5️⃣

From-scratch model

Learning steps

- [x] ~~yt video~~
- [x] ~~notebook/own implementation~~
- [x] ~~book chapter~~

Resources

- website 💻
    - lesson 5
- notebooks 🗒️
    - Linear model and neural net from scratch
    - Why you should use a framework
    - How random forests really work
    - Some great Titanic notebooks: 1; 234
- book 📘
    - chapter 4 and chapter 9
    - solutions to exercises

Quick notes

YouTube video (→ link)

- tabular problem with deeper level model
    - titanic problem
        - we used multiple simple models using Excel
        - this time, Python using the notebook on Kaggle (main folder on Jupyter → course22 → clean → 05...)
            - paperspace gradient notebook in JupyterLab

        - if running a notebook for a competition on Kaggle, it will automatically contain the data needed
- first step is cleaning the data
    - NA values
        - don't delete them all
            - why??
                - no reason to throw it away
                - maybe the fact that something is empty is a great predictor
            - ❗ you can call pandas methods on a dataframe → it will apply it on each row
        - imputing missing values
            - mode == most common value
            - iloc == position (row)
            - fillna()
    - describe() to see what's going on in the rows
        - some models don't like long-tail distributions (linear models, NN often)

          ➡️ use the log to normalize it (diminishes high numbers and less impact on low numbers)
        - unique()
        - describe by type of input in the rows
        - 🤔 we cannot multiply categorical variables by a coefficient (weight)
            - we had columns for each value of each categorical variable (one-hot encoding?)
    - to make computations on the columns, we have to turn them into pytorch's tensors
        - create one tensor for indep variables and one for depend variable
            - len(tensor) == rank
- setting up a linear model
    - we are going to multiply each value by a coefficient
        - nb of coeff == nb of features
            - we don't need a const here
        - we center them by removing 0.5 (they are between 0 and 1)
        - we do matrix*vector
            - numpy broadcasting: we multiply each element in the vector with each row of our tensor
                - 🩳 last dimension of the tensor and the coeffs should match
            - its really fast because it uses a low-level language (C, assembler) on optimized hardware
    - we would like each column's values in the same range

        - we'll use the max for each row, then divide the whole row by that value

          ⇒ easiest way to normalize
    - we now create predictions ⇒ we need a loss function to evaluate
        - we'll use *mean absolute error*
    - lastly we setup everything as a function
- doing a gradient descent step
    - we use the gradients to identify where we could reduce the loss thanks to modifications in the coefficients
        - learning rate is decided arbitrarily
- training the linear model
    - split data between training and validation set
        - we split the independent and dependent variables' tensors using the indices provided after the RandomSplitter
    - we know define functions using the previously defined code
    - we train the model!
        - what are the coefficients?
- measuring accuracy
    - accuracy is defined as the performance measure (❗ ≠ than the loss!)
    - we define our function to compute the accuracy
- using sigmoid
    - if we print the predictions, we see that we sometimes predict a chance of survival outside of [0, 1]

      ⇒ we use the sigmoid to squeeze the results between 0 and 1 (**sympy**)
    - we redefine the calc_preds function to use the sigmoid
- ❓ when using categorical variables and two main categories, the other less represented will usually be handled by FastAI by creating an 'Other' attribute
- submitting to Kaggle
    - bit a feature engineering and preprocessing
        - we create the 'Survived' column
- using matrix product
    - to use a NN, we use a matrix of coefficients instead of a vector
- a neural network
- deep learning (we add hidden layers)
    - we add two hidden layers
    - ❗ pay attention to the correct activation function at the final layer
    - here, deep learning didn't improve the accuracy

    - not necessarily adapted for this simple problem
    - for tabular data, feature engineering plays a crucial role
        - the more features and levels, the more sophisticated the ideal model
- why you should use a framework
    - it could be tedious to do feature engineering if you have to update the model each time ⇒ use a framework
    - TabularPandas is a really useful FastAI function for creating a dataloader

      ⇒ end of the preprocessing
    - we then create a learner with two hidden layers
        - we use fastai to find the ideal learning rate
    - we also have a function to apply the preprocessing steps on the test set
    - ensembling
- random forest