Advanced Machine Learning CIS550 Spring '24

Lab Homework 4

Submitted by:

Name: Manas Vishal (01971464)

Email: mvishal@umassd.edu

Title: Training a model

Training a model

In this lab we are going to learn how to train a model using XGBoost algorithm. In this lab we are going to use the biomedical dataset which was built by Dr. Henrique da Mota. We will be splitting the data into three subsets and those subsets will be used for training, validating and testing the model. The model we use is called XGBoost model. I read more about

XGBoost from here: XGBoost, which stands for Extreme Gradient Boosting, is a scalable, distributed gradient-boosted decision tree (GBDT) machine learning library. It provides parallel tree boosting and is the leading machine learning library for regression, classification, and ranking problems. [XGBoost – What Is It and Why Does It Matter? (nvidia.com)]

About the data

This biomedical dataset was built by Dr. Henrique da Mota during a medical residence period in the Group of Applied Research in Orthopaedics (GARO) of the Centre Médico-Chirurgical de Réadaptation des Massues, Lyon, France. The data has been organized in two different, but related, classification tasks.

The first task consists in classifying patients as belonging to one of three categories:

- *Normal* (100 patients)
- *Disk Hernia* (60 patients)
- *Spondylolisthesis* (150 patients)

For the second task, the categories *Disk Hernia* and *Spondylolisthesis* were merged into a single category that is labeled as *abnormal*. Thus, the second task consists in classifying patients as belonging to one of two categories: *Normal* (100 patients) or *Abnormal* (210 patients).

Each patient is represented in the dataset by six biomechanical attributes that are derived from the shape and orientation of the pelvis and lumbar spine (in this order):

- Pelvic incidence
- Pelvic tilt
- Lumbar lordosis angle
- Sacral slope
- Pelvic radius
- Grade of spondylolisthesis

The following convention is used for the class labels:

- DH (Disk Hernia)
- Spondylolisthesis (SL)
- Normal (NO)
- Abnormal (AB)

For more information about this dataset, see the [Vertebral Column dataset

webpage](http://archive.ics.uci.edu/ml/datasets/Vertebral+Column).

Loading the data

We load the data by following the lab tutorial and run the commands that also loads the necessary python modules.

```
1 import pandas as pd
2 import requests
3 import zipfile
4 import io
5 from scipy.io import arff
6 from sklearn.model_selection import train_test_split
7 import warnings
8 import os
9 warnings.filterwarnings("ignore")
10 from sklearn.metrics import accuracy_score
[1] ✓ 3.1s
```

Fig. 1 Loading all the python modules to initialize the notebook

We then download the data and store in a dataframe variable.

Fig. 2 Loading the data and generating a dataframe variable

We also examine the data by checking its dimensions and the features it has. In this case we have 6 features and a class column.

Fig. 3 Examining the data and its features

Using the encoding we learnt from last lab, we use a class mapper to get numerical values from class column. Here we map Abnormal to be 1 and Normal to be 0.

```
1 class_mapper = {b'Abnormal':1,b'Normal':0}
2 df['class']=df['class'].replace(class_mapper)

[6]
Python
```

Fig. 4 Class mapping

We now prepare the data for training. As mentioned in the lab tutorial there are several ways to split the data into the three datasets. One such way is to split the dataset into target and features. Then, both the datasets are divided into three subsets, that results in a total of 6 datasets.

We move the target column to the first position by the following command.

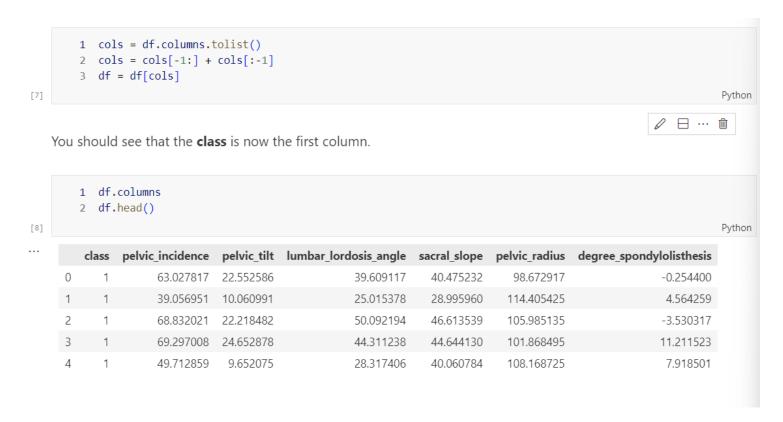


Fig. 5 Moving the target column to the first index.

We now use *scikit-learn* library to split the datasets into two subsets. One of them is used for training and the other is used for validation and testing. This is to ensure the training error is separate from the testing error. We use *train_test_split* function from the *scikit-learn* library. Since we don't have a lot of data, we should make sure to split datasets contain a representative amount of each class. For that we will use *stratify* switch with a random seed to repeat the splits later.

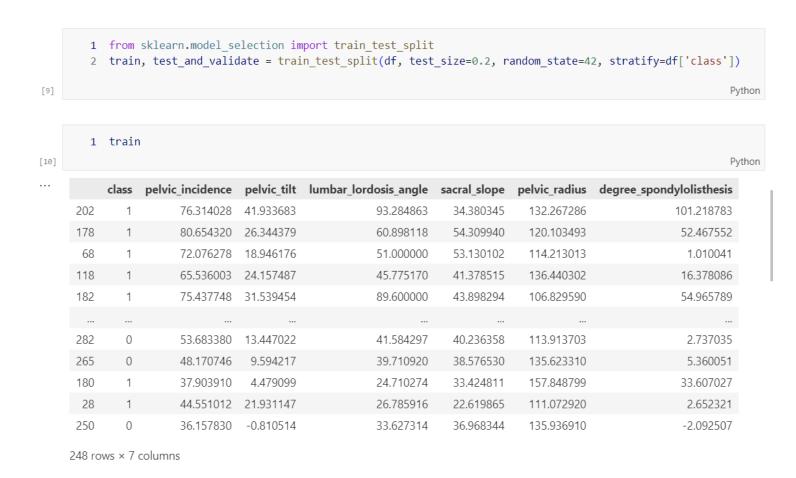


Fig. 6 Splitting the data with stratify

We now split the remaining dataset into two equal parts for testing and validating.

```
Next, split the test_and_validate dataset into two equal parts.

1 test, validate = train_test_split(test_and_validate, test_size=0.5, random_state=42, stratify=test_and_va

Python

Examine the three datasets.

1 print(train.shape)
2 print(test.shape)
3 print(validate.shape)

... (248, 7)
(31, 7)
(31, 7)
(31, 7)
```

Fig. 7 Splitting and examining the remaining dataset

```
1 print(train['class'].value counts())
        2 print(test['class'].value_counts())
        3 print(validate['class'].value_counts())
[13]
                                                                                                            Python
    class
    1 168
    0 80
    Name: count, dtype: int64
    class
    1
       21
       10
    Name: count, dtype: int64
    class
         10
    Name: count, dtype: int64
```

Fig. 8 Checking the distribution of classes

We now have to use xgboost to train the model but I did not have it installed locally so I installed with conda

```
1 !conda install xgboost -y
... Collecting package metadata (current_repodata.json): done
    Solving environment: done
     ==> WARNING: A newer version of conda exists. <==
      current version: 22.9.0
       latest version: 24.3.0
    Please update conda by running
         $ conda update -n base -c defaults conda
     ## Package Plan ##
       environment location: /root/miniconda3
       added / updated specs:
         - xgboost
     The following packages will be downloaded:
        package
                                                 build
     Preparing transaction: done
     Verifying transaction: done
     Executing transaction: done
    Retrieving notices: ...working... done
     Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Fig. 9 Installing XGBoost

We now import XGB classifier from xgboost and run fit to train the model.

```
1 from xgboost import XGBClassifier
        2 model = XGBClassifier(objective='binary:logistic', eval_metric='auc', num_round=42)
        3 print(model.fit(train.drop(['class'], axis = 1).values, train['class'].values))
        4 print("Training Completed")
[21] \( \square 0.4s
                                                                                                                Python
    [20:06:15] WARNING: /croot/xgboost-split_1675457761144/work/src/learner.cc:767:
    Parameters: { "num round" } are not used.
    XGBClassifier(base_score=None, booster=None, callbacks=None,
                   colsample bylevel=None, colsample bynode=None,
                   colsample_bytree=None, early_stopping_rounds=None,
                   enable_categorical=False, eval_metric='auc', feature_types=None,
                   gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
                   interaction constraints=None, learning rate=None, max bin=None,
                   max cat threshold=None, max cat to onehot=None,
                   max_delta_step=None, max_depth=None, max_leaves=None,
                   min_child_weight=None, missing=nan, monotone_constraints=None,
                   n_estimators=100, n_jobs=None, num_parallel_tree=None,
                   num round=42, predictor=None, ...)
    Training Completed
```

Fig. 10 Successful training of the model from xgboost

Conclusion

Concluding the lab on training a model with the XGBoost algorithm on a biomedical dataset, I would like to highlight the important aspects of advanced machine learning techniques and their practical implications. During the lab exercise, I reviewed data loading, feature exploration, class mapping, and dataset for training, testing, and validation. The use of the powerful machine learning algorithm XGBoost, known for its scalability and efficiency in gradient-boosted decision tree models, emphasizes the importance of advanced tool boost in the field of machine learning.

By emphasizing the significance of data exploration, preprocessing, model training, and evaluation, I sought to

provide a comprehensive guide for individuals looking to deepen their understanding of machine learning methodologies.

The integration of class mapping strategies, layered data techniques, and key Python libraries such as XGBoost and scikit-learn highlight the practical relevance of the lab exercise to real-world machine learning applications. In addition to providing practical information. With experience training datasets and laying the groundwork for future testing, validation, and evaluation tasks, I aim to provide readers with the fundamental knowledge and skills needed to navigate complex machine learning workflows. The lab is a valuable resource for beginners and experienced professionals alike, providing a structured approach to model development and validation.

In summary, this lab combines theoretical concepts with practical application strategies, enabling people to use advanced machines. learning techniques and datasets. By promoting a deeper understanding of the complexities involved in training robust and accurate predictive models, the lab aims to improve the reader's machine learning skills.