Data Analytics Project Report

A.Y. 2023/2024

Claudia Citera

claudia.citera@studio.unibo.it

Riccardo Evangelisti

riccardo.evangelist6@studio.unibo.it

INDICE

da aggiungere

1 Introduction

The following project was developed as part of the **Data Analytics** course of the Master's Degree in Computer Science at the University of Bologna.

The objective of the project is to carry out a data analytics study, which involves the implementation of all the analytical pipeline phases studied during the course:

- Data Acquisition
- Data Visualization
- Data Preprocessing
- Modeling
- Evaluation

The main purpose of this study is to recognize the year in which a song was published based on the features of its audio track.

The following functionalities were developed:

- Traditional non-deep supervised Machine Learning techniques
- Supervised ML techniques based on neural networks
- Supervised ML technique with deep models for TabularData

2 Data Acquisition

The data acquisition phase involves collecting the data that needs to be analyzed. Data can be acquired in various ways, including static acquisition, which was used in this project.

The dataset used in this project consists of a single csv file with 252175 rows and 91 columns. One of the columns represents the year of publication of the song, ranging from 1956 to 2009. All other columns contain floating-point numbers related to the audio track, making the entire dataset continuous. For this reason, regression models were used to solve the problem.

Before proceeding with further operations, the dataset was analyzed to check for missing or duplicate values. It was found that there are no missing values and only 52 duplicate rows. Since they are very few and mainly related to the most represented classes, we decided to remove them.

3 Data Visualization

Data visualization is a technique that aims to communicate data in a clear and effective manner through the use of graphs. It is useful for exploring data efficiently and discovering possible relationships.

In this section of the project, several graphs and visual representations were produced to provide an overview of the dataset in question. Since the dataset has a high number of features, we chose to focus only on some key information and represent it visually to make it easier to understand and analyze the data.

As a first step, we checked the correlation between the various columns to see if it was possible to remove any of them.

From Figure 3.1 we can notice that the dataset shows a low degree of correlation among its various features. Even the correlation with the Year column is very low, so it is irrelevant for our investigation. However, a closer examination of the top-left quadrant reveals a cluster of variables exhibiting stronger correlation. A zoomed-in view of this region is provided below (Figure 3.2) for further analysis.

> IMMAGINE CORRELAZIONE GRANDE come Figura 3.1
>
> > TITOLO Correlation between features

> IMMAGINE CORRELAZIONE PICCOLA come Figura 3.2
>
> > TITOLO Correlation close up

A very relevant piece of information is the distribution of the Year attribute, which reveals that the dataset is unbalanced. In fact, more recent songs (from around 1990 to 2009) are much more represented than older ones (from 1990 backwards). It is very likely that we will attempt Under Sampling and Over Sampling.

Then we checked the distribution of the other varibles and we noticed that all the attributes exhibit a **Gaussian distribution** with varying skewness and their means are centered around 0 (or close to it), except for the attribute `S0`.
The plots indicate that the distribution ranges differ a lot across the various columns. Therefore, it's probable that we will require a preprocessing model to **rescale** the data.

> IMMAGINE CORRELAZIONE GRANDE come Figura 3.3
>
> > TITOLO Distribution of S0-44

> IMMAGINE CORRELAZIONE GRANDE come Figura 3.
>
> > TITOLO Distribution of S45-89

4 Data Preprocessing

Before the actual use of supervised ML techniques, a data preprocessing phase was carried out. The dataset used in the project was divided into two parts: train and test, using an 70-30 split scheme. In addition, a seed was set to ensure that the dataset split is consistently reproducible in the future. The same test set is used to evaluate all models, in order to ensure consistent comparability of the results obtained by the various models.

On validation set we behaved differently based on the functionality:

- No validation set was created because cross-validation was performed on machine learning techniques (functionality 1)
- while working on dnn we obtained a validation set by taking 20% of the train set
- when we used the pytorch_tabular module (functionality 3), the models themselves created a validation set by default, using 20% of the train.

4.1 Dimensionality Reduction

Since it is a very large dataset without missing or duplicate values, we tried to reduce the number of columns to have a smaller dataset. Since PCA (Principal Component Analysis) works on the similarity between the various data and our data have a very low correlation in general, we opted for LDA (Linear Discriminant Analysis).

LDA is a supervised technique for dimensionality reduction and classification that uses the class labels of the data. It tries to find a linear projection of the data that separates the classes well and reduces the variation within each class. To do this, it computes the mean vectors for each class and the overall mean vector, and then calculates the within-class scatter matrix and the between-class scatter matrix. It then solves the generalized eigenvalue problem for the matrix inverse of the within-class scatter matrix times the between-class scatter matrix, and chooses the top k eigenvectors as the new basis. Finally, it transforms the data onto the new subspace using the eigenvector matrix. LDA can be used for visualization, feature extraction, or classification.

da continuare

5 Modeling

5.1 Classic Machine Learning Methods

The traditional supervised machine learning techniques are methods used to train machine learning models from a set of labeled data, that is, a set of data for which both the input data and the corresponding output are known.

5.1.1 Linear Regression

Linear regression is a statistical method used to study the relationship between a dependent variable and one or more continuous independent variables. Its primary goal is to find the best-fitting line that represents this relationship between the variables.

In the specific case mentioned, the dependent variable is represented by the song's publishing year, while the independent variables are the song's audio features that can influence the prediction.

The linear regression model is defined by the following linear function:

[\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p]

Where:

(\hat{y}) represents the dependent variable or the variable to be predicted. (x) represents the independent variable or input variable. (\beta_0) is the intercept of the regression line. (\beta_1) is the slope of the regression line. I hope this information is helpful!

5.1.2 Random Forest Regressor

Random Forest is a supervised learning algorithm based on creating an ensemble of decision trees.

During training, multiple independent decision trees are generated. Each tree is built on a random subset of the training data, resulting in different models. Additionally, each tree uses only a random subset of input variables to make predictions.

Here are the hyperparameters and their corresponding values tested in this project. In section 6 **(add link)** we will specify the best configuration and the results obtained:

> Hyperparameters

5.1.3 K-Nearest Neighbors Regressor

The K-Nearest Neighbors (K-NN) algorithm searches for the k closest data points in the train set. The target is predicted by local interpolation of the targets associated of the nearest neighbors in the training set.

Here are the hyperparameters and their corresponding values tested in this project. In section 6 **(add link)** we will specify the best configuration and the results obtained:

> Hyperparameters

5.1.4 Support Vector Regressor

The Support Vector Regressor (SVR) algorithm that uses a machine learning algorithm based on Support Vector Machine to train a regression model. The Support Vector Regressor (SVR) algorithm aims to find a function (or hyperplane) that fits the data in such a way that the prediction error is minimized while allowing for some "acceptable" errors. To achieve this, SVR utilizes the concept of support vectors, which are the data points in the dataset closest to the separation hyperplane. The goal is to find a hyperplane that maximally separates these support vectors, ensuring the maximum distance between the hyperplane and the support vectors.

Mathematically, SVR seeks to minimize a cost function that measures the difference between the predicted values and the actual target values. The choice of the kernel function is crucial when using SVR, as it represents a mathematical function that measures the similarity between two points in the dataset

Here are the hyperparameters and their corresponding values tested in this project. In section 6 **(add link)** we will specify the best configuration and the results obtained:

> Hyperparameters

5.2 Deep Neural Networks

Neural networks are a type of machine learning algorithm inspired by the functioning of the human brain. A neural network consists of a set of artificial neurons that process input information, transmit it to other neurons in the network, and produce an output based on the processed data. During training, the neural network adapts to the training data by optimizing the weights of connections between neurons, aiming to minimize prediction errors.

The neural network used in this project was built using the PyTorch framework. **20% of the train set is used as a validation set** to evaluate the performance of the model on unseen data during training and to choose the model parameters in order to optimize the performance on the test dataset.

In this project we used a feed-forward neural network, which operates with a unidirectional flow, where each input layer and its corresponding output layer move in a single direction—there are no cycles or backward connections. This network consists of one or more layers of neurons, where each neuron is connected to all neurons in the subsequent layer. The input layer neurons receive input, process information, and pass it to the next layer.

During this processing, a linear combination of weighted inputs occurs, followed by the application of a non-linear activation function. The choice of activation function is crucial because it introduces non-linearity into the network, allowing for modeling more complex problems.

Typically, feedforward networks are trained using the gradient descent algorithm, which updates neuron weights to minimize the error between the expected output and the actual output of the network. The goal of the gradient descent algorithm is to find the variable values (weights) that minimize the loss function.

In summary, feedforward neural networks are powerful tools for approximating complex functions and learning from data. They play a significant role in various machine learning tasks, including regression, classification, and pattern recognition.

> Immagine esempio di Feed Forward Network

The network used is a feed-forward network model with a variable number of layers. Each layer consists of a succession of **Linear and ReLU (aggiungi dropout se rick lo mette)**.

Below is the Python code of the model:

> codice

Here are the hyperparameters and their corresponding values tested in this project. In section 6 **(add link)** we will specify the best configuration and the results obtained:

Da modificare con quelli messi da rick e riordinare nel modo in cui li ha messi lui

- hidden_size: indicates the size (number of neurons) in the hidden layers → [256, 512, 1024]
- epochs: refers to the number of training iterations →
- depth: specifies the number of hidden layers in the neural network → [3,4,5]
- dropout: represents the probability that some neurons are deactivated during network training →
- batch_size: defines the size of input data processed together during each training iteration → [8,16,32]
- learning_rate: determines the step size during gradient descent optimization and helps the model move toward a minimum of the loss function → [0.1, 0.01]
- step_size_lr_decay: **spiegazione** → [10, 20]

To find the optimal combination of the model's hyperparameters, the finetuning technique was used, that is, the iteration through the different combinations of hyperparameters to train the model, followed by the evaluation of the performance obtained in order to find the combination of hyperparameters that maximizes the performance. Below is the pseudocode of the model training:

> codice train con for

To avoid training the model beyond the necessary, the early stopping technique was used, which allows to stop the training of the model before the end of the fixed epochs, if no improvement is observed in the validation loss. In case the validation loss does not improve for a predetermined number of epochs, called patience, the training is interrupted and the best model reached during the training is returned. By doing so, the model is not trained beyond the necessary and the best possible model in terms of performance on the validation set is guaranteed. In our case, the maximum patience value was set to 10.

5.3 Tabular Data

PyTorch Tabular is a powerful library that aims to simplify and popularize the application of deep learning techniques to tabular data. The models of this module are capable of handling large amounts of data and finding complex relationships between variables, improving the model's predictive ability. In this project, we were required to use TabNet and TabTransformer, which are two different models for tabular data processing in this library.

5.3.1 Data Config

PyTorch Tabular uses Pandas Dataframes as the container which holds data and accepts dataframes as is, so there is no need to split the data into X and y like in Sci-kit Learn. Pytorch Tabular handles this using a DataConfig object, where you can specify this parameters:

- target: a list of strings with the names of the target column(s)
- continuous_cols: column names of the numeric fields
- categorical_cols: column names of the categorical fields to treat differently → Not used in this project since the dataset doesn't contain categorical columns

5.3.2 Trainer Config

Training a Deep Learning model can get arbritarily complex. PyTorch Tabular, by inheriting PyTorch Lightning, offloads the whole workload onto the underlying PyTorch Lightning Framework. It has been made to make training your models easy as a breeze and at the same time give you the flexibility to make the training process your own.

Deep learning model training can become quite intricate; for this reason PyTorch Tabular, built on the foundation of PyTorch Lightning, simplifies the entire process by leveraging the underlying PyTorch Lightning Framework. It provides an effortless way to train models while allowing users the flexibility to customize the training process according to their needs.

Here are the hyperparameters and their corresponding values tested in this project.

- max_epochs: maximum number of epochs to be run -- TabNet → [50, 150] -- TabTransformer → [50, 150]
- batch_size: number of samples in each batch of training → [512, 1024] -- TabNet → [512, 1024] -- TabTransformer → [512, 1024]
- early_stopping_mode: the direction in which the loss/metric should be optimized → [min]
- early_stopping_patience: The number of epochs to wait until there is no further improvements in loss/metric → [10]
- auto_lr_find: Runs a learning rate finder algorithm when calling trainer.tune(), to find optimal initial learning rate → [True]

5.3.3 Optimizer config

The optimizer plays a central role in the gradient descent process and is a critical component for training effective models. By default, PyTorch Tabular utilizes the Adam optimizer with a learning rate of 1e-3.

Here are the hyperparameters and their corresponding values tested in this project

- optimizer: the network optimizer → [Adam]
- lr_scheduler: the name of the LearningRateScheduler to use → []
- lr_scheduler_params: Dict: the parameters for the LearningRateScheduler → []

5.3.4 Model Config

5.3.4.1 TabNet

TabNet is a deep learning algorithm to process tabular data and it was developed by Google AI in 2019. TabNet takes raw data as input and uses sequential attention to select features to use at each decision step. This allows for better interpretation and learning, making the algorithm highly effective in solving classification and regression problems on tabular data. It employs multiple decision blocks that focus on processing a subset of input features. It employs unsupervised pre-training, initializing the model weights using knowledge from pre-trained models on large datasets or similar problems. By applying this technique to tabular data, it has demonstrated performance improvements. The uniqueness of TabNet lies in its composition: instead of using a single neural network, the model consists of multiple cascading neural networks, each selectively processing data and extracting increasingly relevant and specific information. Additionally, TabNet utilizes a masking mechanism, preventing the neural network from repeatedly using the same features, thereby ensuring better model generalization. The TabNet architecture is based on a set of cascading neural networks that employ selective attention to choose the most relevant features from the input data. Specifically, the TabNet architecture comprises three main elements: Encoder, Decoder, and Masking. The encoder consists of several decision units (Decision Points, DPs) that selectively process input data and produce a series of binary masks indicating which features are selected and which are discarded. The decoder, on the other hand, is a feedforward neural network that uses the masks generated by the encoder for final prediction. Lastly, masking prevents the neural network from reusing the same features multiple times during data processing, leading to better model generalization and aiding in preventing overfitting

> aggiungere le varie foto tra un paragrafo e l'altro

> codice

Here are the hyperparameters and their corresponding values tested in this project. In section 6 **(add link)** we will specify the best configuration and the results obtained:

Da riordinare

- learning_rate: the learning rate of the model →[0.001, 0.005]
- virtual_batch_size: batch size for Ghost Batch Normalization → [64, 128]
- n_indipendent: the number of independent features to use in the transformer network → [2, 3]
- n_shared: the number of shared features to use in the transformer network → [2, 3]
- n_step: number of successive steps in the network → [3, 5]
- gamma: scaling factor for attention updates → [1.3, 1.5]
- seed: the random seed to use for reproducibility → [42]
- n_a: dimension of the attention layer → [8, 16, 32, 64]

- n_d: dimension of the prediction layer → [8, 16, 32, 64]

5.3.2 TabTransformer

TabTransformer is a transformer-based model designed for supervised learning that uses self-attention to capture the dependencies between features and is known for its ability to handle sequential data. It builds upon the foundation of self-attention based Transformers. The Transformer layers play a crucial role in transforming the embeddings of categorical features into robust contextual embeddings. These embeddings enhance prediction accuracy by capturing relevant information from the input data. They are also highly robust against both missing and noisy data features, providing better interpretability. TabTransformer performs well in machine learning competitions and can be used for regression, classification (both binary and multiclass), and ranking problems.

Here are the hyperparameters and their corresponding values tested in this project. In section 6 **(add link)** we will specify the best configuration and the results obtained:

- virtual_batch_size: batch size for Ghost Batch Normalization → [64, 128]
- learning_rate: the learning rate of the model → [0.001, 0.005]

6 Performance Evaluation

6.1 Classic Machine Learning

6.1.1 Linear Regression

6.1.2 Random Forest Regressor

6.1.3 K-Nearest Neighbors Regressor

6.1.4 Support Vector Regressor

6.2 Neural Networks

6.3 Tabular Data

6.3.1 TabNet

6.3.2 TabTransformer