



XGBoost is All You Need

Bojan Tunguz, Data Scientist | GTC Spring 2024/March 18, 2024

XGBoost is All You Need

Doing Data Science and Machine Learning with Gradient Boosted Trees

- Intro and my own Background
- What do I really mean with my (in)famous tagline
- Gentle introduction to Data Science and Machine Learning for tabular data
- What are gradient boosted trees?
- History of XGBoost and its properties
- Accelerated computing, scalable machine learning, data centers, and all of that
- Nontrivial use case 1: multi-GPU and multi-machine training
- Nontrivial use case 2: use of Shapely values for feature selection and feature engineering
- Nontrivial use case 3: use of XGBoost for unsupervised tasks
- Miscellaneous topics and Q&A

Intro and my own Background

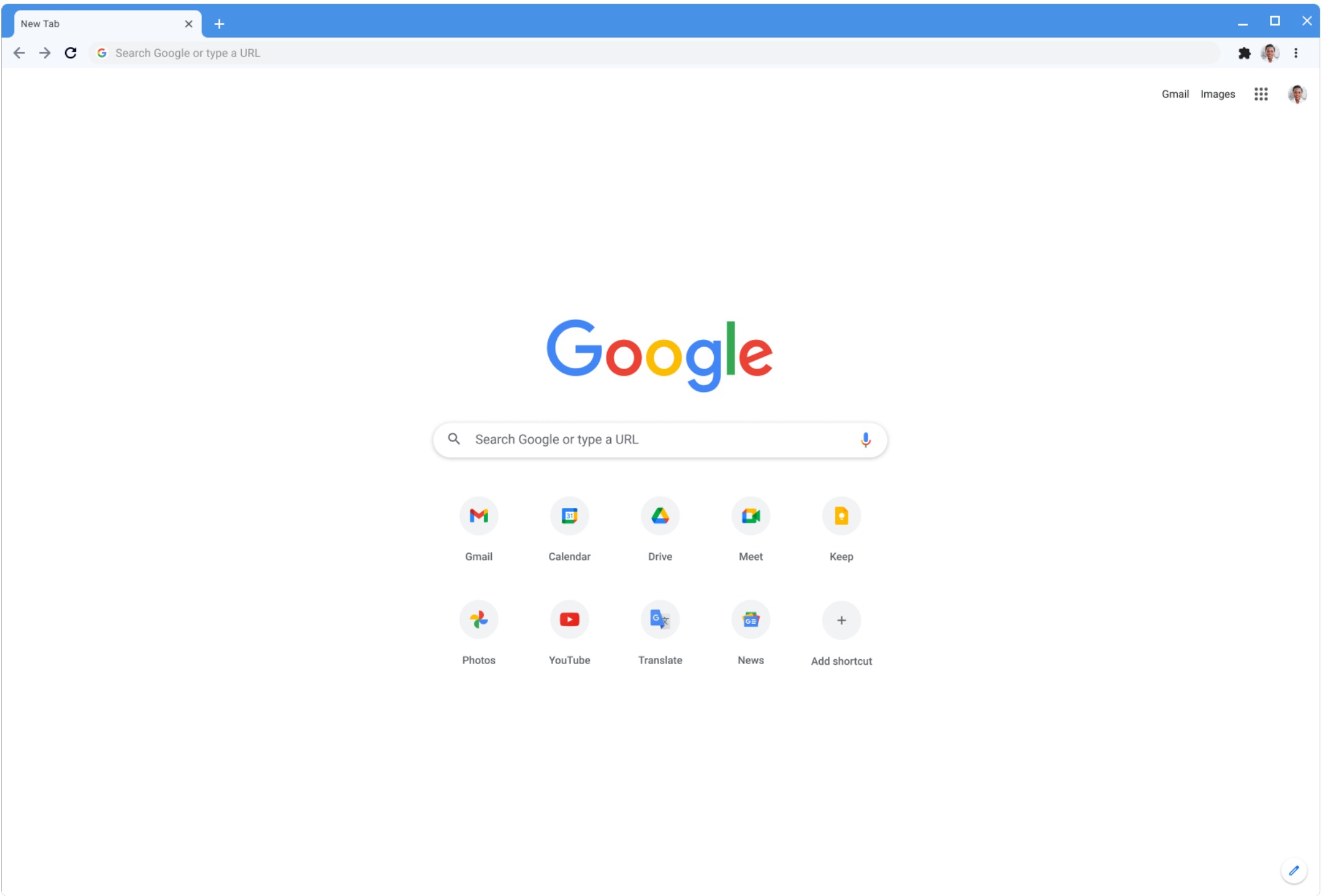
Just enough about me to understand where I am coming from

- Theoretical Physicist by training – degrees from University of Illinois and Stanford
- Discovered Data Science and Machine Learning accidentally online – thanks to MOOCs and similar resources
- Stumbled upon Kaggle, became really into it, and used it to learn more about DS, ML, AI, etc.
- Kaggle – excellent online credentialling
- 100% of everything I know about DS, ML, and AI is either something I've learned on my own or thought work on practical problems
- Worked at several tech startups, and now at Nvidia
- Ever since getting into Data Science my main concern has been with **practical** considerations regarding ML and DS tools and methods

What do I really mean with my (in)famous tagline

No, it is not as crazy as it sounds

- “XGBoost is all you need” - It started as a semi-joke, got a life of its own
- No, I do **not** mean that you should try to tackle every ML problem with XGBoost
- Gradient Boosted Trees (GBTs) are generally very well suited for tabular data ML
- I regularly use all available GBT libraries – HistGradientBoosting, XGBoost, LightGBM, CatBoost, etc.
- All have their strengths and weaknesses; not a single one universally better than the others.
- Maturity and robustness of the libraries
- Practical/usability considerations - ease of install and maintenance
- Supported hardware
- GPUs!
- Scalability
- NVIDIA main maintainer of XGBoost – shoutout to Rory Mitchell, Hyunsu Cho, Jiaming Yuan, and all other NVIDIAans working diligently on this project



What I don't mean by my tagline

- XGBoost is universally superior to any other ML algorithm
- XGBoost is universally superior to all other GBM algorithms/libraries
- You should **never** use neural networks
- You don't need any feature engineering
- You should never use more advanced ML techniques (ensembling, etc.)
- You should build AI with XGBoost

Intro to Tabular Data and Machine Learning

What is tabular data?

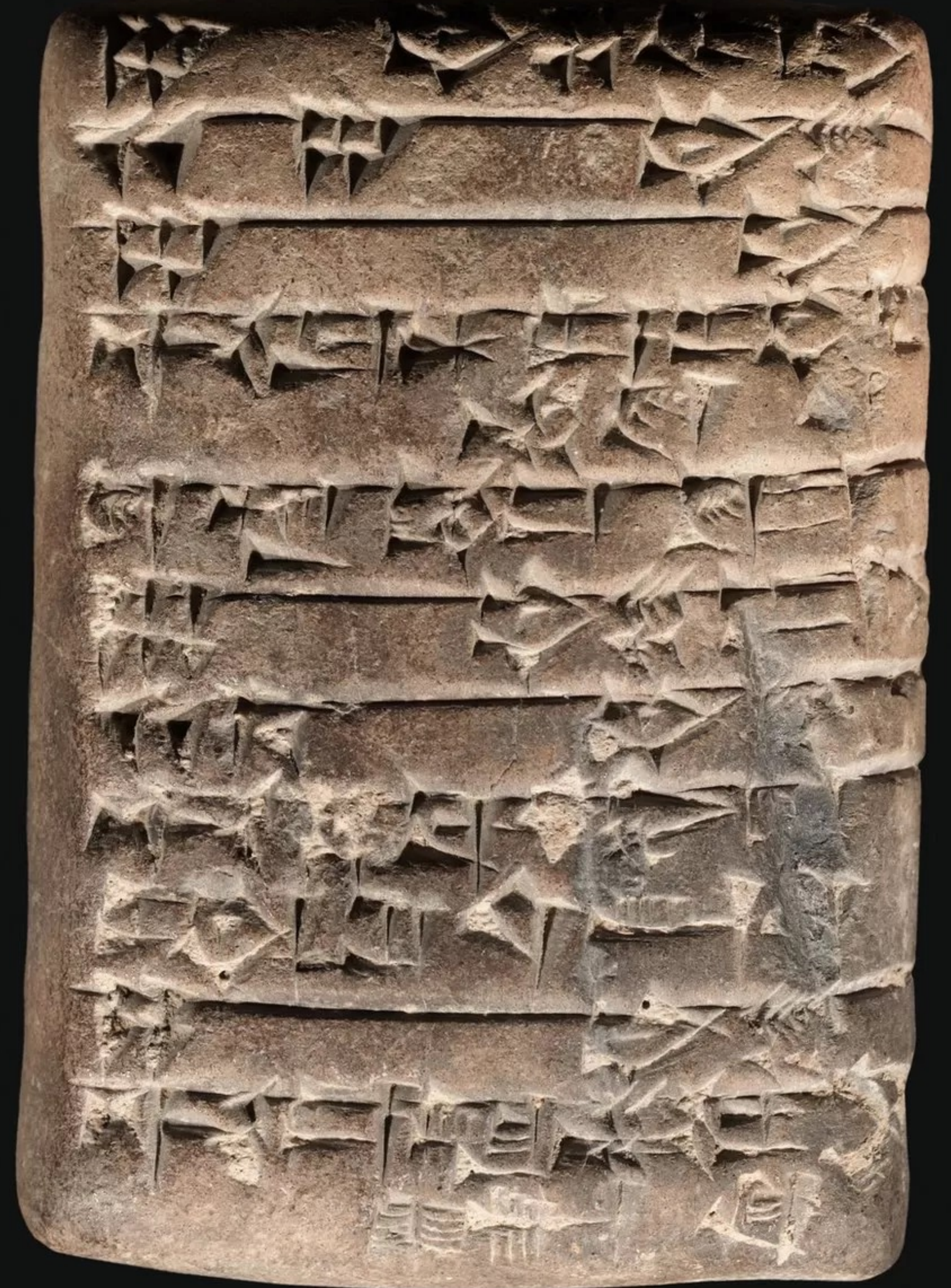
el	trim	body	transmission	vin	state	condition	odometer	color	interior	seller
nto	LX	SUV	automatic	5xyktca69fg566472	ca	5.0	16639.0	white	black	kia mot americ inc
nto	LX	SUV	automatic	5xyktca69fg561319	ca	5.0	9393.0	white	beige	kia mot americ inc
s	328i SULEV	Sedan	automatic	wba3c1c51ek116351	ca	4.5	1331.0	gray	black	financi service remark (lease)
	T5	Sedan	automatic	yv1612tb4f1310987	ca	4.1	14282.0	white	black	volvo n rep/wo omni
s pe	650i	Sedan	automatic	wba6b2c57ed129731	ca	4.3	2641.0	gray	black	financi service remark (lease)

MMWR_week	Recip_County	Recip_State	Series_Complete_Pop_Pct	Series_Complete_Yr
46	Lawrence County	AR	42.3	6938
46	Escambia County	AL	33.1	12112
46	Tama County	IA	58.0	9777
46	Mariposa County	CA	0.0	0
46	Jackson County	IA	49.2	9568

How the world's first accountants counted on cuneiform

- Tabular data as transactional data has existed since the dawn of writing
- May even preceded “language” writing

(BBC story: <https://www.bbc.com/news/business-39870485>)



Why do we care about tabular data?

- Most commonly used type of data
- No definitive measure, but estimates range from little over 50% to well over 90% of practicing Data Scientists use tabular data as their primary type of data in their professional setting
- Growth of relational databases
- There is money in tabular data – sometimes quite literally
- Intellectually challenging – most heterogenous and diverse type of data. Stymies many out-of-box ML approaches

What is tabular data?

- One common misconception/concern: tabular data refers to the format, not anything particular about the data itself
- Tabular data qualitatively different from text, sound, image and video data
- “Non-tabular” data: main feature is existence of some intrinsic local structure
- Tabular data: generally, no local structure
- In general, you **can** shuffle tabular data columns any way you like
- Is time series data tabular data? Depends – time-series has a local structure (time ordering), but in practical terms can often be transformed into “pure” tabular data (various aggregation feature engineering, etc.)

Practical considerations with tabular data 1

- Tabular data is very often **transactional** data – data that is derived from various business-process transactions
- This makes tabular data acquisition **very** expensive – you essentially need to run an actual business in order to acquire meaningful tabular datasets
- The bigger the business, the bigger the dataset(s) that one acquires
- Most of the biggest and most informative tabular datasets proprietary – crown jewels of many businesses
- As a practical matter, tabular data defines Data Science as a discipline
- However, getting good at tabular data ML modeling can be challenging
- Most educational institutions (and even researchers) have poor access to meaningful tabular datasets

Practical considerations with tabular data 2

- Getting good at tabular data modeling artisanal – requires access to bespoke datasets, processes, and mentors
- “Classic” Kaggle – wide variety of accessible and meaningful tabular data datasets and problems
- Many industries in which tabular problems are present (healthcare, finance, insurance, etc.) are highly regulated and hence very conservative in terms of the advancements of the field
- Since data and the business are highly intertwined, getting more and higher quality data more valuable than developing better models
- All these considerations make tabular data a very complex topic, spanning several different domains
- In this talk we will focus just on one part of the tabular data “problem” – Machine Learning modeling for tabular datasets

Tabular data ML has resisted the DL revolution

- Using Neural Networks for tabular data modeling remains an “unsolved” issue
- The best out-of-the box algorithms for tabular data remain gradient boosted decision trees (“XGBoost is all you need.”)
- Several recent attempts to come up with robust NN-based algorithms/libraries for tabular data
- Nonetheless, most of them don’t work very well outside of a few handpicked cases
- Most of them require long training times
- By their very nature, Neural Networks contain “locality inductive bias”

Gradient Boosted Trees

- Out of the box trees are **much** better suited for the irregular structure of tabular data
- Trees work with minimal preprocessing of the data
- Most gradient boosted tree libraries can handle missing values with no preprocessing
- Not very sensitive to outliers
- Don't require fiddling with the underlying algorithm architecture
- Can be very effective with identifying feature importance(s)
- Very fast to train, especially on GPUs
- Libraries easy to install and maintain

Comparing decision boundaries

Image taken from [arXiv:2207.08815v1](https://arxiv.org/abs/2207.08815v1)

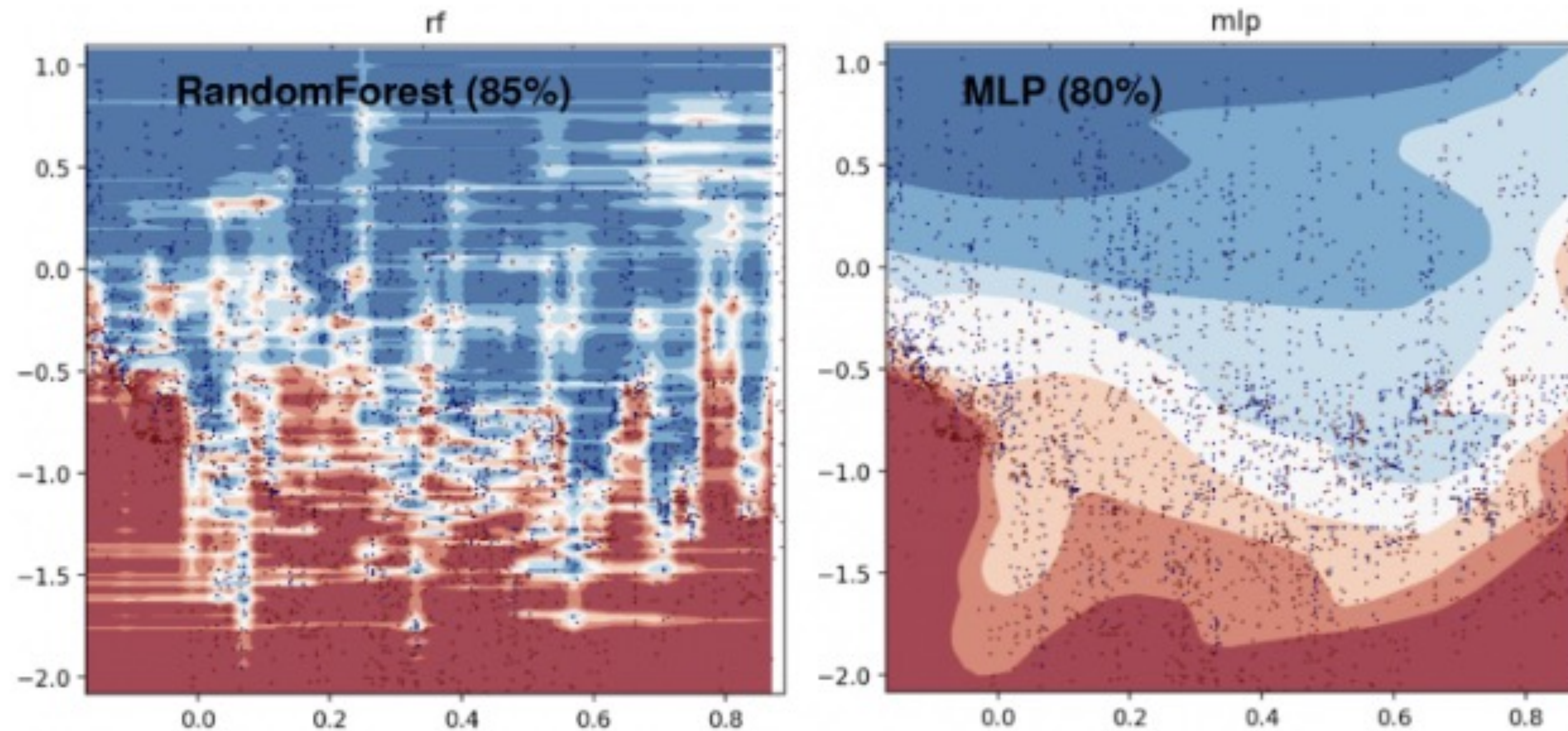


Figure 20: Decision boundaries of a default MLP and RandomForest for the 2 most important features of the *electricity* dataset

GBT shortcomings

- Not sensitive to **linearities** in data
- Not good at predicting new values that are well outside the range of the training set
- Boosting is an inherently “shallow” algorithm – can handle only one “layer” at the time
- No implicit feature engineering/feature embedding/feature selection mechanism
- Trained model files can be extremely large
- Putting them in production can still be a challenge

Need more GBT research

- The basic GBT algorithms have not changed much in years
- Most GBT libraries written in C/C++/CUDA, making them less accessible than algorithms written in Python
- GBTs algorithms not modular, harder to make iterative small alterations/improvement

When to use which approach/technique with a given dataset

These are my rules of thumb, and caveats could fill an entire book

1. Up to a few hundred datapoint, use stats
2. For few hundred to few thousand use linear/logistic regression
3. Between few thousand to about 10,000 it's anyone's guess. Gradient boosters generally do well here, with other "classical" algorithms (SVM for instance) sometimes shining.
4. Many thousands to about a billion datapoint is where gradient boosted trees rule. If you need just one algorithm, go with this. You'll never go wrong.
5. If you have several billion datapoint, or many times that amount, check out neural nets. They have the capacity to absorb those kinds of datasets easily.

The background of the slide features a series of overlapping, diagonal, light green bands that create a sense of depth and movement. The bands are slightly curved and have a soft gradient, giving the overall design a modern and dynamic feel.

XGBoost

A Brief Intro to XGBoost

- XGBoost is interchangeably used both for the particular algorithm (regularizing gradient boosting trees) and a software library
- XGBoost library was first released on March 27 2014 – almost exactly 10 years ago!
- It started as a research project by Tianqi Chen as part of the Distributed (Deep) Machine Learning Community
- Core library written in C++, with packages in many popular programming languages
- It was used as the winning solution for the Higgs Machine Learning Kaggle Challenge, after which it became very well known in ML circles
- In terms of popularity it's still the most installed and used GBT library
- Install base on par with most popular deep learning libraries

dmlc
XGBoost



What are decision trees?

- a type of supervised machine learning algorithm used for both classification and regression tasks
- they model decisions and their possible consequences as a tree-like structure, where each internal node represents a "test" on an attribute (e.g., whether a customer is older than 50 years)
- each branch represents the outcome of the test, and each leaf node represents a class label (decision) in classification tasks or a continuous value in regression tasks.
- the paths from root to leaf represent classification rules or regression paths.
- popular due to their simplicity, interpretability, and ability to handle both numerical and categorical data.
- they work by recursively splitting the dataset into subsets based on the feature that results in the highest information gain or the lowest impurity (Gini impurity or entropy), aiming to reach the most homogeneous subsets (or leaf nodes) possible.



What is boosting?

- an ensemble technique that aims to create a strong classifier from a number of weak classifiers.
- this is done by building a model from the training data, then creating a second model that attempts to correct the errors from the first model.
- continues this process, adding models until the training set is predicted perfectly or a maximum number of models are added.

What is boosting (cont.)

Key points about boosting include:

- 1.Sequential Training:** Boosting trains models in sequence. Each new model incrementally improves upon the previous ones.
- 2.Focus on Misclassifications:** Boosting methods adjust the weight of training instances based on the previous model's accuracy, giving more weight to incorrectly classified instances.
- 3.Weak Learners:** The models in the sequence are often weak learners, meaning they do only slightly better than random guessing at the classification task. Boosting combines these to create a strong overall model.
- 4.Reduction in Bias and Variance:** Boosting can reduce errors by reducing both the bias and the variance of the combined model.

What is gradient boosting?

- builds on the logic of boosting, creating a highly accurate prediction rule by combining many weak and inaccurate rules/models.
- involves the optimization of an arbitrary differentiable loss function
- builds the model in a stage-wise fashion, with each new model being trained to minimize the loss function, using the gradient descent algorithm.
- Gradient Descent Step: in each step, a tree is built to predict the negative gradient of the loss function with respect to the previous model. In essence, it's using the gradient of the error to tell us how to change our model's predictions to improve accuracy.

What is eXtreme gradient boosting?

- Efficient and more advanced form of gradient boosting
- Optimized for both speed and performance
- Built-in mechanisms to prevent overfitting
- Robust way to deal with missing values in the data
- Efficiently handles sparse data
- Parallel and distributed processing

**Accelerated computing, scalable
machine learning, data centers, and
all of that**

Accelerated computing, scalable machine learning, data centers, and all of that

GPUs are not just for neural networks

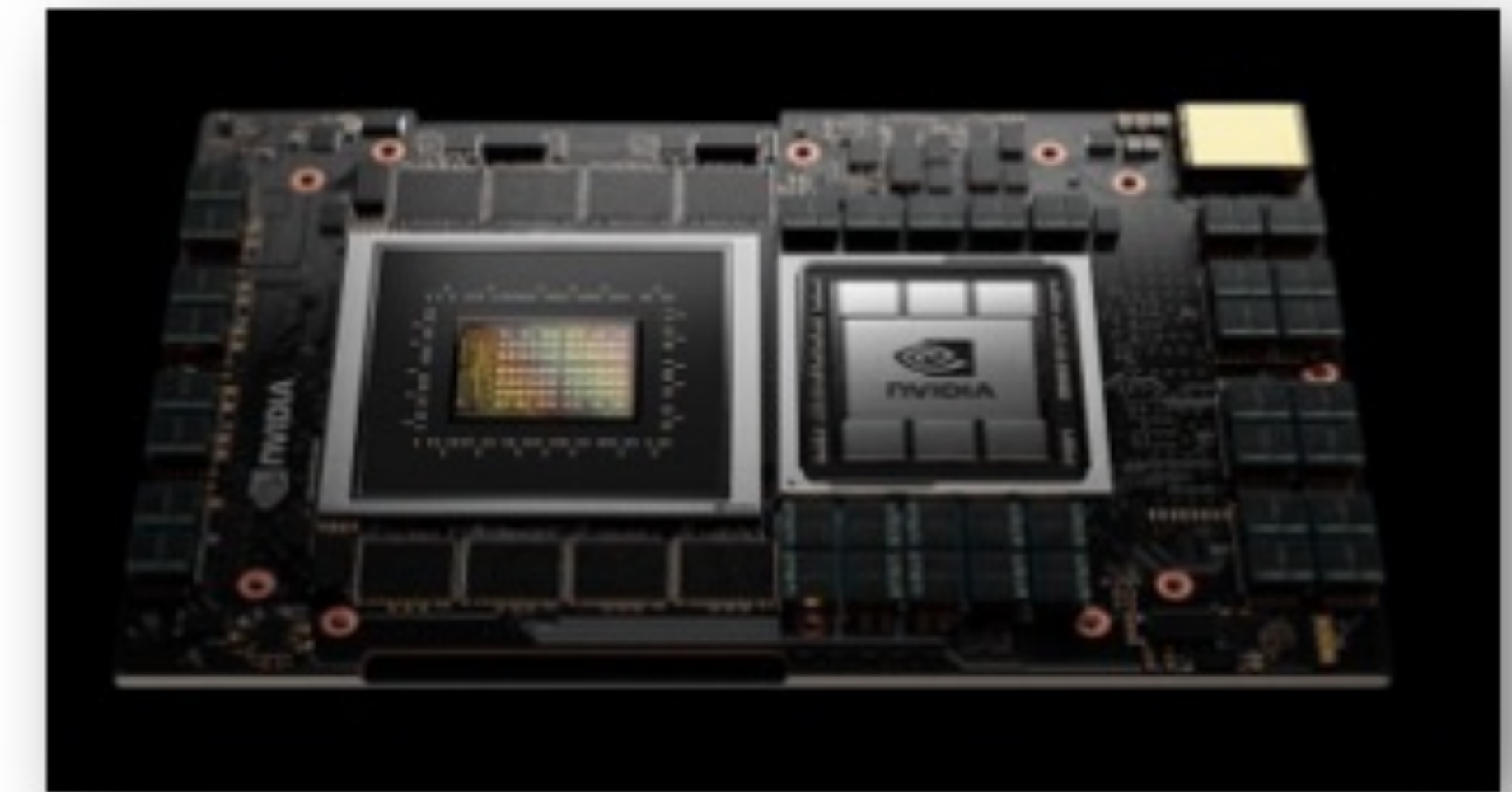
- Speedup of many “classical” machine learning and data science algorithms
- Rapids, XGBoost, etc.
- Many algorithms that were once considered too slow or unwieldy are having second life thanks to GPU acceleration – SVMs have been accelerated by several orders of magnitude for instance
- GPU-accelerated Shapley values are having a major impact on interpretable Machine Learning
- GPU-accelerated Shapley **interaction** values – a great source of information for feature engineering

Mission: accessible accelerated computing

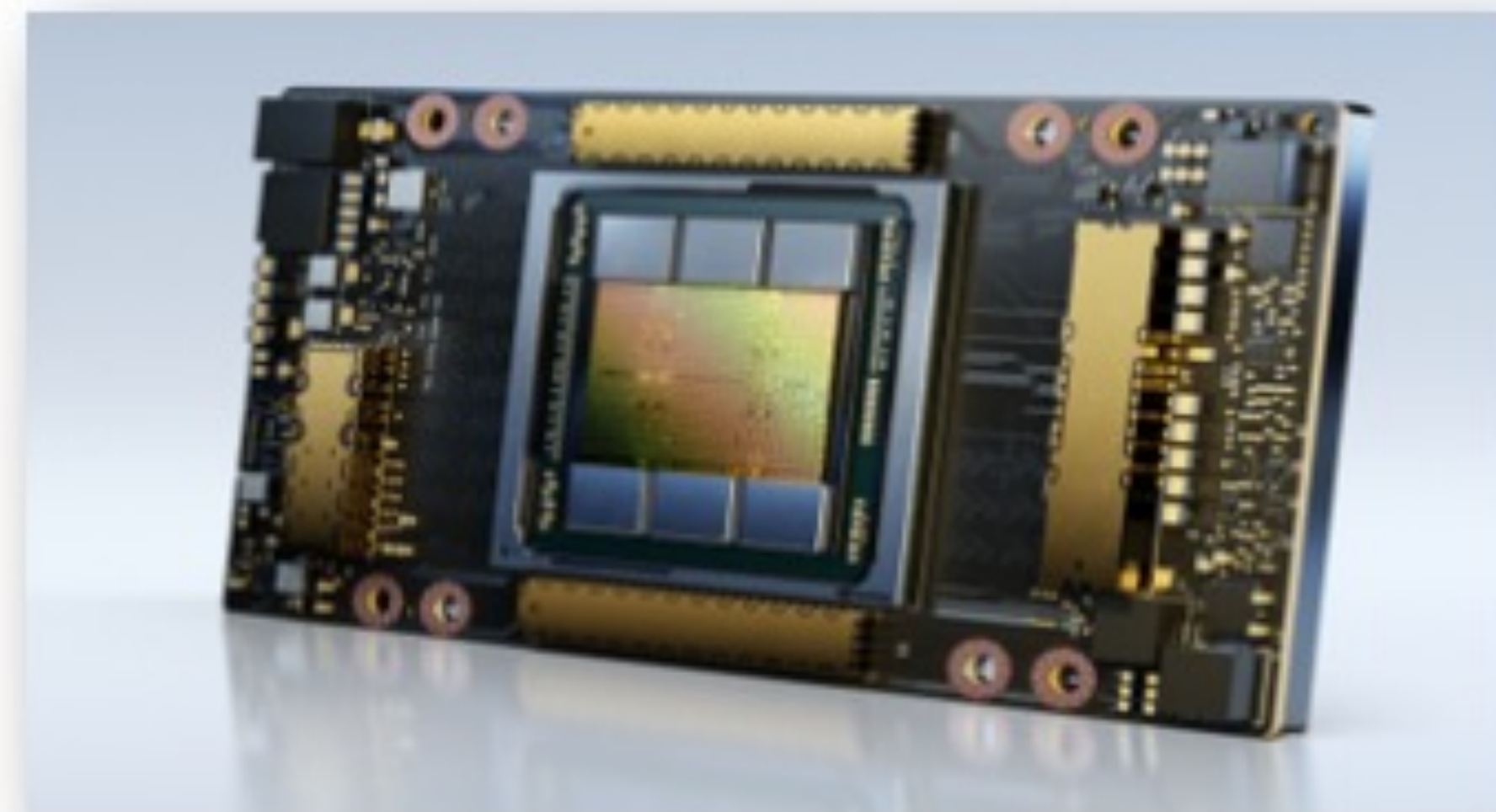
Accelerated computing with **no pain of complex distributed programming**

Office Online Frame

```
# Generate a random positive semi-definite matrix
A = scipy.sparse.random(n, n, format="csr")
A = 0.5 * (A + A.T) + n * scipy.sparse.eye(n)
# Estimate the maximum eigenvalue of A
x = numpy.random.rand(A.shape[0])
for _ in range(iters):
    x = A @ x
    x /= numpy.linalg.norm(x)
result = numpy.dot(x.T, A @ x)
```



Grace Hopper Superchip



GPU



DGX



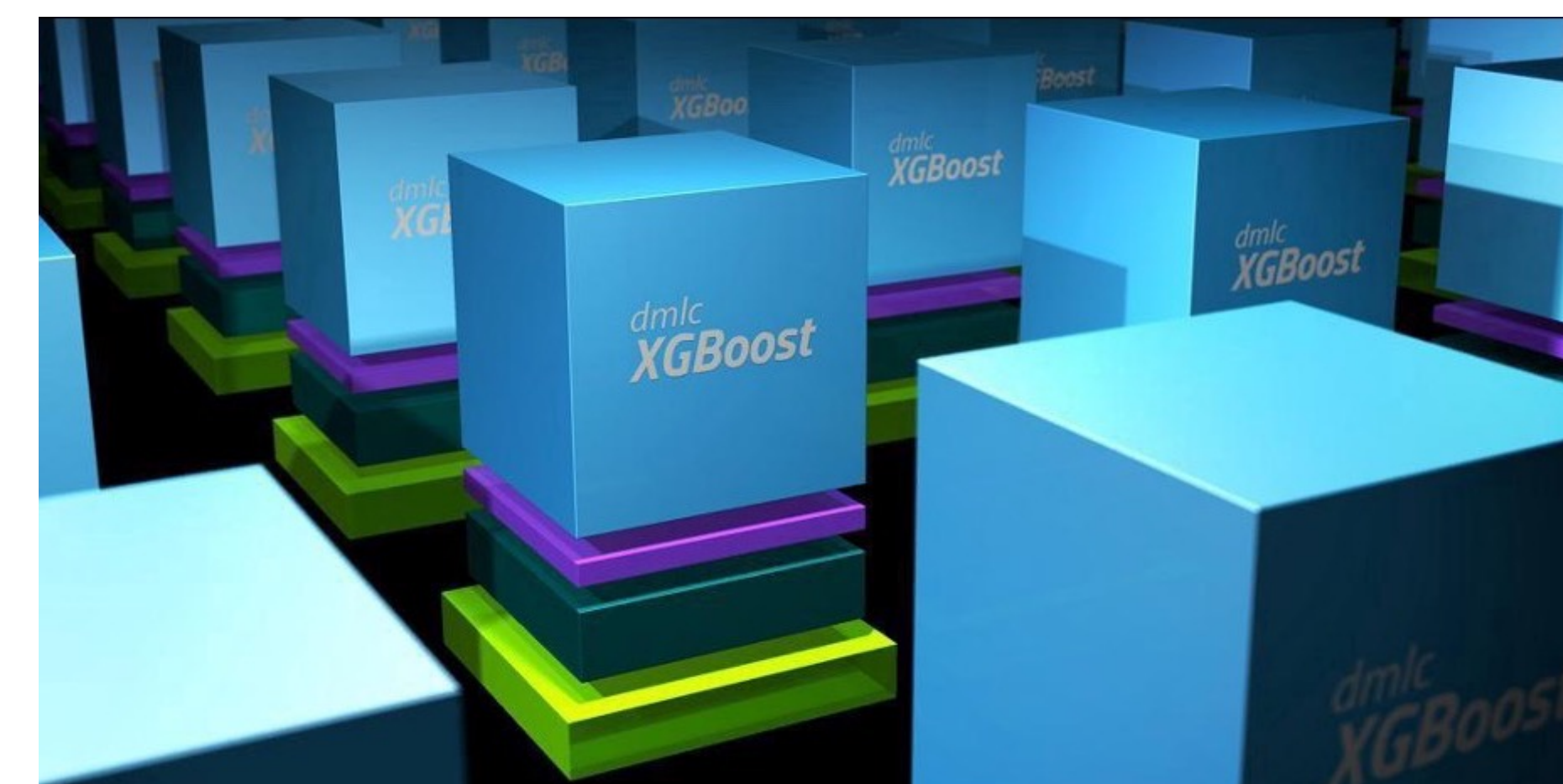
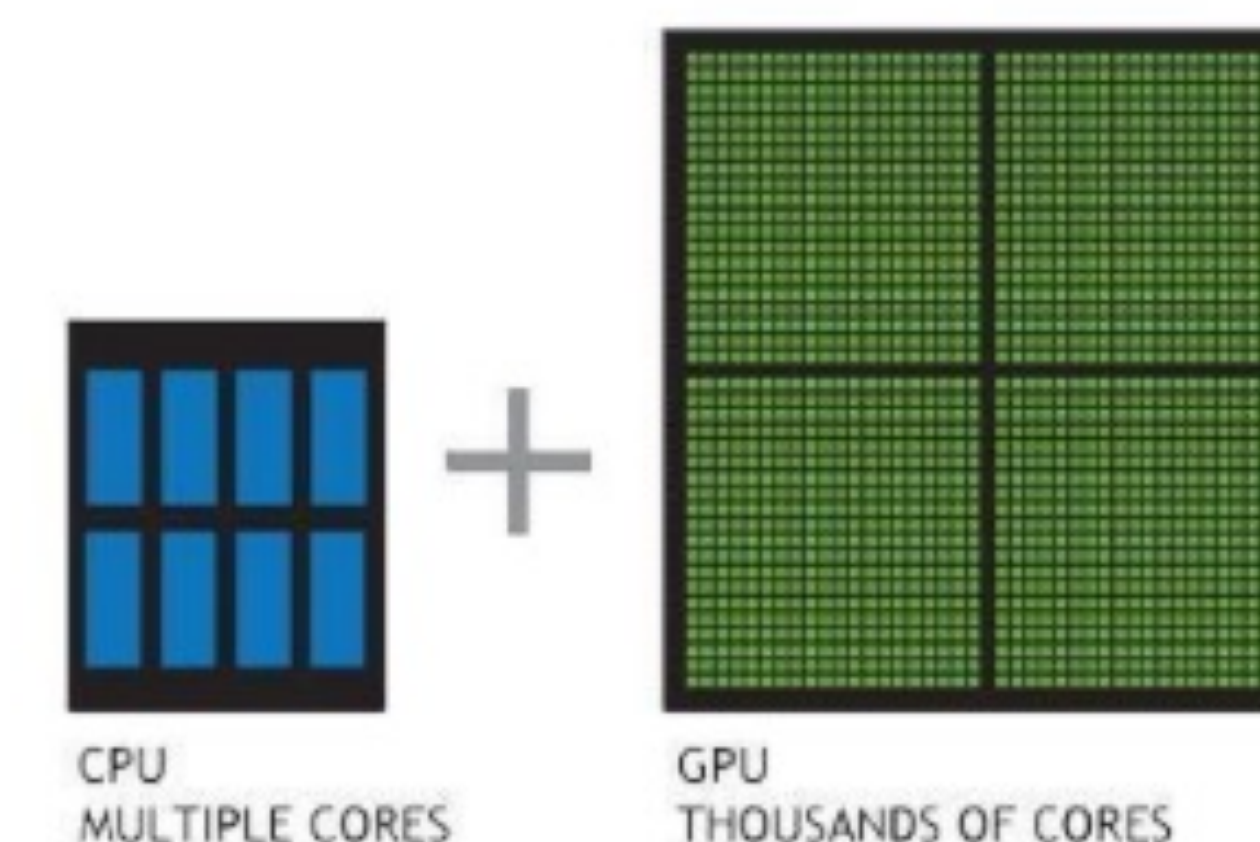
DGX Cloud
DGX SuperPOD



Nontrivial use case 1: multi-GPU and multi-machine training

XGBoost distributed computing and GPU Support

- (CUDA) GPU support first introduced in 2017 with the 0.7 version of XGBoost
- Version 0.9, released in 2019, introduced multi-GPU support
- Version 1.0 of XGBoost introduces use of Dask for distributed computing
- Scaling across clusters for preprocessing and training
- With version 1.4 all multi-GPU and multi-machine support is now handled with Dask



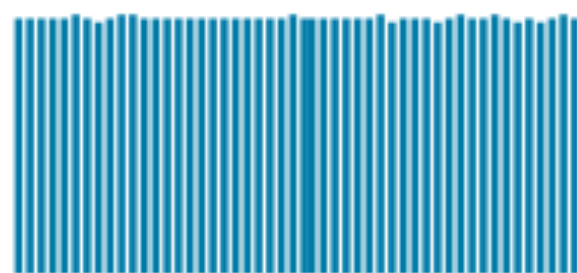



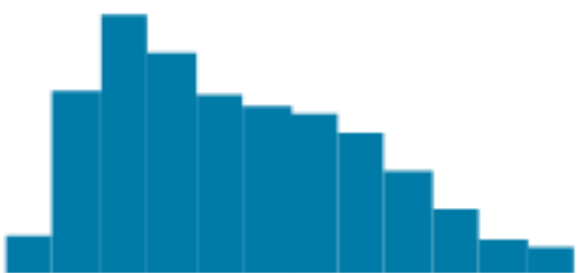

XGBoost and Dask for hyperparameter optimization

<https://www.kaggle.com/competitions/porto-seguro-safe-driver-prediction/>

Porto Seguro's Safe Driver Prediction

Predict if a driver will file an insurance claim next year.



train.csv (115.85 MB)						⬇	🔍	>
DetailCompactColumn						10 of 59 columns ▾		
id	# target	# ps_ind_01	# ps_ind_02_cat	# ps_ind_03	# ps_ind_04			
 71.49m	 01	 07	 -14	 011	 -1			
7	0	2	2	5	1			
9	0	1	1	7	0			
13	0	5	4	9	1			
16	0	0	1	0	0			

XGBoost and Dask for hyperparameter optimization

Tools we use

- DGX H100
- XGBoost
- Dask
- Optuna
- All code can be found here:

https://github.com/tunguz/TabularBenchmarks/tree/main/datasets/Porto_Seguro

In [3]: `%watermark --gpu`

```
GPU Info:
GPU 0: NVIDIA A100-SXM4-80GB
GPU 1: NVIDIA A100-SXM4-80GB
GPU 2: NVIDIA A100-SXM4-80GB
GPU 3: NVIDIA A100-SXM4-80GB
GPU 4: NVIDIA A100-SXM4-80GB
GPU 5: NVIDIA A100-SXM4-80GB
GPU 6: NVIDIA A100-SXM4-80GB
GPU 7: NVIDIA A100-SXM4-80GB
```

In [4]:

```
from dask.distributed import Client
from dask_cuda import LocalCUDACluster
from dask import dataframe as dd
from dask.delayed import delayed
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.model_selection import KFold
from sklearn.metrics import roc_auc_score

import optuna
import gc
import logging
```

In [5]: `%watermark --iversions`

```
logging: 0.5.1.2
numpy   : 1.24.4
pandas  : 1.5.3
xgboost: 1.7.6
optuna  : 3.3.0
dask    : 2023.3.2
```


Setting up Dask cluster and loading files

In [7]:

```
cluster = LocalCUDACluster(n_workers=8)
client = Client(cluster)
```

In [10]:

```
train_folds = []
val_folds = []
train_ys = []
val_ys = []

for i in range(5):
    print(f'Loading fold {i}')
    train_fold_d = delayed(pd.read_csv)(f'../input/xgtrain_fold_{i}.csv.gz')
    train_fold = dd.from_delayed(train_fold_d)

    val_fold_d = delayed(pd.read_csv)(f'../input/xgval_fold_{i}.csv.gz')
    val_fold = dd.from_delayed(val_fold_d)

    train_y = train_fold['target']
    train_fold = train_fold[train_fold.columns.difference(['target'])]

    val_y = val_fold['target']
    val_fold = val_fold[val_fold.columns.difference(['target'])]

    train_folds.append(train_fold)
    val_folds.append(val_fold)

    train_ys.append(train_y)
    val_ys.append(val_y)
```


Optuna objective function

```
In [12]: train_oof = np.zeros((target.shape[0],))

num_round = 1000

def objective(trial):

    params = {
        'objective': trial.suggest_categorical('objective', ['binary:logistic']),
        'tree_method': trial.suggest_categorical('tree_method', ['gpu_hist']), # 'gpu_hist', 'hist'
        'lambda': trial.suggest_loguniform('lambda', 1e-3, 10.0),
        'alpha': trial.suggest_loguniform('alpha', 1e-3, 10.0),
        'colsample_bytree': trial.suggest_uniform('colsample_bytree', 0.3, 1.0),
        'subsample': trial.suggest_uniform('subsample', 0.4, 1.0),
        'learning_rate': trial.suggest_loguniform('learning_rate', 0.001, 0.1),
        #'n_estimators': trial.suggest_categorical('n_estimators', [1000]),
        'max_depth': trial.suggest_int('max_depth', 3, 25),
        #'random_state': trial.suggest_categorical('random_state', [24, 48, 2020]),
        'min_child_weight': trial.suggest_int('min_child_weight', 1, 300),
        'eval_metric': trial.suggest_categorical('eval_metric', ['logloss']),
    }

    kf = KFold(5, shuffle=True, random_state=137)

    for i, (train_index, val_index) in enumerate(kf.split(train, target)):
        dtrain = xgb.dask.DaskDMatrix(client, train_folds[i].values, train_ys[i], enable_categorical=True)
        dval = xgb.dask.DaskDMatrix(client, val_folds[i].values, val_ys[i], enable_categorical=True)

        output = xgb.dask.train(client, params, dtrain, num_round)
        booster = output['booster'] # booster is the trained model
        booster.set_param({'predictor': 'gpu_predictor'})
        predictions = xgb.dask.predict(client, booster, dval)
        predictions = predictions.compute()
        train_oof[val_index] = predictions
        del dtrain, dval, output
        gc.collect()
        gc.collect()

    gini = gini_normalized(target, train_oof)

    return gini
```


Logging and optimizing

- Each trial takes between 30 s and 1 min

In [13]:

```
logger = logging.getLogger()
logger.setLevel(logging.INFO) # Setup the root logger.
logger.addHandler(logging.FileHandler("optuna_xgb_output_0.log", mode="w"))

optuna.logging.enable_propagation() # Propagate logs to the root logger.
optuna.logging.disable_default_handler() # Stop showing logs in sys.stderr.

study = optuna.create_study(storage="sqlite:///xgb_optuna_allstate_0.db", study_name="five_fold_optuna_xgb_0", di
```

In [14]:

```
%%time
logger.info("Start optimization.")
study.optimize(objective, n_trials=3)
```


Dask Training on a Cluster

- Can be done on a CPU or a CUDA cluster
- To create a cluster from the command line, Dask-CUDA workers must be connected to a scheduler started with dask scheduler:

```
$ dask scheduler
distributed.scheduler - INFO - Scheduler at: tcp://127.0.0.1:8786

$ dask cuda worker 127.0.0.1:8786
```

- To connect a client to this cluster:

```
from dask.distributed import Client

client = Client("127.0.0.1:8786")
```


XGBoost: train anywhere, deploy anywhere

- Model trained on DGX H100 server
- Model weights saved as a json file
- Model loaded and inference done on Raspberry Pi Zero

```
import pandas as pd
(1000, 217)
Loading the test file took 0.2480621337890625 seconds
Loading the XGBoost model file took 2.5075156688690186 seconds
XGBoost inference took 0.3106880187988281 seconds
tupou7@raspberrypi:~$
```




Nontrivial use case 2: use of Shapely values for feature selection and feature engineering

Shapley Values

The power of interpretability

- Shapley Values are unique way, based in game theory, of ascribing feature importances to an ML Model
- Allows individual attribution of each feature for each data point
- **Extremely** computationally demanding to calculate - the exact calculator grows exponentially with a number of features
- Most Shapley Value packages use some kind of approximation
- GPUs accelerate computation dramatically
- Use of Shapley Values for Unsupervised Learning
- Shapely Interaction Values - Feature Engineering

Nice Shapley Properties for model predictions

1. Additivity - the sum of all Shapley values adds up to the prediction
2. Symmetry - permuting features does not change their Shapley Values
3. Dropping features will not change the relative importance of the remaining features

Shapley values might be the only feature explanation scheme where all these things hold

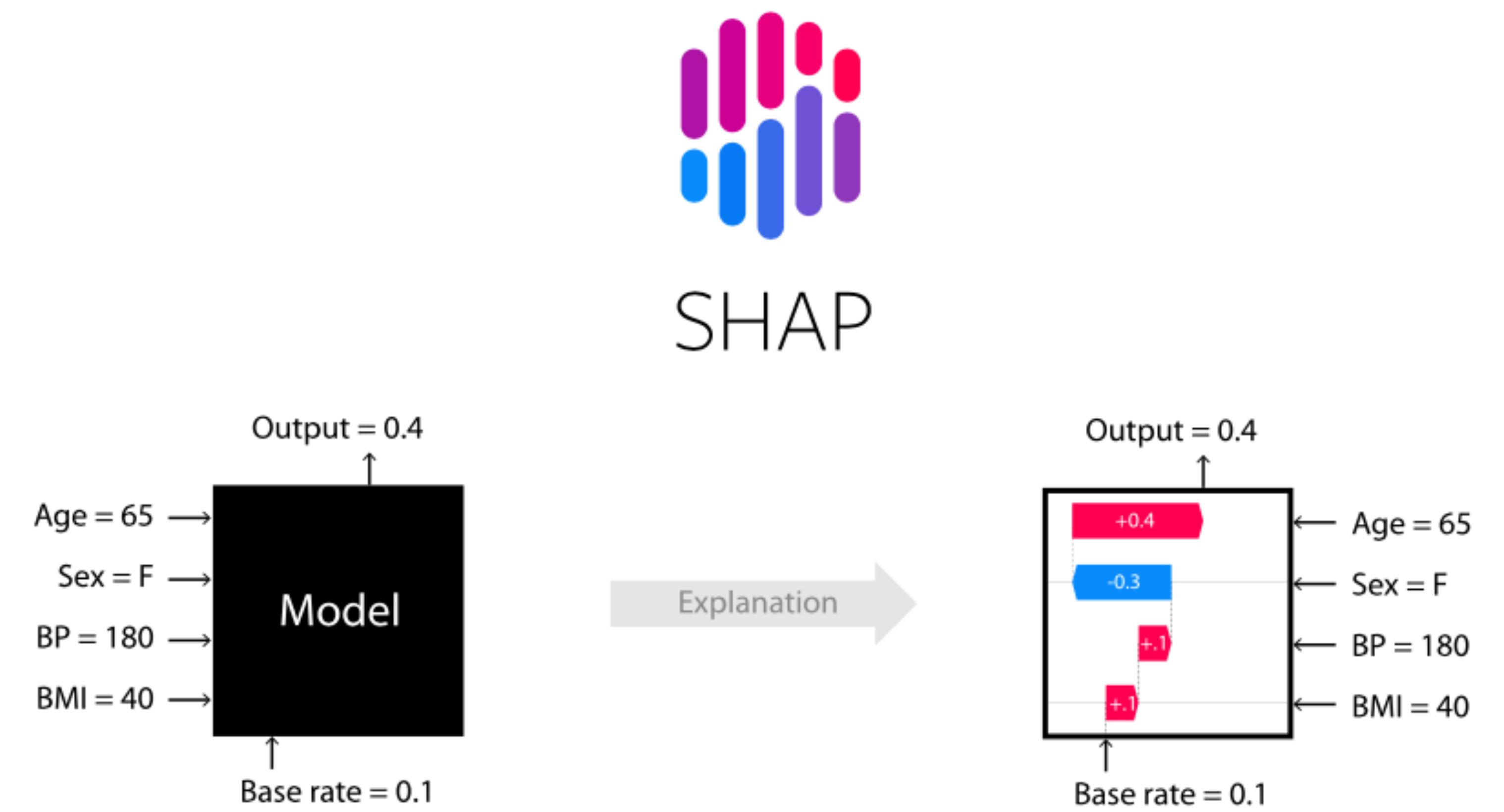
As model explainability becomes more and more important, and is required by law in many jurisdictions, Shapley Values might become the only legally valid way to explain a model

Shapley values in XGBoost

- XGBoost standard Python API allows for their direct calculation
- Same with Shapley interaction values
- GPU accelerated Shapley value calculations available since the version 1.3 that was released in late 2020
- GPU acceleration a **major** game changer for usefulness of Shapely values, especially with larger datasets
- Speedups often on the order of 500X – 1,000X compared to the CPU version
- GPUShap - cuda implementation of the TreeShap algorithm by Lundberg et al. for Nvidia GPUs.

SHAP package

- Very nice and user-friendly Shapley values Python package
- High speed exact algorithm for tree ensemble models (XGBoost, LightGBM, CatBoost, etc.)
- can explain output of any ML algorithm
- very nice visualizations, works with javascript in Jupyter



Shapely values with Porto and XGBoost

In [23]:

```
%%time  
shap_preds = model.predict(dval, pred_contribs=True)
```

CPU times: user 1.32 s, sys: 389 ms, total: 1.71 s
Wall time: 1.7 s

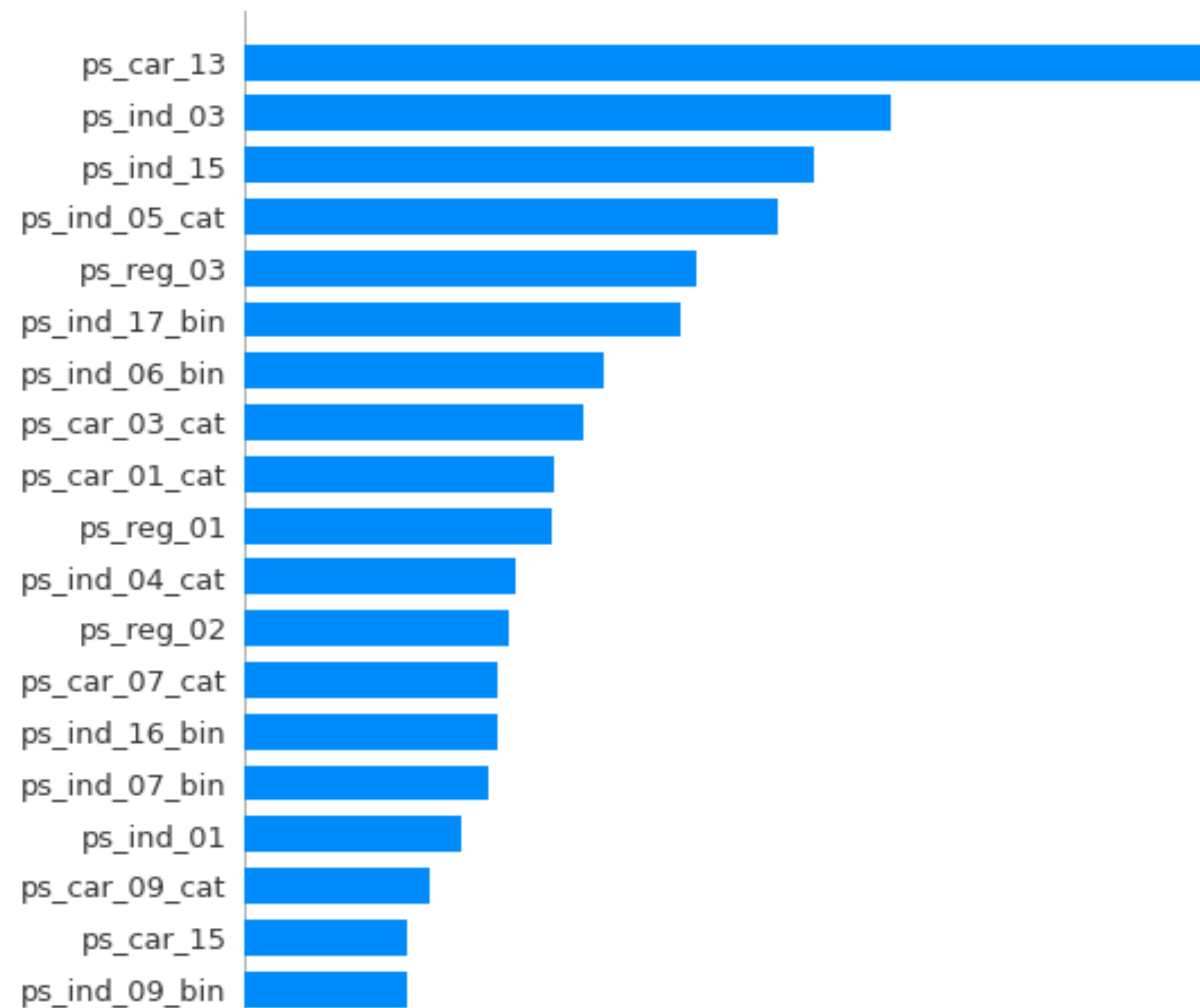
In [24]:

```
%%time  
shap_interactions = model.predict(dval, pred_interactions=True)
```

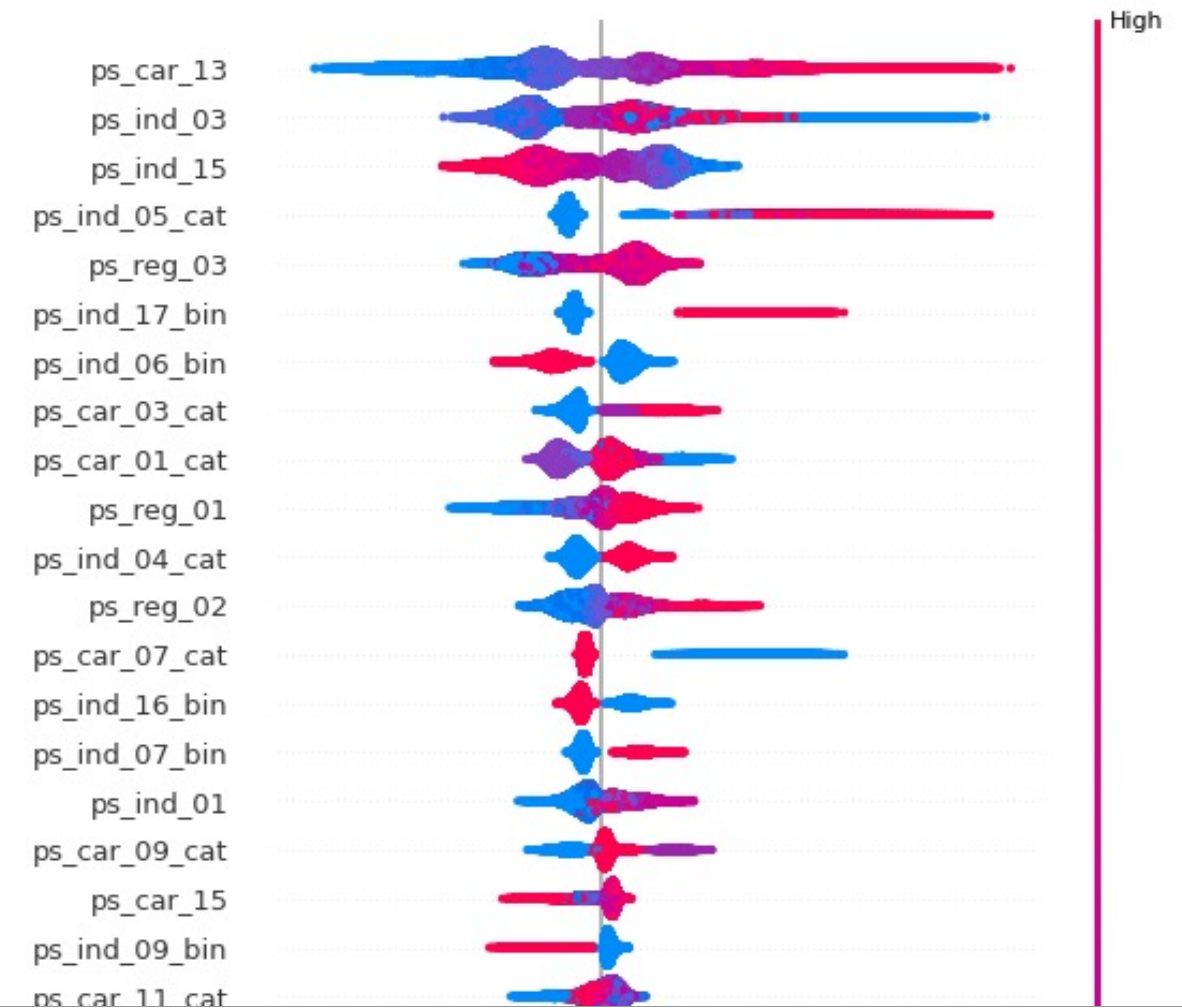
CPU times: user 13.4 s, sys: 4.49 s, total: 17.9 s
Wall time: 17.9 s

Shapely values with Porto and XGBoost

```
In [34]: shap.summary_plot(shap_preds[:, :-1], val_features, plot_type="bar", max_displ
```



```
In [30]: shap.summary_plot(shap_preds[:, :-1], val_features, max_display=57)
```



Dropping unimportant features and retraining

- Val score improves 0.2851 -> 0.2859
- No re-optimizing of the hyperparameters

```
In [40]: np.sort(mean_abs_shap_values)

Out[40]: array([ 0.         ,  0.         ,  0.         ,  0.         ,  0.         ,
        0.         , 187.95482, 198.37328, 315.82446, 369.79617,
        386.58417, 446.84824, 502.49487, 524.928  , 553.32733,
        607.4125 , 615.64545, 648.3115 , 673.9844 , 703.20557,
        810.2731 , 973.8078 , 983.38696, 1053.3447 , 1094.644  ,
        1136.4413 , 1161.1857 , 1261.1857 , 1284.7091 , 1287.981  ,
        1312.7737 , 1427.5542 , 1619.1409 , 1633.7672 , 1768.4987 ,
        2113.4912 , 2784.3406 , 2903.5469 , 3047.0662 , 3048.4797 ,
        3468.1892 , 4098.422  , 4584.218  , 4756.268  , 4763.656  ,
        4975.725  , 5081.0654 , 5756.228  , 5829.0474 , 6358.967  ,
        6752.292  , 8206.366  , 8490.776  , 10002.108 , 10699.994  ,
        12136.905 , 17992.69  ], dtype=float32)

In [43]: faulty_columns = np.where(mean_abs_shap_values == 0)[0]

In [45]: features[faulty_columns]

Out[45]: Index(['ps_ind_10_bin', 'ps_ind_11_bin', 'ps_ind_12_bin', 'ps_ind_13_bin',
               'ps_ind_14', 'ps_car_10_cat'],
              dtype='object')
```

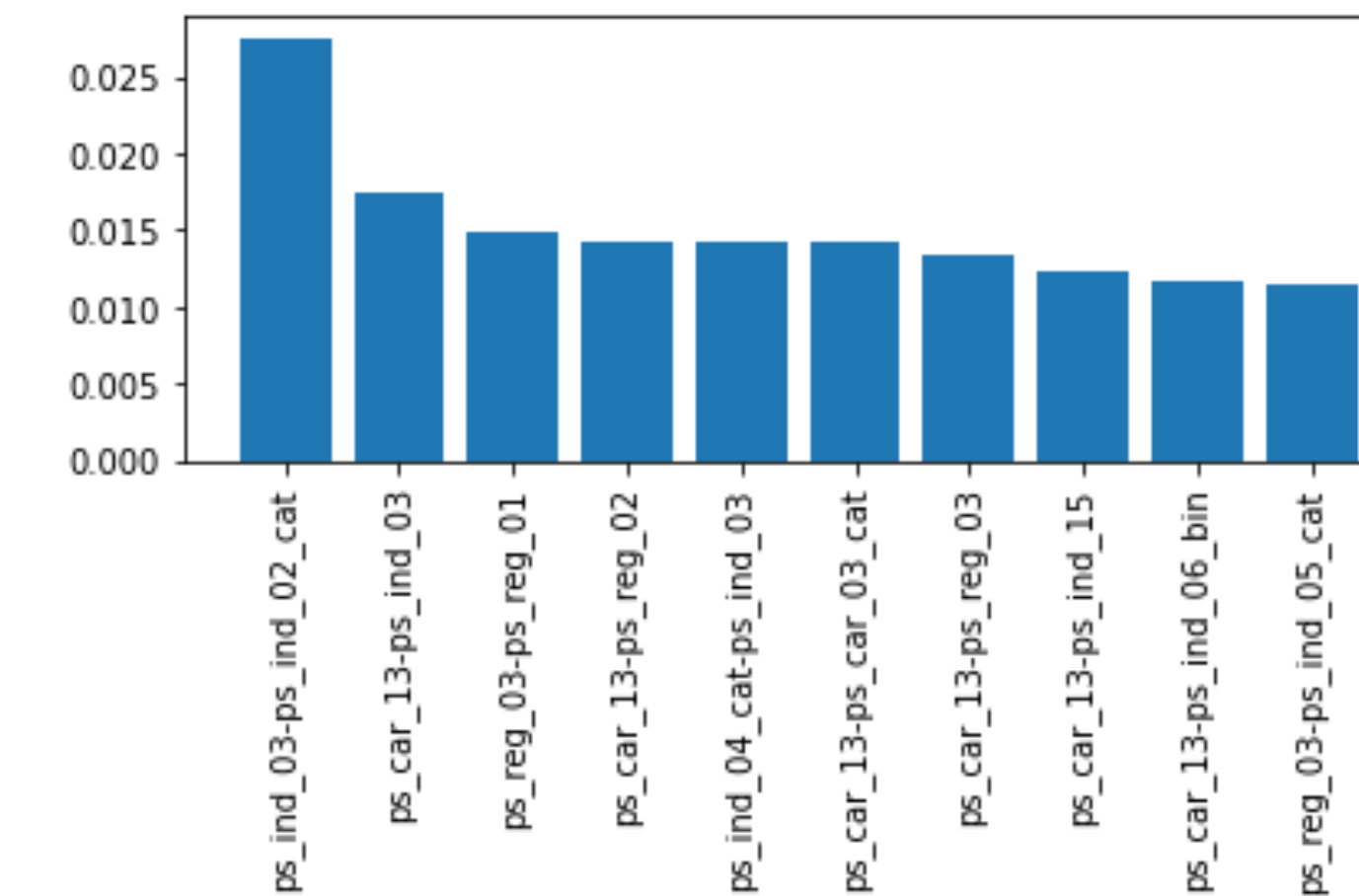

Shapely interactions and FE

```
In [67]: train['ps_ind_03-ps_ind_02_cat'] = train['ps_ind_03']*train['ps_ind_02_cat']
```

```
In [86]: train['ps_car_13-ps_ind_03'] = train['ps_car_13']*train['ps_ind_03']
```

Val improves 0.2859 -> 0.2871

```
In [51]: def plot_top_k_interactions(feature_names, shap_interactions, k):  
# Get the mean absolute contribution for each feature interaction  
aggregate_interactions = np.mean(np.abs(shap_interactions[:, :-1, :-1]), axis=0)  
interactions = []  
for i in range(aggregate_interactions.shape[0]):  
    for j in range(aggregate_interactions.shape[1]):  
        if j < i:  
            interactions.append(  
                (feature_names[i] + "-" + feature_names[j], aggregate_interactions[i][j] * 2))  
  
# sort by magnitude  
interactions.sort(key=lambda x: x[1], reverse=True)  
interaction_features, interaction_values = map(tuple, zip(*interactions))  
plt.bar(interaction_features[:k], interaction_values[:k])  
plt.xticks(rotation=90)  
plt.tight_layout()  
plt.show()  
  
return interactions  
  
top_interactions = plot_top_k_interactions(features, shap_interactions, 10)
```





Nontrivial use case 3: use of XGBoost for unsupervised tasks

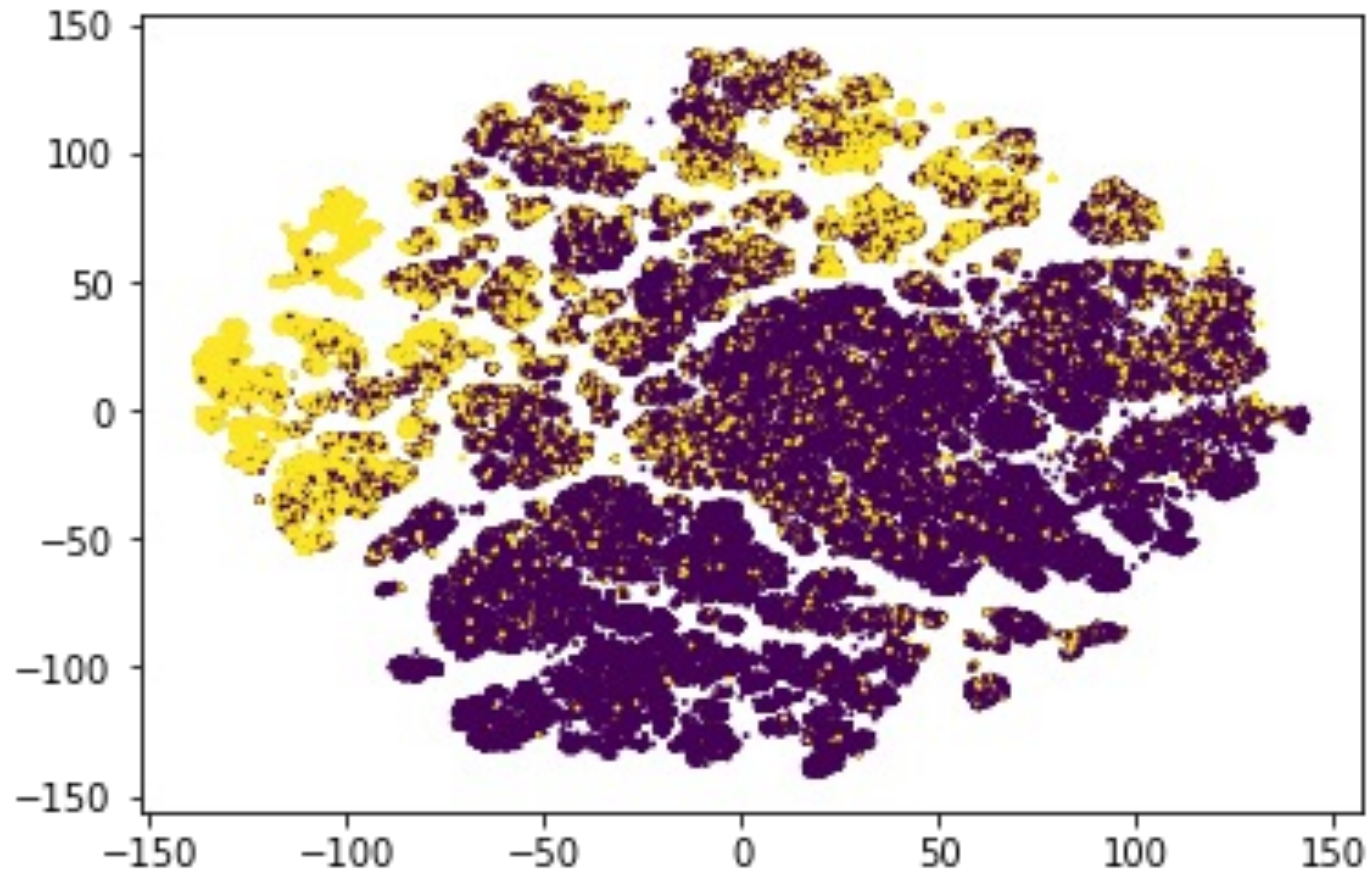
Visualization and Clustering

- Tabular datasets tend to be hard to be tricky to visualize and interpret
- Use of Shapley values from the **supervised** model for **unsupervised** tasks
- Why use Shapley values for unsupervised ML tasks?
- One convenient feature of Shapely Values - they come with a predetermined “scale”
- Essentially the same number of features as the original dataset
- No need to experiment with the scaling of features
- No need to worry about how to encode categorical features
- No need to worry about the missing values (for tree-based models)
- Next step: build a new **auxiliary** XGBoost model for cluster interpretability

Simple Visualization and Clustering Procedure

- For a **supervised** learning task – create a simple XGBoost model (don't worry about predictive performance)
- Get Shapley values for all datapoints
- Use a simple dimensionality reduction scheme (t-SNE, UMAP) for visualization of features
- Use a clustering algorithm to extract clusters
- Other dimensionality reduction schemes: neural nets and autoencoders

Example – t-SNE



Interpreting the clusters with XGBoost

- Assign labels to all clusters
- Build an XGBoost classifier to predict those clusters
- Calculate Shapley values for the model
- Use those values for feature importances
- Find top feature importances for each cluster
- Interpret those clusters based on those importances
- Example use case: customer segmentation

Why XGBoost and not simpler statistical methods?

- In principle, all of the above could be done with lots of careful statistical analysis
- However, that analysis often requires lot of time and/or expertise, both statistical and subject matter
- Straightforward procedure with powerful computational tool can get you pretty far with relatively low effort

Beyond supervised tasks

- What about **pure** unsupervised tasks?
- XGBoost based autoencoders?
- Dataset embeddings with XGBoost?

Conclusion

- XGBoost is a powerful, versatile, well supported, and scalable Machine Learning library
- Well suited for a vast variety of tabular Machine Learning tasks and workloads
- It will continue to be relevant in the age of LLMs and AI



Miscellaneous topics and Q&A