iscte TECNOLOGIAS E ARQUITETURA

Algoritmos para Big Data
Lab class handout #3

Feature extraction and transformation **2024/2025**

Goals

This lab class will focus on feature extractors and transformers, which are critical components in the field of machine learning and also for data preprocessing.

Note: It is acceptable that the work to be undertaken may require some time beyond the class. If that is the case, work should continue outside of class. And if needed, assistance will be provided during the office hours established or to be agreed with the lecturer.

Supporting information

- Course slides
- Machine Learning Library (MLlib) Guide
- Plotly Open Source Graphing Library for Python

Introduction

Among other aspects, an important usage of feature extractors and transformers relates to converting raw data into more suitable formats to train machine learning models. While feature extractors move data towards a lower-dimensional space, e.g. reducing the number of variables to those most meaningful to the tasks ahead, the feature transformers look primarily at modifying such features into more suitable formats.

Hereafter, we will focus on three types of operators regarding features: extractors, transformers and selectors. The Apache Spark's MLib library provides various algorithms to support those operations.

Regarding datasets to be used in this lab class, it is needed a data file located at:

`https://bigdata.iscte-iul.eu/datasets/iot-devices.csv`

Finally, alongside this handout describing the tasks to accomplish, a notebook is also provided, that it is expected to be completed with the implementation of the tasks.

Tasks to accomplish

A. Data ingestion and preparation

In this initial task, do execute the following operations:

1. Download the data file already mentioned and save it in a proper directory, within the working directory of your Pyspark instalation.

2. After creating a *SparkSession* to work with, read the data into a DataFrame and then check it. Make sure the data is prepared/cleaned to be worked with. Let us consider the name of such DataFrame as `df_iot`.

3. Create a new DataFrame with columns of interest, to be called `df_devices` and after `df_iot`. The columns in `df_devices` will be:

   - `device_id`
   - `device`, based on `device_name` but replacing the minus char by the space char
   - `device_words`, based on `device` but as an array of the words
   - `battery_level`
   - `c02_level`
   - `humidity`
   - `temp`
   - `cn`

   Also, create two arrays with names of numeric and non-numeric columns, as follows:

   - `input_cols_num`, containing `battery_level`, `c02_level`, `humidity` and `temp`
   - `input_cols_str`, containing `cn`

   These structures will be used in the coming tasks.

4. With the previous `df_devices`, `input_cols_num` and `input_cols_str`, do plot a few graphs to grasp the kind of data we are working with. For example, plot histograms using *plotly.express*.

B. Basic statistics

In this task we will look at useful statistical operators to get basic statistics regarding the raw data of `df_devices`, previously set. Execute the following operations:

1. Plot a graph with the pairwise correlations among numeric columns named in `input_cols_num`, which was set in the initial task. For the purpose, consider the following:

   (a) First, create a DataFrame, to be called `df_features`, with just a single column containing a vector of the assembled numeric data of concern. This is a common requirement for machine learning algorithms in Spark MLib. To do so, use *VectorAssembler* from Spark's MLlib, which is considered as a feature transformer.

   (b) Then use *Correlation* from Spark's MLlib upon the previous `df_features`.

   (c) In the end, draw a heatmap using *plotly.express*.

   Also, to better understand *VectorAssembler*, compare row-wise `df_devices` and `df_features`.

2. Compute summary statistics for the previous `df_features` using *Summarizer* from Spark's MLlib. Consider the following column-wise metrics: *min*, *max*, *mean*, *sum*, *count*, *variance*, *std*, *normL1* and *normL2*. Notice that you can get all at once or a particular one independently.

C. Feature extraction

In this task, we are going to extract features from the raw data of `df_devices`, previously set. The techniques below are proven to be useful in some machine learning use cases.

Unless stated otherwise, the algorithms mentioned below are part of the Spark's MLib library.

Execute the following operations:

1. Apply the *TF-IDF* [1] technique to the column `device`, in order to capture the importance of each term in a row relatively to all rows. Notice that unique terms specifically to one row (rare across all rows) will receive higher TF-IDF scores. The process works as follows:

[1] Term Frequency-Inverse Document Frequency (TF-IDF), commonly used in text mining.

(a) Starting with the already tokenized column `device_words`, apply *HashingTF* from Spark's MLlib to hash the words into feature vectors.

(b) Then use *HashingTF* from Spark's MLlib to rescale the feature vectors.

(c) Last, show the relevant columns: `device_id`, `device_words` and the column with the TF-IDF features.

2. Use the estimator *Word2Vec* to map words in column `device_words` into high-dimensional vectors of size 4. Show the outcome.

   As a note, this technique is particularly useful in natural language processing where words with similar meanings will have similar representations. (Average of all words are taken into account).

3. Use the estimator *CountVectorizer* to map words in column `device_words` into vectors of word counts. Use `vocabSize` as 3 (top 3 words). Show the outcome.

4. Use *FeatureHasher* to project the columns that were set in the arrays in the initial task, `input_cols_num` alongside `input_cols_str`, into feature vectors of dimension 4 (as always, it should lead to a lower space: $5 \rightarrow 4$). Show the outcome.

   Notice that operations dealing with both numeric and categorical values are frequent in problems of classification and clustering.

D. Feature transformation

In this task, the focus is on techniques to modify raw data into a more suitable format, that is, more refined and structured so later on they can be used effectively by algorithms.

As in the previous task, the raw data under analysis is from `df_devices` (recall initial task) and unless stated otherwise, the algorithms mentioned below are part of the Spark's MLib library.

Hence, execute the following operations:

1. Use *Tokenizer* to show a column containing the text tokens that exist in the column `device`. It is a so-called bag of words. Then figure out how such outcome compares with the column `device_words` (recall the related computation in the initial task).

2. Use *StringIndexer* to map the column `cn` (a string field) into a column of numerical indices. Show the outcome.

   Recall that the majority of machine learning algorithms cannot work directly with string data.

3. Use *OneHotEncoder* to map the columns `battery_level` and `c02_level` (category indices) into a binary format vector. Show the outcome.

   This kind of operation is particularly useful in cases of further transforming categorical variables after using the above *StringIndexer*.

4. Use *Binarizer* to map the column `battery_level` (a continuous feature) into a binary (0/1) feature, according to a threshold of 5.0. Show the outcome.

5. Use *Bucketizer* to map the column `battery_level` (a continuous feature) into a column of feature buckets, based on the split $[0.0, 2.0, 4.0]$ so leading to three buckets. Show the outcome.

   In general, this type of operation allows to convert continuous variables into categorical ones.

6. Use *StandardScaler* to scale the columns set in `input_cols_num` (recall initial task), by removing the mean and scaling to unit variance. Show the outcome.

   Worth remembering that many algorithms assume input features as being on similar scales.

7. Use *MinMaxScaler* to scale the columns set in `input_cols_num` (recall initial task), to the range $[0, 1]$. Show the outcome.

   This operation is particularly useful when there is a need to normalize the features to a specific range, without distorting differences in the ranges of values. Indeed, there are algorithms very sensitive to feature scaling e.g. those related to gradient-based optimizations.

8. Use *PCA* to reduce the dimensionality of the data in the columns set in `input_cols_num` (recall initial task) to three components, while preserving its essential information. Show the outcome.

   In the end, this technique identifies the most important features (principal components) that explain the maximum variance of data, and algorithms that use it can be more efficient.

E. Feature selection

In this task, the focus is on selecting a relevant subset of features from a larger set of features. This is particularly useful when we want to select a subset automatically, that can be used in algorithms later on. For example, in the context of feature engineering to build machine learning models.

As in the previous task, the raw data under analysis is from `df_devices` and the features to work with relate to the columns set in `input_cols_num` (recall initial task). Also, unless stated otherwise, the algorithms mentioned below are part of the Spark's MLib library.

Hence, execute the following operations:

1. Create a new DataFrame after `df_devices`, by adding a new column to be called `danger`, such as its value will be 1.0 if `(battery_level < 2 | c02_level > 1000 | temp > 26)`; 0.0 otherwise.

2. Use *ChiSqSelector* to select the two most important features under analysis, against the label column `danger`, set before. The computation is therefore based on the chi-squared statistical test (measure of independence among variables).

3. Use *VectorSlicer* to select the subset of features related to the columns `humidity` and `temp`.