AMD - SM2L JOINT PROJECT

Davide Gavio
930569

Date: October 27, 2020

Abstract

This is the report where I analyze the joint project for "Algorithms for massive datasets" and "Statistical methods for machine learning" courses. The project is based on the analysis of the "2013 American Community Survey". The objective is to implement a learning algorithm for regression with square loss in order to predict a label chosen among five.

1 Problem definition and background

1.1 Introduction

As said in the abstract, the goal is to write from scratch an algorithm for regression with square loss in order to predict a label chose among five.

1.2 Mathematical prelimaries

Regression Regression models are used to predict a continuous value. Predicting prices of a house given the features of house like size, price etc is one of the common examples of Regression. It is a supervised technique. Dave (2018)

Linear regression Simple linear regression is a type of regression analysis where the number of independent variables is one and there is a linear relationship between the independent(x) and dependent(y) variable. Based on the given data points, we try to plot a line that models the points the best. The line can be modelled based on the linear equation shown below.

$$Y = a_0 + a_1 * x$$

The objective of the linear regression algorithm is to find the best values for a_0 and a_1. The cost function helps us to figure out the best possible values for a_0 and a_1 which would provide the best fit line for the data points. Since we want the best values for a_0 and a_1, we convert this search problem into a minimization problem where we would like to minimize the error between the predicted value and the actual value. We square the error difference and sum over all data points and divide that value by the total number of data points. This provides the average squared error over all the data points. Therefore, this cost function is also known as the Mean Squared Error(MSE) function which will be used to adjust the values of a_0 and a_1. Gradient descent is a method of updating a_0 and a_1 to reduce the cost function(MSE). The idea is that we start with some values for a_0 and a_1 and then we change these values iteratively to reduce the cost. Gandhi (2018)

Ridge regression Ridge regression is a regularized version of Linear regression: adding a regularization factor to the cost function so the algorithm is forced to keep weights as lower as possible. The regularization term is usually called *alpha*. Ridge regression is used to avoid overfitting when linear regression is applied on a dataset with lots of features. The penalty factor is added to the cost function so the values of the features will be less important varying on the value of the regularization factor. There are two types of regularization: ridge regression uses the second type, which produces small coefficients without nullifying them. Regularization factor is an hyperparameter that needs to be tuned.
The mathematical formula behind ridge regression is the following:

$$w^\star = \underset{w \in \Re^d}{argmin}(||X \cdot w - y||^2 + \lambda ||w||^2)$$

where $\lambda||w||^2$ is the regularization term. λ is the regularization factor to be tuned. Provino (2019)

1.3 Dataset

The American Community Survey is an ongoing survey from the US Census Bureau. In this survey, approximately 3.5 million households per year are asked detailed questions about who they are and how they live. Many topics are covered, including ancestry, education, work, transportation, internet use, and residency. There are two types of survey data provided, housing and population. For the housing data, each row is a housing unit, and the characteristics are properties like rented vs. owned, age of home, etc. For the population data, each row is a person and the characteristics are properties like age, gender, whether they work, method/length of commute, etc. Each data set is divided in two pieces, "a" and "b" (where "a" contains states 1 to 25 and "b" contains states 26 to 50).

File	Focus	Rows	Columns	Size
ss13husa.csv	House	756065	231	601569 KB
ss13husb.csv	House	720248	231	572853 KB
Total	House	1476313	231	1174422 KB
ss13pusa.csv	People	1613672	283	1484981 KB
ss13pusb.csv	People	1519123	283	1397516 KB
Total	People	3132795	231	2812497 KB

Some statistical results from data exploration:

```
+-------+-----------------+
|summary|             WAGP|
+-------+-----------------+
|  count|          2582042|
|   mean|25652.55809239354|
| stddev|47360.01051271143|
|    min|                0|
|    max|           660000|
+-------+-----------------+
```

2 Existing solutions

Writing an algorithm from scratch is a complex operation. For instance the Ridge Regression is already implemented inside Scikit-learn package.

Scikit-learn In order to apply Scikit-learn ridge regression on real life dataset it is necessary to preprocess and clean it before doing everything else. Once done it's easy to use it, like is noticeable from the picture below:

```
from sklearn.linear_model import Ridge
import numpy as np

n_samples, n_features = 10, 5
rng = np.random.RandomState(0)
y = rng.randn(n_samples)
X = rng.randn(n_samples, n_features)
ridge = Ridge(alpha=1.0)
ridge.fit(X, y)
```

The above script is taken from Sklearn ridge regression page and is performed using random numbers in the dataset. After calling *fit* method is possible to predict the target label or just take a look at the parameters.

3 Project design and implementation

3.1 Preprocessing

In order to use the data contained in the dataset it is necessary to preprocess and clean it. The operations I have performed to do so are: missing values management, outliers removal, value scaling, feature selection.

Missing values management In order to execute the implemented regression algorithm is mandatory to have only numerical values. In the case of the dataset used there aren't categorical values, so this is almost not a problem. There are still a certain quantity of missing data, identified with *null* values. Given that it's not possibile to drop all the row and all the columns containing such values, some work on them is necessary. First of all we have removed all constant values features (column RT) becose they don't carry out any information. Then we have removed two columns specified in the project assignment, given the label to predict (removed HINCP and FINCP). After that, all rows and columns containing more than 66% of missing values are dropped. This happens because it's useless to infer rows or columns with too many missing values, it will result in completely made up records. If there is a missing or null value in label, the corresponding row will be removed. This has to be done because training using data not corresponding to a label is not useful. Finally, for every column the number of null values gets counted and then if it's over the forementioned thresh the column gets removed. After those three operations, it's necessary to fill the remaining missing values. In order to do so it's necessary to infer some value where the value is missing. To do so it's possible to choose an arbitrary value (maybe 0 or -1), but these could lead to some distortion in the informative content and in the predicted values. Another possible way is to use the mean or the median value of the column. There is a built in method to impute the requested value in corresponding null values. For each column is calculated the mean and imputed in the right positions:

Outliers removal An outlier is an observation that is unlike the other observations. It is rare, or distinct, or does not fit in some way. Using a real world dataset like American Community Survey, it's normal to have outliers due to the human factor. Only a domain expert is able to define and locate outliers in a dataset. Nevertheless, we can use statistical methods to identify observations that appear to be rare or unlikely given the available data. A good statistic for summarizing a distribution sample of data is the Interquartile Range. The IQR is calculated as the difference between the 75th and the 25th percentiles of the data. Brownlee (2018)

Vector assembling The first thing to do is to use the VectorAssembler to combine a given list of columns into a single vector column. This is mandatory in order to apply other transformers in the transformation pipeline. To do so it's necessary to instantiate a VectorAssembler object like in the picture below:

```
features_assembler = VectorAssembler(inputCols=feature_columns ,\\
    outputCol='features')
  target_assembler = VectorAssembler(inputCols=feature_columns,\\
    outputCol='target')
```

After performing those two transformations the dataframe will have two more columns, the ones specified as *outputCol* in the two lines of code. Those two columns will be the two which will be used in further transformations.

Feature selection Feature selection is the automatic selection of attributes in your data (such as columns in tabular data) that are most relevant to the predictive modeling problem you are working on. It helps you by choosing features that will give you as good or better accuracy whilst requiring less data. Fewer attributes is desirable because it reduces the complexity of the model, and a simpler model is simpler to understand and explain Brownlee (2014).
The technique used in the project is a chi-square selection, which calculates how strong is the dependence relationship between the input variable and the predicted one. In order to do so a PySpark transformer called *ChiSqSelector* comes in hand. The *numTopFeatures* parameter is used to decide how many features are kept at the end of the execution.
ChiSqSelector is unable to work with more than 10000 distinct values, so in order to perform an experiment with the entire dataset this cannot be used.

Pipeline In this project a pipeline is used to perform a sequence of transformations on the dataframe. the specific order of execution is:

- *features_assembler*
- *target_assembler*
- *feature_selector*

and it's visible in the piece of code below:

```
  df = Pipeline(stages=[features_assembler, target_assembler, feature_selector]).
      fit(df).transform(df)
```

After the execution of the pipeline there will be the two columns that will be used for the next workflow steps:

```
  +--------------------+---------+
  |   selected_features|   target|
  +--------------------+---------+
  |(50,[0,1,2,3,4,5,...|    [0.0]|
  |(50,[0,1,2,3,4,5,...|[27500.0]|
  +--------------------+---------+
```

The first column contains the selected features, assembled in a vector. The second column contains the target column.

Value scaling StandardScaler transforms a dataset of Vector rows, normalizing each feature to have unit standard deviation and/or zero mean. In order to scale correctly my dataset, I have split the dataset into *training set*, *validation set* and *test set* with ratio [.6, .2, .2]. The formula behind the normalization is the following: $z = \frac{x-\mu}{\sigma}$ where μ is the original mean of the feature and σ is the original standard deviaton. After splitting the dataset I have trained the scaler on the training set and then I trainsformed the three sets using the model trained on the first one. Below is shown an example of two scaled rows of the training set:

```
+--------------------+--------------------+
|     scaled_features|       scaled_target|
+--------------------+--------------------+
|[0.30824462897154...|[-0.7074913402612...|
|[-0.7161957269573...|[-0.7074913402612...|
+--------------------+--------------------+
```

3.2 Hyperparameter tuning

In order to tune the hyperparameters of the algorithm (learning rate and regularization factor) the idea is to use the grid search. Grid search is a method that creates a model for every parameter (or combination of parameters) given as input. After performing grid search for each candidate parameter candidate (or combination of them), the hyperparameters that give the best results in the grid search are used in the general algorithm. The best regularization factor is chosen looking at the smallest value of the Root Mean Squared Error (RMSE). For each candidate paramater there is the use of the gradient descent technique, so it is pretty long to execute.

3.3 Algorithm implementation

Some ideas were taken from Erik Linder-Norén machine learning project.
In order to implement the Ridge Regression from scratch, I decided to structure it as a class which is possibile to instantiate. Doing so allows to use the instantiated variables referring to and in the scope of the instantiated object. All the instantiated variables are used during the gradient descent. After initializing all the necessaries variables, the fit method is called. At each iteration of the gradient descent, some quantities are updated. The variable *n_iterations* is the number of training iterations the algorithm will tune the weights for. The variable *learning_rate* is the step length that will be used when updating the weights. After the fitting operation the prediction method is called. For every row of the test set dataframe composed by [label, features] is returned a new tuple composed by te label and the dot product of the trained weights before multiplied by the features.

Evaluation In order to evaluate the results produced by the algorithm the metrics that I used are:

- **MSE**: it is the average of the squared differences between the target label and and the label predicted by the regression algorithm. Squaring the differences, it penalizes even small errors. The mathematical formula is: $MSE = \frac{\sum(y-\hat{y})^2}{n}$
- **RMSE**: it is used widely and mostly when large errors are undesired, due to its peculiarity to square errors before averaging that penalizes large errors. Mathematically: $RMSE = \sqrt{\frac{\sum(Predicted_i - Actual_i)^2}{n}}$
- **MAE**: it measures the absolute difference between the target label and e label predicted by the algorithm. It is more robust to outliers and is less penalizing than MSE. $MAE = \frac{\sum|y-\hat{y}|}{n}$
- **R2**: it's the coefficient of determination. Compares the model with a constant baseline chosen by taking the mean of the data and drawing a line at the mean. It is scale free, so this metrics will always be less or equal to 1. The higher the value of R2, the better it is. The formula is: $R^2 = 1 - \frac{MSE(model)}{MSE(baseline)}$

?

3.4 PySpark

In order handle the possibile the increasing size of the datasets in real world it is necessary use technologies developed for that task. The tool I decided to use is called PySpark. PySpark is the python library for Apache Spark. Apache Spark is a unified analytics engine for large-scale data processing Apache (2009a). It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs.

If the dataset is small enough to be stored in main memory, the common practice is to preprocess and clean it in form of a Pandas Dataframe while machine learning algorithms are applied on the final dataframe version converted to NumPy Arrays. With big data this is impossible because it's impossible to predict how big will become the dataset. Working with Big Data it is advisable to exploit the multi-core capabilities of computing engines (Amazon AWS, Google Cloud Platform, Microsoft Azure) while working on smaller portion of data in order to not saturate the main memory. The two technologies used in this project are: PySpark Dataframe, PySpark RDDs.

PySpark Dataframe A DataFrame is a Dataset organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood. DataFrames can be constructed from a wide array of sources such as: structured data files, tables in Hive, external databases, or existing RDDs. The DataFrame API is available in Scala, Java, Python, and R. In Scala and Java, a DataFrame is represented by a Dataset of Rows. In the Scala API, DataFrame is simply a type alias of Dataset[Row]. While, in Java API, users need to use Dataset<Row> to represent a DataFrame. A Dataset is a distributed collection of data. Apache (2009b). Leveraging those peculiarities allows the developer to exploit all the scaling capabilities of a cluster.

PySpark RDD Spark revolves around the concept of a resilient distributed dataset (RDD), which is a fault-tolerant collection of elements that can be operated on in parallel. There are two ways to create RDDs: parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat Apache (2009c). With the latest updates it is advisable to use Dataframe instead of RDDs.

4 Experiments

4.1 Machine setup

I have executed all the experiments on Google Colab, which is so composed:

- **OS name**: Linux
- **OS version**: 4.19.112
- **Architecture**: x86_64
- **CPU physical cores**: 1
- **CPU total cores**: 2
- **RAM amount**: 12.72 GB
- **Disk size**: 107.77 GB
- **Python version**: 3.6.9
- **Spark version**: 3.0.1
- **Hadoop version**: 2.7

Each one of the experiment is composed by the steps already discussed in a more theoretical way before, below there will be only the technical point of view. For both experiments the .csv files needed are downloaded from kaggle website.

4.2 Data exploration

4.3 Experiment #1

The first experiment is performed following exactly the steps seen in Section 3:

- **Predicted label**: WAGP
- **Dateset reading**:
  - *Rows read*: 70000 rows
  - *Execution time*: 1min 48s
- **Missing values management**:
  - *Drop threshold*: .66
  - *Imputing strategy*: mean
  - *Initial shape*: 70000 rows by 283 columns
  - *Final shape*: 58107 rows by 234 columns
  - *Execution time*: 2min 58s
- **Outliers removal**:
  - *Initial shape*: 58107 rows by 234 columns
  - *Final shape*: 50950 rows by 234 columns
  - *Execution time*: 57.2 s

- **Value imputing**:
  - *Imputing strategy*: mean
  - *Execution time*: 46.3 s
- **Pipeline**:
  - *Number of features*: 50
  - *Initial shape*: 50950 rows by 234 columns
  - *Final shape*: 50950 rows by 50 columns
  - *Execution time*: 1min 52s
- **Training, validation, test split**:
  - *Split weights*: .6, .2, .2
  - *Training set size*: 30536
  - *Validation set size*: 10135
  - *Test set size*: 10279
  - *Execution time*: 1min 32s
- **Grid search**:
  - *Alphas tested*: 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1, 10, 100, 1000
  - *Learning rate tested*: 0.00001
  - *Gradient descent iterations*: 150
  - *Execution time*: 38min 9s
- **Training**:
  - *Alpha*: 0.0001
  - *Learning rate*: 0.00001
  - *Gradient descent iterations*: 150
  - *Execution time*: 4min 10s
- **Predicting**:
  - *Execution time*: 74.6 ms
- **Training metrics**:
  - *MSE*: 0.37770147579468943
  - *RMSE*: 0.6145742231778758
  - *MAE*: 3.781215436239726e-18
  - *R2*: 0.6222861547448344
- **Testing metrics**:
  - *MSE*: 0.38531527115224384
  - *RMSE*: 0.6207376830451361
  - *MAE*: 0.007350844331173962

  - *R2*: 0.6163675467945217

4.4 Experiment #2

The second experiment is performed on the same portion of the dataset without outliers removal.

- **Predicted label**: WAGP
- **Dateset reading**:
  - *Rows read*: 70000 rows
  - *Execution time*: 2min 5s
- **Missing values management**:
  - *Drop threshold*: .66
  - *Imputing strategy*: mean
  - *Initial shape*: 70000 rows by 283 columns
  - *Final shape*: 58107 rows by 234 columns
  - *Execution time*: 3min 20s
- **Outliers removal**: not performed
- **Value imputing**:
  - *Imputing strategy*: mean
  - *Execution time*: 49.4 s
- **Pipeline**:
  - *Number of features*: 50
  - *Initial shape*: 58107 rows by 234 columns
  - *Final shape*: 58107 rows by 50 columns
  - *Execution time*: 2min 23s
- **Training, validation, test split**:
  - *Split weights*: .6, .2, .2
  - *Training set size*: 34900
  - *Validation set size*: 11553
  - *Test set size*: 11654
  - *Execution time*: 1min 44s
- **Grid search**:
  - *Alphas tested*: 1e-4, 1e-3, 1e-2, 1, 10
  - *Learning rate tested*: 0.00001
  - *Gradient descent iterations*: 300
  - *Execution time*: 42min 58s
- **Training**:

  - *Alpha*: 0.0001
  - *Learning rate*: 0.00001
  - *Gradient descent iterations*: 300
  - *Execution time*: 4min 44s
- **Predicting**:
  - *Execution time*: 73 ms
- **Training metrics**:
  - *MSE*: 3.008066504534152e+35
  - *RMSE*: 5.484584309256402e+17
  - *MAE*: 878.5861318051576
  - *R2*: -3.0081526980211743e+35
- **Testing metrics**:
  - *MSE*: 3.0131107153401775e+35
  - *RMSE*: 5.4891809182611e+17
  - *MAE*: 3456267346130641.5
  - *R2*: -3.027014612247103e+35

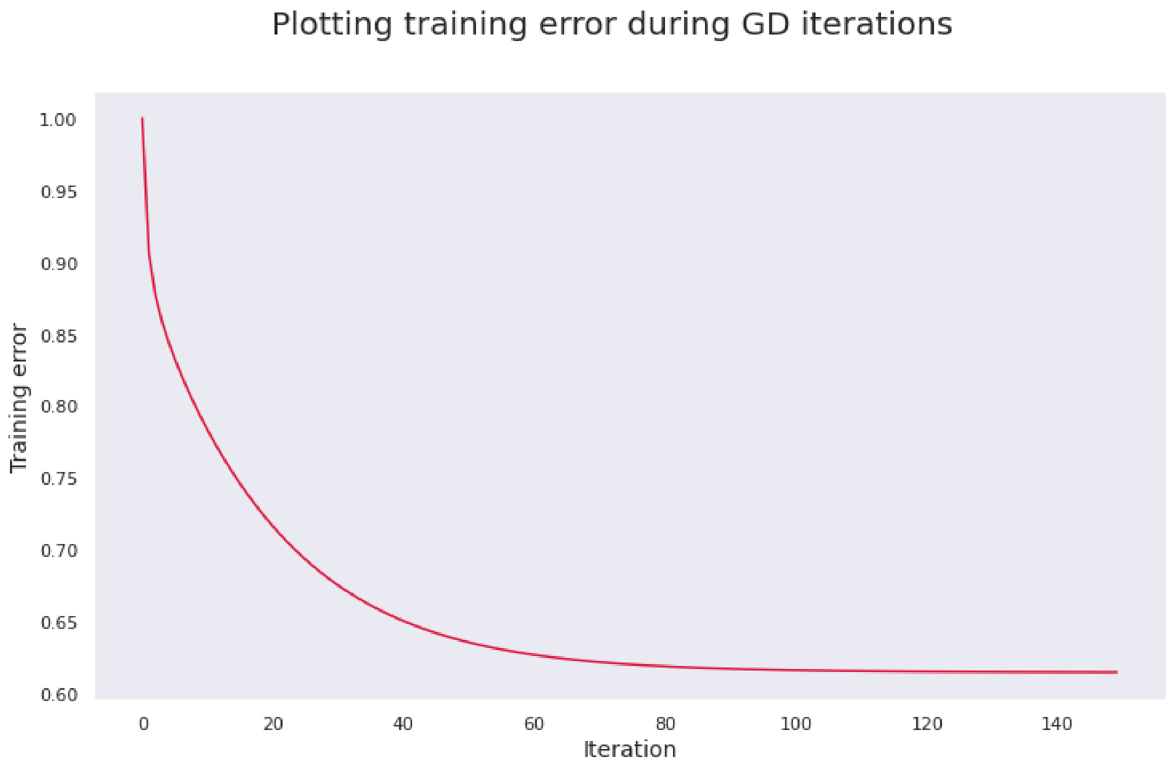
4.5 Future experiments

In order to have a more complete view over the problem discussed in the report, it's advisable to perform at least one experiment using the whole dataset. I couldn't perform it due to technological limitations, for example Google Colab prospects five days of training time which reported to the grid search it's a long time. The solution could be using some online computational engine, such as Google Cloud Platform, Microsoft Azure or Amazon AWS.
Some other experiments can be performed, like tuning some variables (drop threshold for example). Due to technological limits I couldn't perform all the possible experiments.

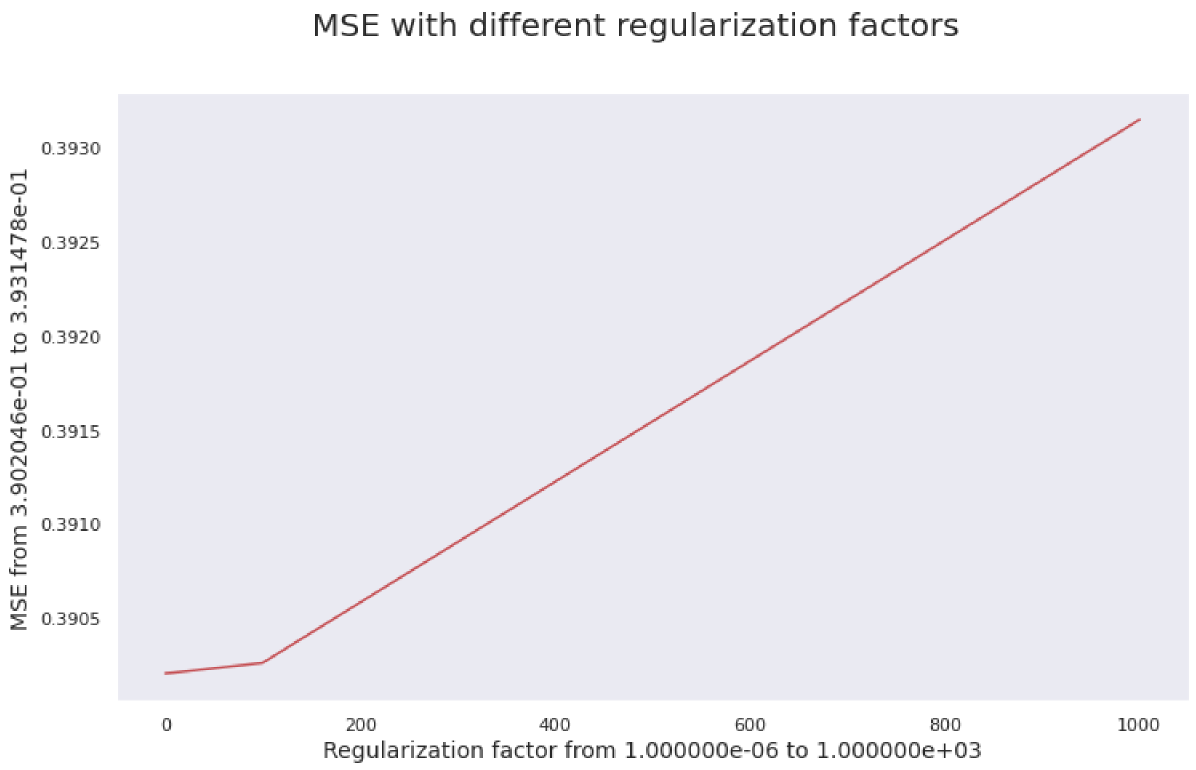
5 Results

5.1 Experiment #1

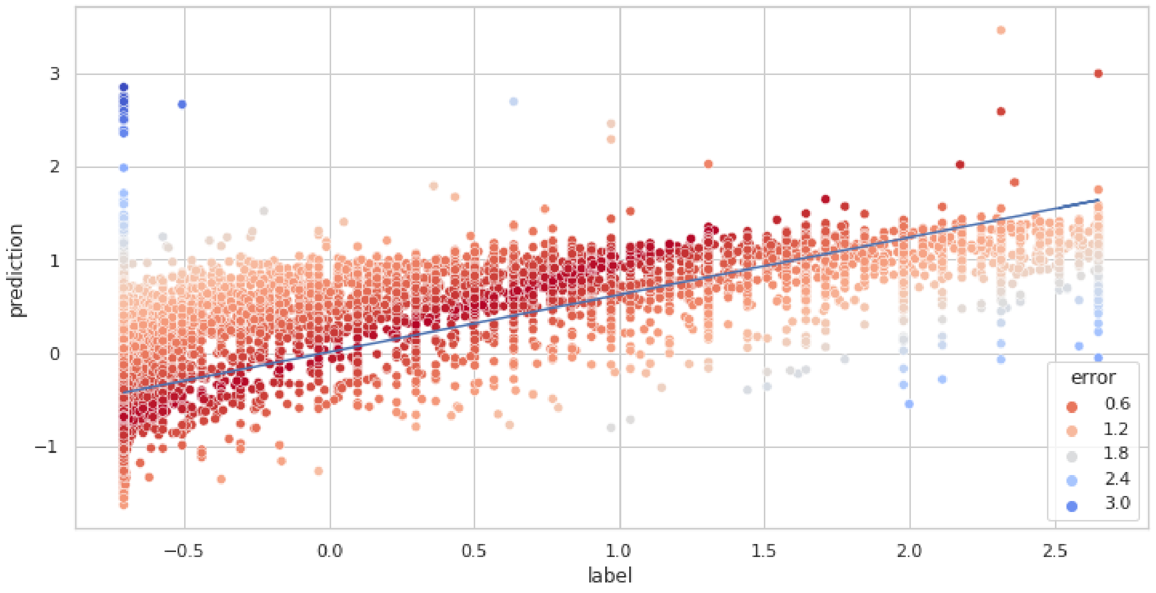
In this experiment the convergence of gradient descent is reached in more or less 150 iterations, as visible in the picture below:

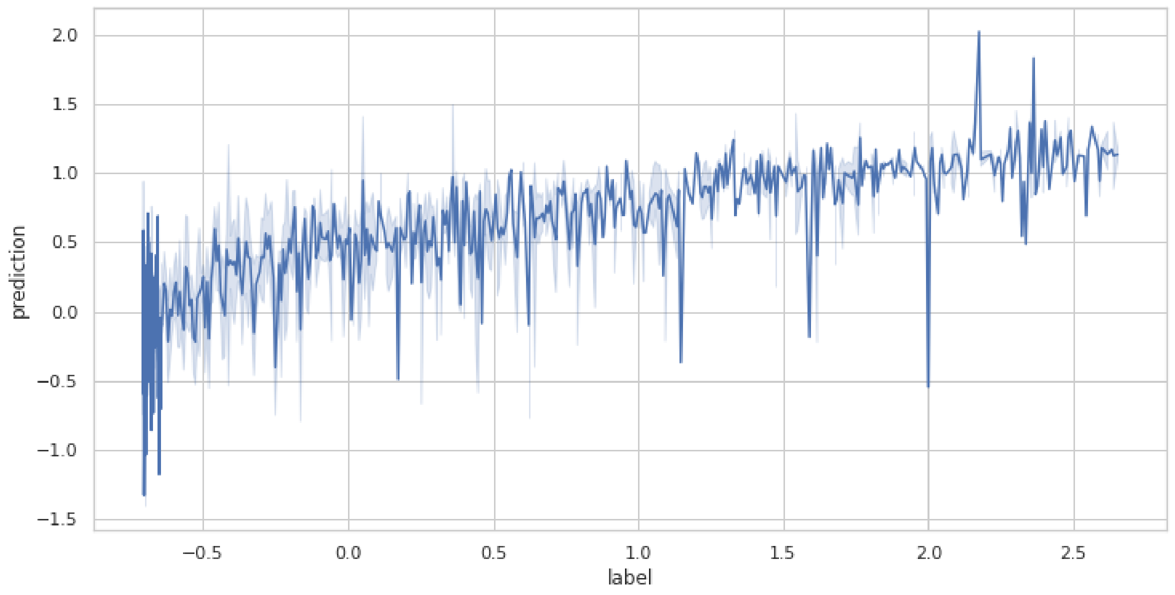


Due to this fast training time it’s possibile to perform the grid search in order to find the best regularization factor among a list of some. An example of grid search graph is visible in the picture below:



As noticeable from the pricture, the first element in the alphas list (0.0001) is the best, as the MSE calculated for each training model keeps growing. The metrics extrapolated from the predictions done by the algorithm show that the performances are quite good but that could be better. MSE and R2 metrics that should be respectively near 0 and near 1 demonstrate that not all the predictions are good. The two pictures below show this phenomena:

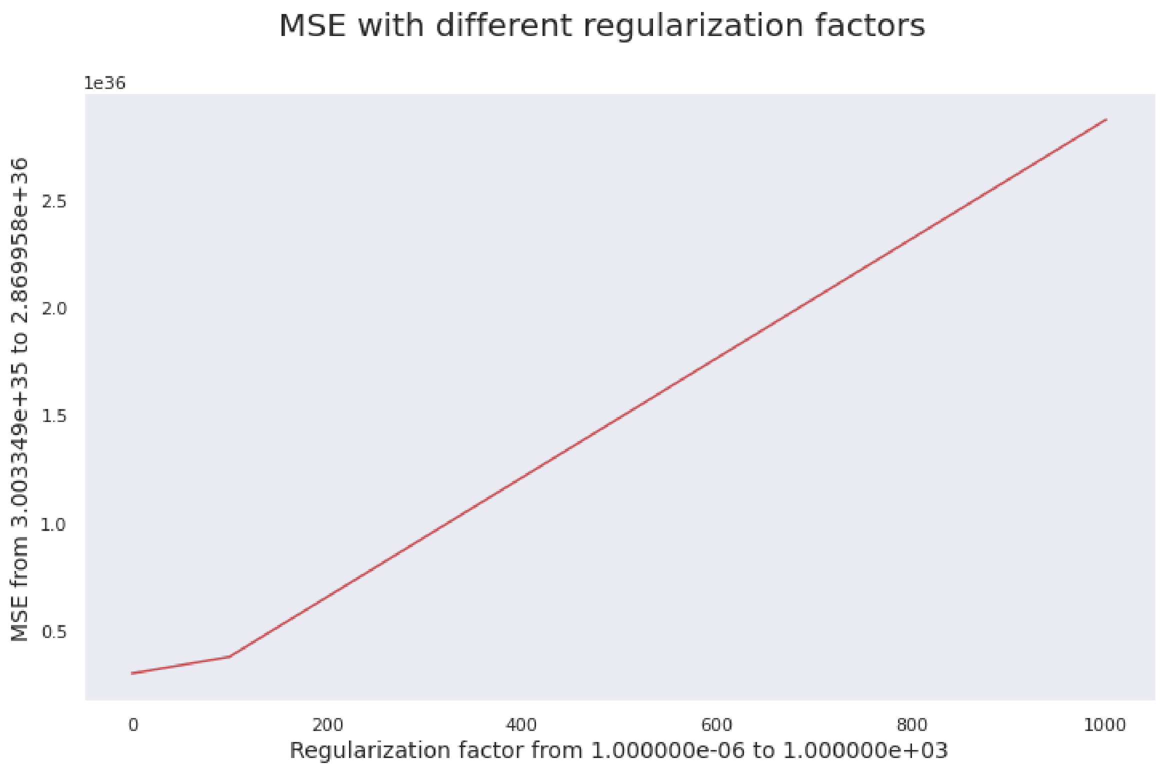




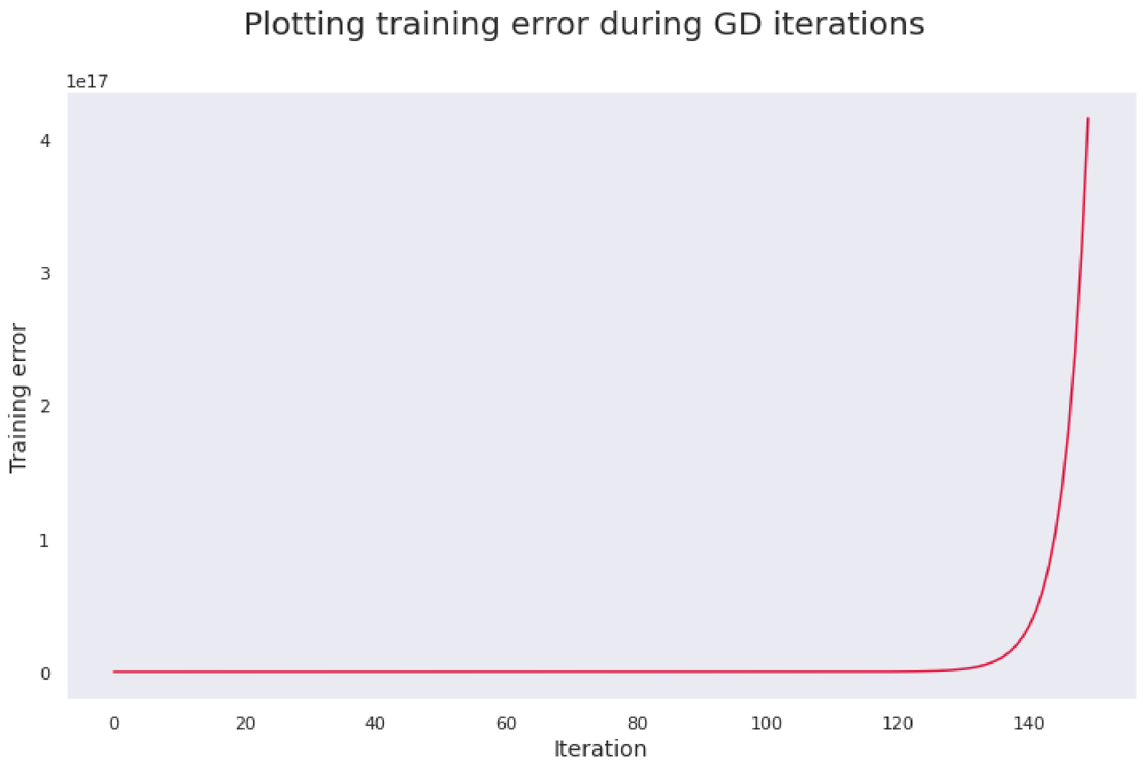
In the first picture is visible that a lot of points (they are all the predictions) are in the regression line area, while some are pretty distant from it, confirming that the model is good but not perfect. In the second picture is visible that the two lines follow more or less the same "route", confirming what emerges from the scatterplot and from the metrics

5.2 Experiment #2

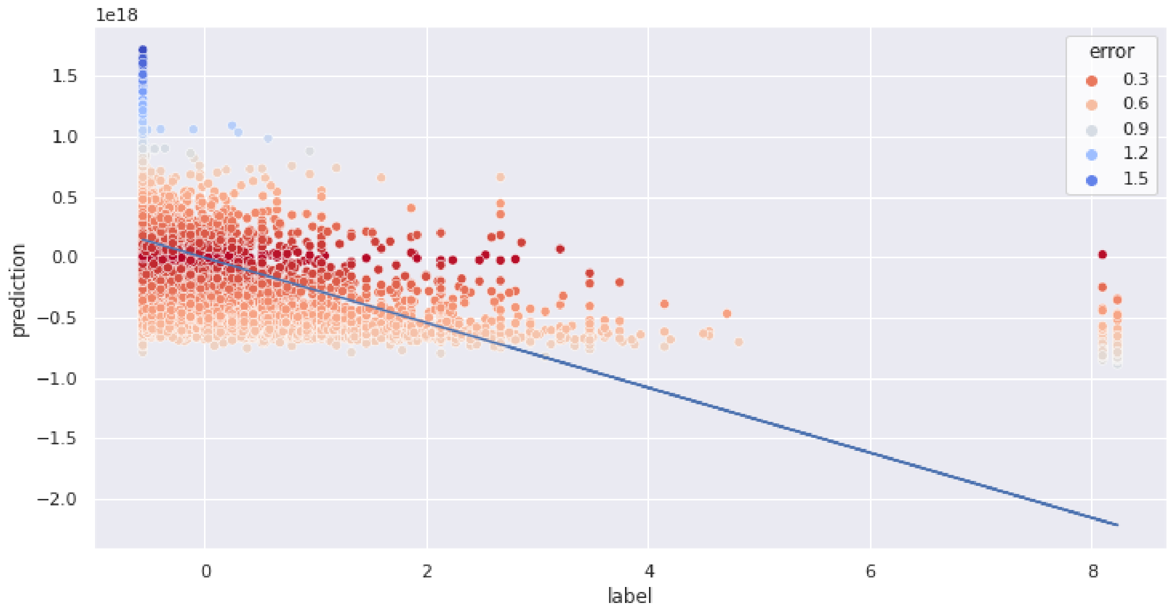
Like the first experiment, the grid search is pretty similar and so is his graph:

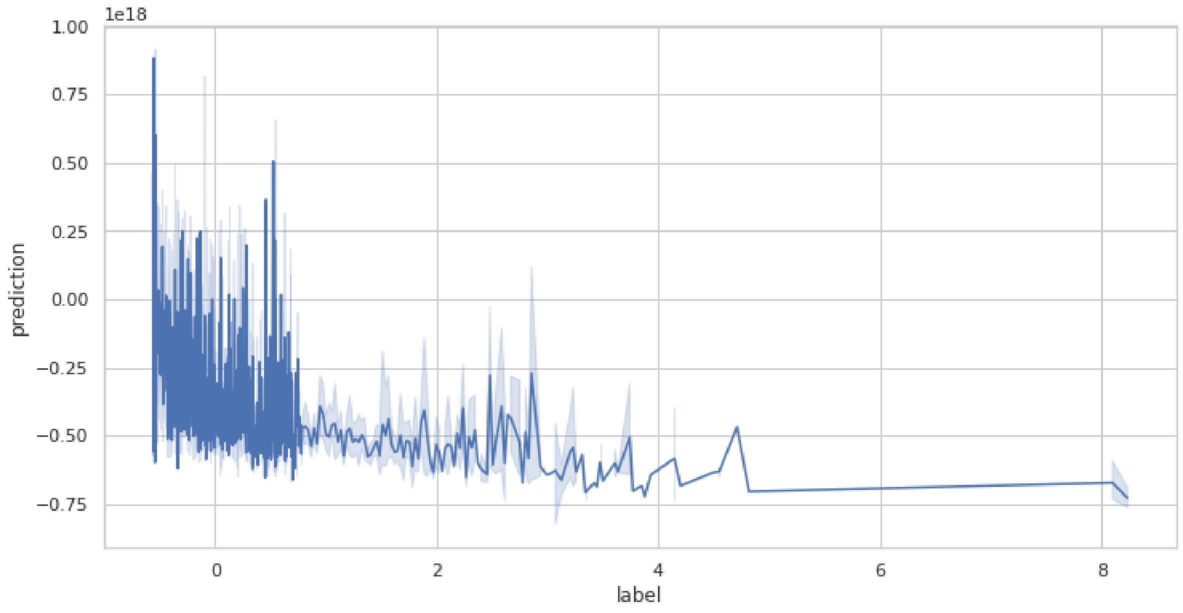


Speaking about the gradient descent graph, it is noticeable by the figure below that it presents some problem in its execution:



As visible from the graph, that experiment configuration is wrong, maybe caused by a wrong learning rate (maybe too big, causing the "explosion" of the gradient), by some of the outliers not removed that cause wrong values in the mathematical operation in which they are. Looking at the predictions scatterplot and lineplot it's visible that the predictions are not correct, resulting in exponentially wrong metrics (noticeable in Experiment section):





5.3 General considerations

It is noticeable that removing outliers improves a lot the correctness of the algorithm. Performing some more experiments could resolve some problems, like the ones faced in the configuration without outliers removal, or maybe it's possible to improve the already created model. Exploring a bigger space of values for learning rate and regression factor could be done with a bigger computational power. The project scales with the dimension of the dataset using PySpark and Spark capabilities. Trying to perform local preprocessing using pandas on the entire dataset it's impossible for me, it becomes possible using PySpark obviously taking more time than Pandas. Speaking about Pandas and Sklearn, it could be interesting to implement the project using a tool that parallelizes the two mantioned, called Dask.

6 Personal data

I declare that this material, which I now submit for assessment, is entirely myown work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties the would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

References

Apache (2009a). Spark. https://spark.apache.org/.

Apache (2009b). Spark dataframe. https://spark.apache.org/docs/latest/sql-programming-guide.html#dataframes.

Apache (2009c). Spark rdd. https://spark.apache.org/docs/latest/rdd-programming-guide.html#resilient-distributed-datasets-rdds.

Brownlee, J. (2014). An introduction to feature selection. https://machinelearningmastery.com/an-introduction-to-feature-selection/.

Brownlee, J. (2018). How to remove outliers for machine learning. https://machinelearningmastery.com/how-to-use-statistics-to-identify-outliers-in-data/.

Dave, A. (2018). Regression in machine learning. https://medium.com/datadriveninvestor/regression-in-machine-learning-296caae933ec.

Gandhi, R. (2018). Introduction to machine learning algorithms: Linear regression. https://towardsdatascience.com/introduction-to-machine-learning-algorithms-linear-regression-14c4e325882a.

Provino, A. (2019). Ridge regression. https://andreaprovino.it/ridge-regression/.