Data Pre-processing

ALS User Meeting

September 11, 2023

Tanny Chavez, **Tibbers Hao**, Alex Hexemer, Wiebke Koepp, Dylan McReynolds, Eric Roberts, Zhuowen Zhao, and Petrus Zwart

Why?

Default (GPT-3.5)

GH

Tell me, what's the dummy answer of why we need data preprocessing for ML?

The dummy answer for why we need data preprocessing in machine learning is: "Because computers are picky eaters and need their data served in a specific way. Data preprocessing is like making a fancy meal for a finicky toddler – you have to chop, clean, and arrange everything just right before they'll even consider taking a bite. Similarly, machine learning algorithms demand data that's properly formatted, cleaned, and transformed so they can digest it and spit out useful predictions. Without preprocessing, it's like feeding them a messy, unappetizing plate of data slop, and you won't get the delicious results you're hoping for!"

If garbage goes in, garbage comes out

Professional data scientists spend 80% of their time on preparing and managing data.

Just like us, we spend quite a lot of time tuning and aligning beams before the experiment. Why? Cause you don’t want to get garbage out

What to do?

Preparing data is more or less like cooking

- Missing ingredients
- Throw out the "bad" stuff
- arrange to the good size
- Lactose Intolerance?
- Too salty? Too sweet?
- Cooking a feast with only leftovers from fridge
- Squeeze out the juice
- I need 3 dishes
- Write down the secret recipe

- Dealing with missing values
- Cleaning Outliers and Anomalies
- Normalization / Standardization
- Handling Categorical Data
- Data Balancing
- Data Augmentation
-
- Feature Extraction / Dimensionality Reduction
- Data Splitting
- Save Processing Steps

How?

The bare minimum

- Check Size and Dealing with Missing Values
- Normalization / Standardization
- Save Processing Steps

Check Size

- First thing to do: know your data
- Most ML models require unified input size.
- Meanwhile, they can’t handle NaNs

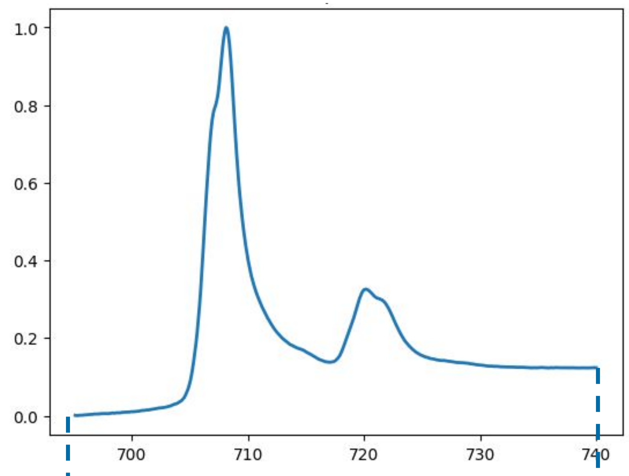
Handling Missing Values

- Ultimate solution: Drop
  - Think: can I afford the data loss?
- Replace with 0 or a certain number
  - Think: does that number affect behavior of your target?
- Impute values
  - Think: Which imputation method should I use?
  - Scikit-learn does have plenty options to do that

Example: You got a bunch of XAS Spectra

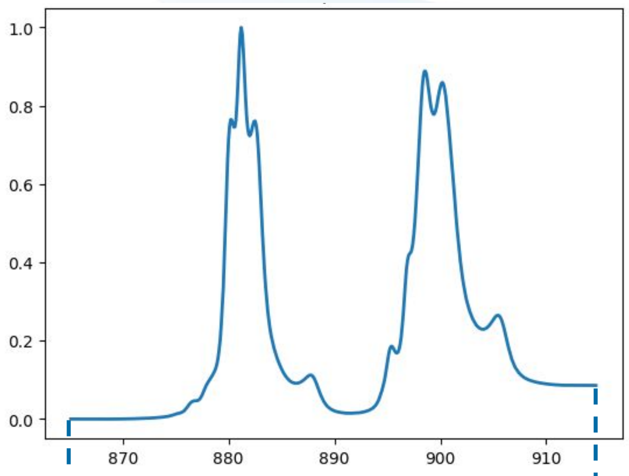
- Different energy range and length
- Goal: Bring them to the same size

Fe



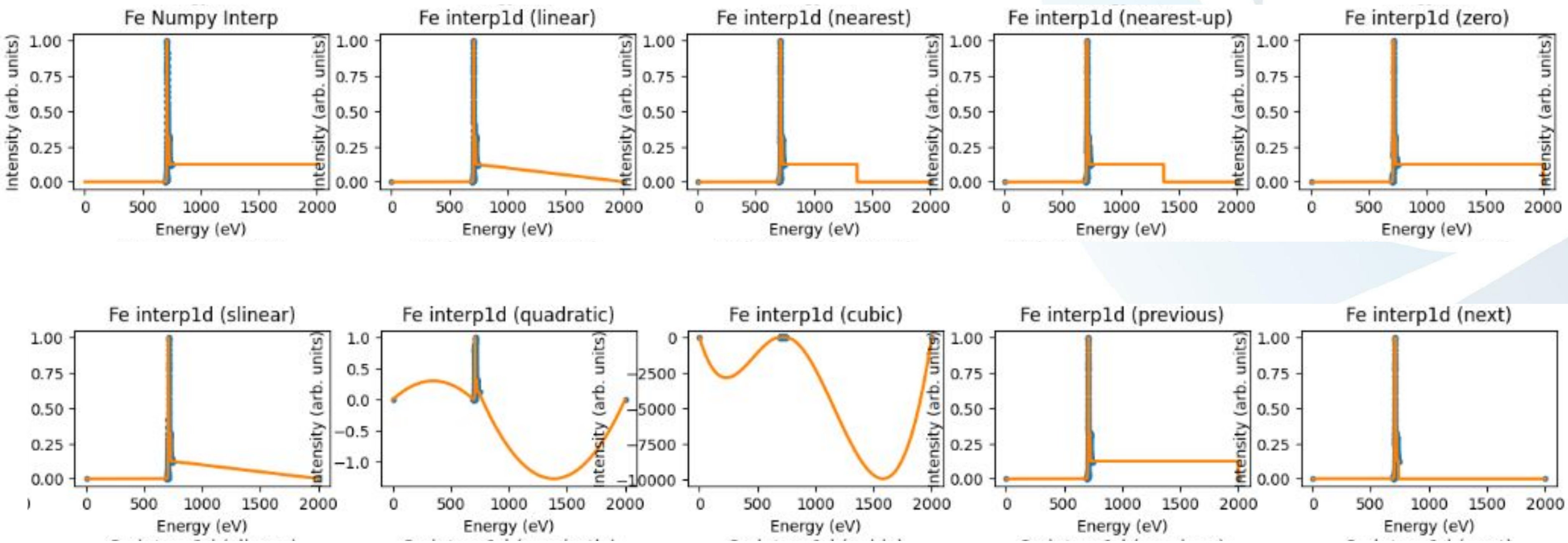
...

Ce



Example: You got a bunch of XAS Spectra

Both Numpy and Scikit-learn have options to do simple imputation



Normalization / Standardization

- This is critical, why?
  - Faster Gradient Descent Convergence
  - Maintain a consistent scale for the activations throughout the network
  - Make models more robust

- How to do that?
  - A lot of methods: Min-Max, Z-score, Log Transformation, Box-Cox …
  - We will see the most common two in notebook

Handling Categorical Data

- This is commonly happened to your labels
- Models don't know what is Fe, Ce, …
- But they know 0 and 1's
- You need to transfer that information into a way which computer can process
- How? Feature Encoding

Handling Categorical Data

- Way 1: Map your labels to an integer

	Small Particle	Medium Particle	Large Particle	WooW Monsters!	…
Ordinal Labels	0	1	2	3	…

- How: write a map function, or use sklearn LabelEncoder

- Be careful: Does your label have inherit orders?

Handling Categorical Data

- Way 2: One Hot Encoding

Element	Is Fe?	Is Cu?	Is Co?
Fe	1	0	0
Cu	0	1	0
Co	0	0	1

- Great for nominal labels (you can’t order those labels)
- How: from sklearn.preprocessing import OneHotEncoder

Write that Down

- Keep a good track of preprocessing step you do

- You will need to apply that to new data if you want to use your model

- Successful experience can be served as a good start for new pipelines

Before we Jump into the Notebook

- Identify and deal with outliers/anomalies
  - Throw out the “bad” stuff

- For spatial data, think about scaling.
  - Are your px and py the same across images?

- You may need to revisit your preprocessing steps during training if bad results keep popping up regardlessly.