides out-of-the-box support for common interfaces and supports at-scale Machine Learning deployments:

- Azure Databricks is a fully-managed, cloud-based Big Data and Machine Learning platform.
- It empowers developers to accelerate AI and innovation by simplifying the process of building enterprise-grade production data applications
- It is built on a joint effort by the team that started Apache Spark and Microsoft
- It is a single platform for big data processing and Machine Learning.

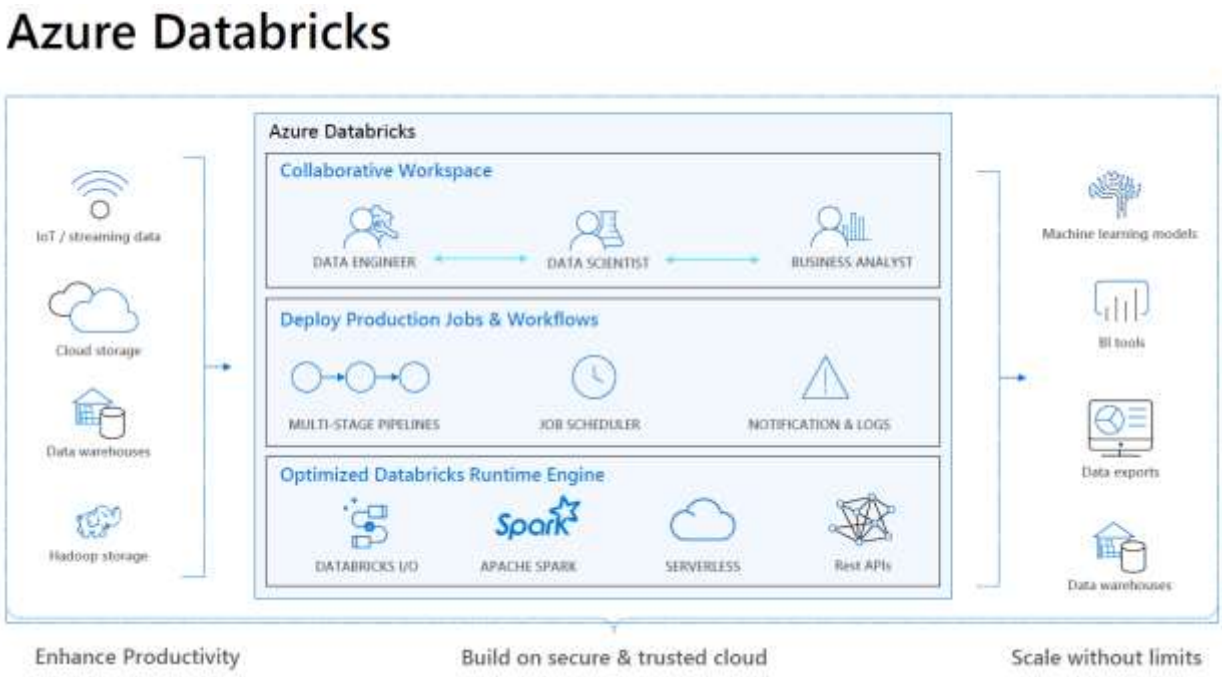


Figure 2: Azure Databricks.

SYSTEM ARCHITECTURE USED FOR THIS CASE STUDY

The following is a list of the software components that must be implemented:

- A workspace in Azure
- A development environment for ML
- An Azure Databricks cluster deployed with the following configuration:
    - Databricks Runtime version: (latest stable release) (Scala 2.11)
    - Python version: 3
    - Driver/Worker type: Standard_DS13_v2
    - Python libraries installed:

      ipython==2.2.0, pyOpenSSL==16.0.0, psutil, azureml-sdk[databricks], cryptography==1.5

In the implementation for this case study, the Databricks cluster includes 2-8 nodes that are DS3_v2 virtual machines, which is a low-cost option to demonstrate the feasibility of the application. screen dumps below provide additional detail on utilized software components.

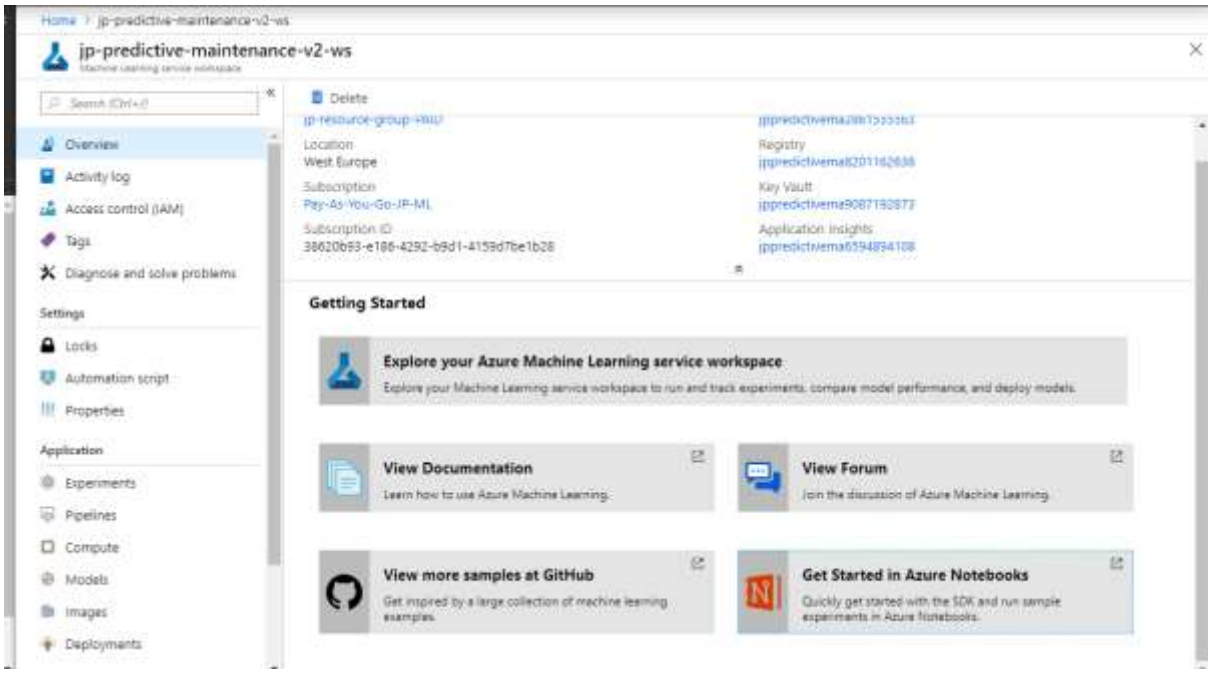


Figure 3: Azure Machine Learning Workspace; use the "view forum" option (bottom right) to ask Microsoft product managers for advice.

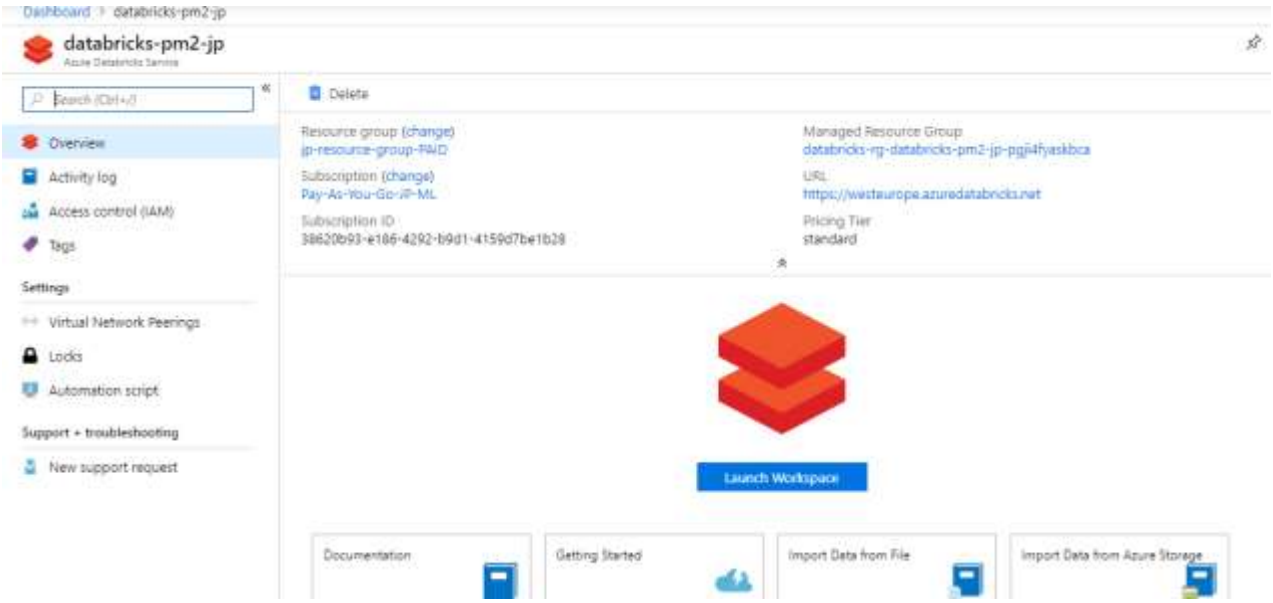


Figure 4: Databricks Workspace.

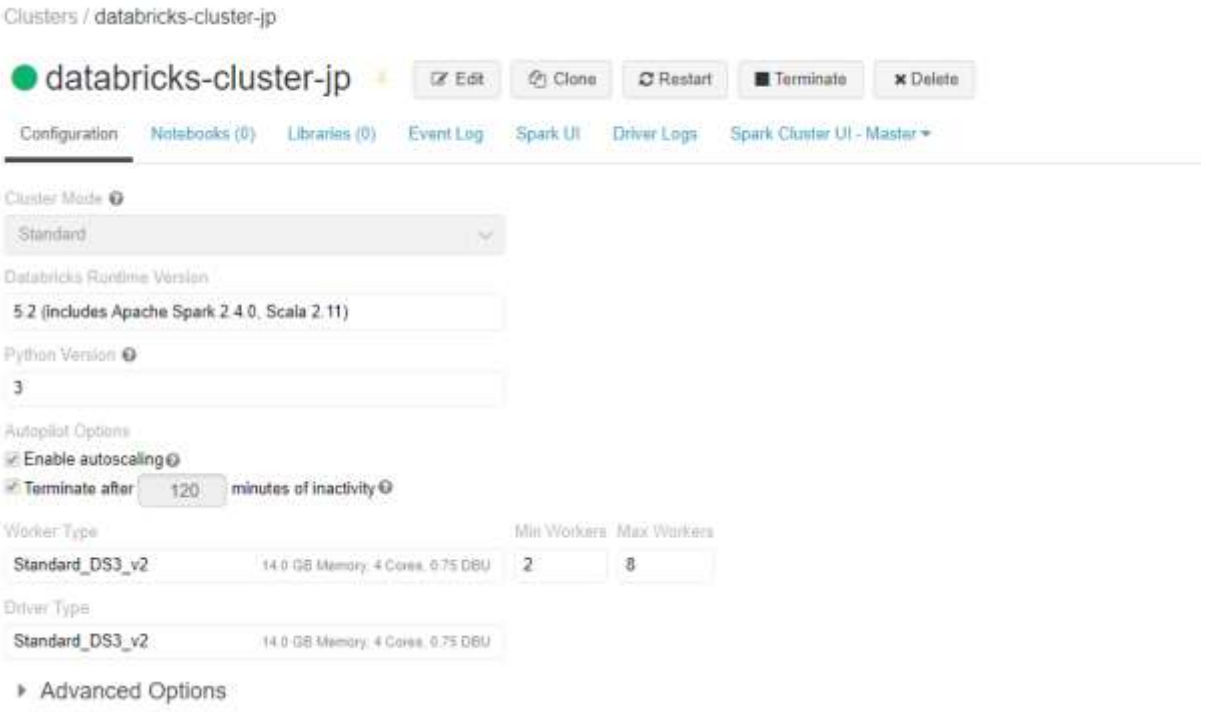


Figure 5: Databricks cluster; set the terminate flag to avoid being charged after the job is done; enable autoscaling to allow databricks to grow or shrink according to job requirement.

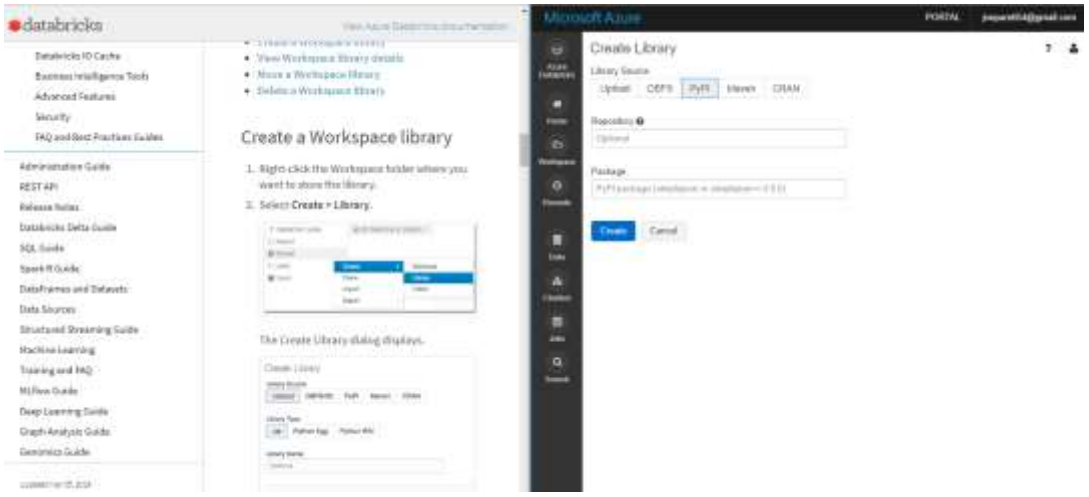


Figure 6: Libraries on databricks cluster. Left: excerpt from the User's guide, Right: configuration used for this case study.

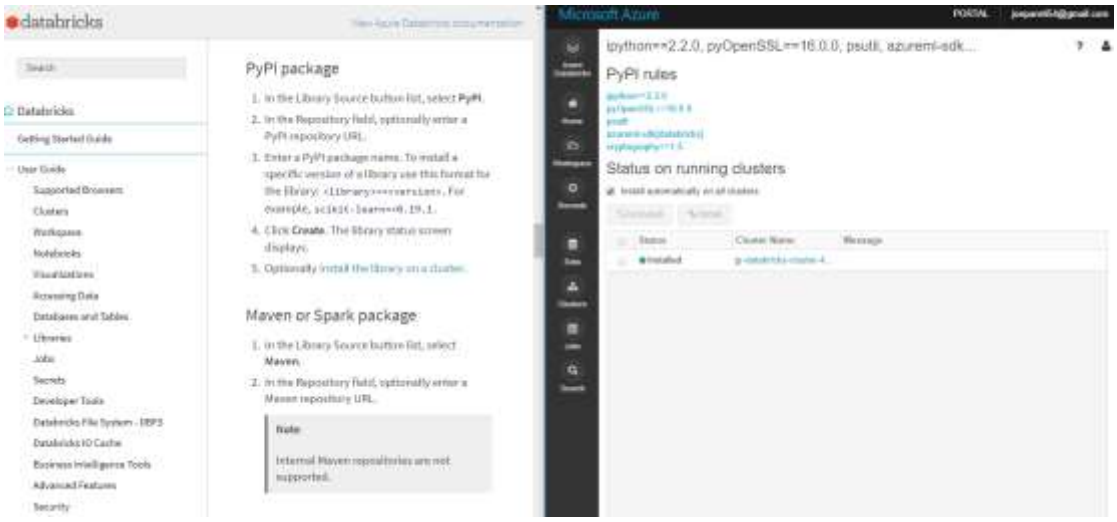


Figure 7: PyPI package on databricks cluster. Left: User's guide, Right: configuration used for this case study.

RESULTS

The result of this study is an enhanced demo version compared to the one reported in case study 212. If you have an Azure subscription, you can build the demo by yourself starting here. If you need a step-by-step tutorial on how to deploy Azure ML SDK and databricks, you can use this youtube video.

PERFORMANCE BENCHMARKING

This demo was run on a databricks cluster with 2 to 8 nodes that are Standard_DS13_v2: the table below provides performance figures for the current and previous release, which was tested on a single DSVM with 4 vCPUs and 16 GB RAM. *Use these figures to get a feel for improvement and not as absolute values.*

Release	Notebook 2 Time-to-solution	Notebook 3 Time-to-solution
previous	71 minutes	10 minutes
current	27 minutes	21 minutes

CONCLUSION & RECOMMENDATIONS

As recommended by Microsoft, we have run the predictive maintenance application using the new software architecture including Azure Machine Learning SDK and Azure Databricks.

The data, features set, labels and model are the same as in the previous release, therefore the performance is also approximately the same.

The significant advantages of the current release compared to the previous one, are:

- *Shorter time to solution*, reported in the paragraph Performance Bench-marking.
- *Easier administration,* because the entire application runs in Azure, i.e. the client-side workbench component has been removed.

Predictive Maintenance is one of the most popular ML applications, and one that is widely considered prime time for productive deployment: custom solutions exist, as well as off-the-shelf packages, and software building blocks that may be sold with consulting services. We have recently met with several companies across the world that either implement or propose predictive maintenance for production. These companies are SEW, Duerr, Compacer, Siemens, SAS, and more.

One benefit of this case study is that it is built on open source software, and hence it can be adapted to support a customer's proof of concept for predictive maintenance of machines, industrial or end-user equipment, plants, etc.

While the available results are promising in a demo environment, a significant effort to customize the model for real use cases would be needed, especially to build features that effectively classify failures. This effort requires both data science skills and domain expertise.

Potentially, Microsoft recently announced Automated Machine Learning could help to optimize the model precision by iteratively determining the best hyperparameters in a very effective, automated way in order to accelerate time-to-market for the application.

If you are interested to implement predictive maintenance in your environment using ML, we can offer a discovery workshop and Scope of Work service in collaboration with industry partners Microsoft and Nvidia if appropriate. Please approach joepareti54@gmail.com.

APPENDIX

If you are a developer, you may find this information useful: in this paragraph, we are providing a step-by-step qualitative description of the model. Note that the code is not given here, but if you want to build the demo by yourself, and want to compare your results with a complete reference data set, please send e-mail at joepareti54@gmail.com. All (Jupiter) notebooks in this demo are written in Python 3.

Notebook 1 – Data ingestion

- Import needed libraries such as pandas, seaborn, matplotlib
- Define comma separated value (csv) files that contain:
  - Machine data

  - Maintenance history
  - Error logs
  - Telemetry data: voltage, rotation speed, pressure, vibration
  - Failure events
- Convert data into spark format
- Plot some variables for verification
- Write data to Azure blob storage for processing by next notebooks

Notebook 2 – Features engineering

Note that the features are in the columns, used for classifying failures.

- Set up the environment by importing libraries and for plotting
- Set timers for performance measurements (note that you have to use "Run all", which we did not use in our own testing, hence the figures in the paragraph "performance benchmarking" are not accurate: the real performance ought to be better than reported)
- Read in the data
- Calculate means and standard deviations
- Choose the timestamps to align the data
- Define rolling windows to build lag features (12h, 24h, 36h)
- Calculate lag values
- Create a column for each error id
- Join the telemetry data with error data
- Create a column for each component replacement
- Align component information on telemetry features timestamps
- For component 1 records, get on each machine the last replacement date for each timepoint, and calculate the number of days between replacements: this is expected to be a good failure predicting feature because the longer a component is in service, the higher the probability of failure
- Same operations as above for components 2,3,4
- Join component 3 and 4
- Join component 2, 3, and 4
- Join component 1 to 2, 3, 4
- Choose the timestamp to align the data
- Collect the data
- One hot encoding of the variable model
- Join error features with component maintenance features
- Join with machine features (static data represented by Boolean variables)
- Clean up unnecessary columns
- Join telemetry with error, maintenance, machine features to create a final feature matrix
- Map the failure data to final feature matrix
- Get the frequency of each component failure
- Lag values to manually backfill: this is because we need to predict impending failures so that human intervention in a business-dependent time window is possible. For this exercise, a 7 days window is selected

- Create label feature
- Restrict that label to be in the range 0-4 and remove extra columns
- Write labeled feature data to (Azure) storage

Notebook 3 – Model building

- Import libraries, azureml components and components from pyspark.ml for creating pipelines and the model
- Set timers as in Notebook 2
- Enter workspace details (subscription id, etc.)
- Persist workspace information in a json file which will be needed in Notebook 4
- Define list of input columns for modeling; use the known label and key variables
- Assemble features
- Set maxCategories so features with > 10 distinct values are treated as continuous
- Fit on whole dataset to include all labels in index
- Split the data into training and test data based on date
- Train the model: there are 2 options, i.e. decision tree and random forest – My experiment uses random forest that is an ensemble of decision trees and has less risk of overfitting
- Chain indexer and model in a pipeline
- Train the model, this also runs the indexer
- Make predictions: the pipeline does the same operations on the test data
- Create the confusion matrix for the multiclass predictions: the correct predictions are in the main diagonal, values above or below are false alarms, i.e. if above, the model predicted failures that did not occur
- Compute test error based on prediction and true label
- False positives
- False negatives
- Accuracy calculation
- Calculate precision and recall
- Save the model

Once the model is trained, we follow up with model registration in Azure ML. i.e model.register call. See next paragraph for details.

Notebook 4 – Operationalization

- Create docker image with scoring script
- Register image in Azure container registry
- Deploy image to Azure container instance for prediction serving. For production deployment, Azure ML support Azure Kubernetes cluster for serving predictions at scale.

Thank you for your interest in our free and voluntary UberCloud Experiment

If you, as an end-user, would like to participate in this Experiment to explore hands-on the end-to-end process of on-demand Technical Computing as a Service, in the Cloud, for your business then please register at: http://www.theubercloud.com/hpc-experiment/

If you, as a service provider, are interested in promoting your services on the UberCloud Marketplace then please send us a message at https://www.theubercloud.com/help/

Annual UberCloud Compendiums:

1st Compendium of case studies, 2013: https://www.theubercloud.com/ubercloud-compendium-2013/
2nd Compendium of case studies 2014: https://www.theubercloud.com/ubercloud-compendium-2014/
3rd Compendium of case studies 2015: https://www.theubercloud.com/ubercloud-compendium-2015/
4th Compendium of case studies 2016: https://www.theubercloud.com/ubercloud-compendium-2016/
5th Compendium of case studies 2018: https://www.theubercloud.com/ubercloud-compendium-2018/

Awards for UberCloud's Technology and Community Contributions:

HPCwire Readers Choice Award 2013: http://www.hpcwire.com/off-the-wire/ubercloud-receives-top-honors-2013-hpcwire-readers-choice-awards/
HPCwire Readers Choice Award 2014: https://www.theubercloud.com/ubercloud-receives-top-honors-2014-hpcwire-readers-choice-award/
Gartner Names The UberCloud a 2015 Cool Vendor in Oil & Gas: https://www.hpcwire.com/off-the-wire/gartner-names-ubercloud-a-cool-vendor-in-oil-gas/
HPCwire Editors Choice Award 2017: https://www.hpcwire.com/2017-hpcwire-awards-readers-editors-choice/
IDC/Hyperion Innovation Excellence Award 2017: https://www.hpcwire.com/off-the-wire/hyperion-research-announces-hpc-innovation-excellence-award-winners-2/
HPCwire Editors Choice Award 2018: https://www.theubercloud.com/ubercloud-receives-top-honors-2018-hpcwire-readers-choice-award/
IDC/Hyperion Innovation Excellence Award 2018: https://insidehpc.com/2018/11/ubercloud-wins-hyperion-hpc-innovation-excellence-award-neuromodulation-project/

If you wish to be informed about the latest developments in technical computing in the cloud, then please **register at http://www.theubercloud.com/** and you will get our free monthly newsletter.