Report for joint project AMD-SM2L

Stefano Taverni (961903)
stefano.taverni@studenti.unimi.it

January exam session

Academic year 2019/2020

Abstract

This project focuses on the creation of a ridge regression model from scratch, based on the public Kaggle dataset "2013 American Community Survey". Specifically, it has been emphasised the scalability of the proposed solution together with the achieved results.

I declare that this material, which I now submit for assessment, is entirely my own 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 that 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.

1 Dataset

In the *2013 American Community Survey* approximately 3.5 million households per year are asked detailed questions about who they are and how they live [1]. The survey is divided into two types: the first one is housing data, which contains information about houses; the second one is the population data, where each entry is a person and the characteristics are properties like age, gender, whether they work.

The experiments are performed focusing on the second dataset, the population dataset. The dataset is splitted into two pieces, "a" and "b", where "a" contains states from 1 to 25 and "b" contains states from 26 to 50. In particular, the piece "a" has been chosen to perform the training and the test of the model, using only 10% of the rows.

The population dataset "a", called `ss13pusa.csv`, has exactly 1613673 rows and 283 columns. The chosen property to be predicted by the model is **PINCP**, i.e. the total person's income. About 17% of the entries are missing and, as stated by the data dictionary [2], it means they are entries associated with a person with less than 15 years old (how to tackle these missing values will be described in section Pre-processing). Some of the statistics about the **PINCP** property are:

statistic	values
valid	1.33m
missing	281k
mean	35.9k
std. deviation	54.5k
min (bottom-coded)	-11.6k
max (top-coded)	1.27m

Table 1: **PINCP** statistics

2 Data organization

The dataset is loaded into the system using the `DataFrame`. A `DataFrame` is a distributed collection of data, organized into named columns; they are equivalent to a table in a relational database, allowing easy manipulation and interrogation of the data [3]. The `DataFrame` is an high-level API provided by Apache Spark, an engine for large-scale, distributed data processing [4]. By default, Apache Spark is shipped with Hadoop's client libraries, a framework for running applications on large clusters built of commodity hardware, implementing a distrubuted file system named Hadoop Distributed File System, HDFS [5].

To load the dataset, Spark allows the initialization of a `DataFrame` using a fraction of the input file, sampling the data in a random way. To make the experiments reproducible, a seed can be specified. Once the data are loaded, the schema of the `DataFrame` can be dinamically inferred by Spark, although the time required depends on the number of rows of the `DataFrame`. Furthemore, an header is added to all columns, permitting the interrogation and the manipulation of the data by name instead of by index.

Thank to the inferred schema, the type of all the columns can be inspected. All of the data are numerical, aside three columns. These are RP, NAICSP e SOCP:

- RP is a flag value, which is set to "p" for all of the rows, meaning that the record is of person type. Since this value doesn't bring any information, it can be removed without any concern.
- NAICSP represents the "industry recode for 2013 and later based on 2012 naics codes" [2]. This column contains information about the work of a person, encoded as an alphanumerical string. The dataset contains another column which has the same purpose but using a different representation: this is INDP "industry recode for 2013 and later based on 2012 ind codes". Since INDP and NAICSP bring the same information, instead of transforming NAICSP into a numerical values, it can be suppressed without any loss of information.
- The same kind of reasoning may be applied to SOCP, "SOC occupation code for 2012 and later based on 2010 SOC codes", which can be suppressed in favour of OCCP "occupation recode for 2012 and later based on 2010 OCC codes". As a consequence, the SOCP column has been removed.

Once the pre-processing phase is completed (as described in Pre-processing), the underlying structure of the `DataFrame`, the `RDD`, is extracted. `RDD` stands for Resilient Distributed Dataset, which is a fault-tolerant collection of elements that can be operated on in parallel [6]. The `RDD` is used to organize data into `LabeledPoint`. A `LabeledPoint` is a Spark class which represents a label and features of a data point [7]. In this way, the data are easier to manipulate, and the power of MapReduce paradigm is fully available, working directly on the RDD.

3 Pre-processing

Several operations are performed in order to adjust the dataset and let the computation of the model be more precise and efficient. Before any action may be done, the dataset is divided in training and test set: in this way, the test set will not bias the model, acting as in a real scenario. The following actions are all done using the `DataFrame`.

3.1 Missing values in target label

To deal with null value in the target label, two approaches are possible.

The first one is to directly remove the entire data point. Doing so, the amount of available data points decreases, reducing also the variance of the training set. However the advantage of this approach is to reduce the risk of associating a label to a feature vector that doesn't reflect any real data point.

The second possible approach is replacing the missing value with 0. This decision is supported by the fact that, as stated in [2], a null value corresponds to a less than 16 years old person. Since such kind of person hasn't got any income, it could be good to associate a 0 value to these records.

The differences and effectivenesses of these two approaches will be inspected in section Discussion on results.

3.2 Missing values in feature vector

Since for each column the absence of a value means diffent things, an exhaustive analysis, as the one did for missing value in target label, becomes unfeasible. So, a more easy and agile strategy is to substitute the absent values with a measure of central tendency; in particular, the mean or the meadian.

Spark implements this operation via `Imputer`, which complets missing values, either using the mean or the median of the columns in which the missing values are located [7]. The mean or meadian is computed after filtering out missing values.

The `Imputer` class is part of the Spark estimator functions, which expose a `fit()` method that accepts a `DataFrame` and produce a transformer. A transformer exposes a `transform()` method which maps a `DataFrame` into a new one, perfoming a specific action [8]. In this case, `Imputer` creates a model which complets the missing value.

All the experiments are led using the mean substitution strategy, but the code can also support the meadian strategy.

3.3 Cope with outliers

An outlier may be defined as a data point that significantly differs from the other observations. The presence of outliers may be due to variability in the measurement or it may indicate experimental error.

The technique used to detect outliers is the interquartile range, or IQR, which is a measure of statistical dispersion, computed by the difference between the 75th and the 25th percentiles. The IQR is so computed as $IQR = Q3 - Q1$, where Q3 and Q1 are the third and first quartile, respectively.

With this technique, an outlier is defined as an observation which falls below $Q1 - 1.5IQR$ or above $Q3 + 1.5IQR$; once detected, it may be removed from the training set [10].

How much this method influences the model created by the regression algorithm will be further investigated in section Discussion on results.

3.4 Standardization

Standardization is the process of calculating the number of standard deviation, or z-scores, by which the value of a data point is above or below the mean of what is being observed [9]. This value is computed as:

$$z = \frac{x - \overline{x}}{s}$$

where $\overline{x}$ is the mean of the sample and s is the standard deviation of the sample.

The process of standardization is done thank to the Spark class `StandardScaler`, which transforms a `DataFrame` into a new one where each feature has unit variance and removing the mean [11].

This process is useful because the features of the dataset have different scales and in a linear regression algorithm, where the distance between units in the data is important, will otherwise lead to biased results.

3.5 Principal Component Analysis

Principal component analysis, PCA in short, is a feature extracting technique performing dimensionality reduction through a linear transformation between the data and the map space. This transformation is chosen so as to preserve the information retained by the mapped points, where the amount of information is measured through the variance of data along the various dimensions.

Spark library implements a distributed version of the PCA algorithm, exposing the `fit()` method to create a model from a given `DataFrame`. The obtained model allows the inspection of the retained variance in the new space, in order to determine how meaningful the new data are. Calling `transform()` on the model, the dimensionality reduction is applied over the input `Dataframe`.

Using this technique, the data space becomes smaller and easier to manipulate. To see how much the space can be reduced maintaining the usefulness of the data, three different experiments are performed: with 10 features in the new reduced space, 50 and 100. They are all described in section Experiments.

4 Regression algorithm

Two main algorithms are implemented in order to predict the target label. The first one is a predictor which always returns the mean of the target label in the dataset. Although this model hasn't got any real utility, this is quite useful to compare the quality of other models. Indeed, if another model performs similar or worse than this baseline algorithm, surelly there are some problems in the implementation of the algorithm. With this in mind, the mean predictor works well as a rough comparison with the main regression algorithm.

The real and useful implemented algorithm is the Ridge Regression predictor. This kind of predictor stems from the Linear Regression predictor, which is a linear function $h : \mathbb{R}^d \to \mathbb{R}$ parameterized by a vector $\mathbf{w} \in \mathbb{R}^d$, called weights, that associates to a given input vector the value $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. The Linear Regression predictor which minimizes the training error, i.e. the ERM (empirical risk minimizer) predictor for Linear Regression, with respect to the square loss function is the following:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \sum_{t=1}^{m} (\mathbf{w}^T\mathbf{x}_t - y_t)^2$$

where m is the number of points in the training set, $\mathbf{x}_t$ is the t-th feature of the training set and y_t is the label associated to $\mathbf{x}_t$. Since we are working with a convex function, the gradient of the function may be set to 0 in order to find the minimum. This lead to the following solution:

$$\hat{\mathbf{w}} = (S^T S)^{-1} S^T \mathbf{y}$$

where $\mathbf{y}$ is the vector of all the label points in the training set, and S is the design matrix, i.e. a $m \times d$ matrix (with m the cardinality of the training set and d the dimension of the feature vector) where the i-th row contains the $\mathbf{x}_i$ feature vector.

Since the matrix $S^T S$ may not be invertible, a regularizer may be introduced in the ERM computation to reduce the oscillation of $\hat{\mathbf{w}}$ due to the training set.

This regularization creates a new predictor known as Ridge Regression predictor, that has the following form:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \sum_{t=1}^{m} (\mathbf{w}^T\mathbf{x}_t - y_t)^2 + \alpha||\mathbf{w}||^2$$

where α is the regularizer term, and must be strictly greater than 0.

The solution to this problem works as before, setting the gradient of the function to 0 and computing the minimum of the function. This lead to:

$$\hat{\mathbf{w}} = (\alpha I + S^T S)^{-1} S^T \mathbf{y} \tag{1}$$

where I is the identity matrix.

The formula 1 is the one chosen in the implementation of the algorithm.

4.1 Implementation

The expression 1 can't be blindly translated into code, because the design matrix S may not fit in main memory, since the number of rows of the matrix is the same as the size of the training set; when the size of the training set increases, the algorithm will not scale at all. In order to implement the model in a scalable way, a dimensional analysis is required to assess the operation to be performed in a MapReduce style.

First of all, the product $S^T S$ yields a $d \times d$ matrix, where d is the number of features: this number can't be greater than the number of starting features, which is 279. So, in the worst case, the space required to store the product matrix is quadratic in the number of that maximum feature dimension, and so the product $S^T S$ may be saved in main memory without harm. However it is not possible to use the canonical matrix-matrix multiplication, since the matrices cannot be saved in main memory. The adopted strategy computes the product of two matrices through the outer product operation. Given two matrices A, a $n \times d$ matrix, and B, a $d \times m$ matrix, the outer product $\otimes$ is defined as follows:

$$AB = \sum_{k=1}^{d} P_k$$

where $P_k = A_k \otimes B^k = A_k (B^k)^T$, with A_k the k-th column of A and B^k the k-th row of B. In our case, where the product is performed on the same matrix, the formula can be further simplified:

$$S^T S = \sum_{k=1}^{d} x_k \otimes x_k \tag{2}$$

where x_k is the k-th row of the matrix. The operation may be implemented as a MapReduce operation in this way:

```
def outer_product(lp):
  return numpy.array([numpy.dot(i,lp.features)
                      for i
                      in lp.features])

def outer_sum(m1, m2):
```

```
  return m1+m2

result = training_set.map(outer_product) \
                     .reduce(outer_sum)
```

Listing 1: implementation of $S^T S$ in a MapReduce fashion.

The first method, `outer_product`, works on a `LabeledPoint`, and it implements the outer product in 2, multiplying the `LabeledPoint` by its transpose, yielding an $d \times d$ matrix. The method `outer_sum` does nothing more than summing two matrices. In this way, when the training set is stored in a distributed way, the computation requires $O(d^2)$ to store the intermediate results of the outer products. In this way, the computation scales up well when the number of training points increases.

The other operation that requires a deeper analysis is the product $S^T \mathbf{y}$. Both matrix and vector cannot be stored in main memory, since they depend on the number of training point, potentially large. The matrix-vector product implementation is based on the one proposed in [12]; the technique of dividing the matrix and the vector into stripes allows the computation to be performed in main memory, loading only a vertical stripe of the matrix in main memory, together with an element of the vector. Taking into account how the matrix S^T is composed, the i-th column contains the feature vector of the i-th data point in the training set: each element of this vector will be multiplied at each step with the label of the same data point it belongs to. Recalling that the training set is stored as a RDD of `LabeledPoint`, the entire multiplication boils down to a MapReduce task where in the Map phase, a scalar product between the feature vector and its corresponding label is performed, and in the Reduce phase, the intermediate vectors are added together. The code may be written as follows:

```
def stripes_product(rdd):
    return rdd.map(lambda lp: lp.label*lp.features) \
              .reduce(lambda v1, v2: v1+v2)
```

Listing 2: implementation of stripe product for $S^T \mathbf{y}$

Doing so, at each step, the required storage is $O(d)$, allowing the algorithm to scale well when the size of the training set increases.

Once assessed these operations, the others may be performed in main memory, because there is no more worry of saturating the RAM. We can now define the Ridge Regression predictor, using the `numpy` Python library:

```
def ridge_regression(rdd, alpha=1):
    A = rdd.map(outer_product).reduce(outer_sum)
    numpy.fill_diagonal(A, A.diagonal()+alpha)
    V = stripes_product(rdd)
    return numpy.linalg.inv(A).dot(V)
```

Listing 3: implementaion of Ridge Regression predictor, as in 1.

4.2 Tuning the hyperparameter

Since the Ridge Regression predictor highly depends on the value of the regularization parameter α, the search of the right value is crucial. Since α may assume any value greater than 0, the exhaustive search is not possible. In the

experiments described in section Experiments, a subset of $\mathbb{R}_+$ is used, in particular two different subsets logarithmically spaced and then joint together in the range $[10^{-8}, 100]$.

The applied technique to find out the best α exploits the K-fold cross-validation procedure. For each α in the chosen subset, a K-fold cross validation is performed, yielding the expected risk of the Ridge Regression predictor with regularization parameter α; once all the different predictors are computed, the one with the lowest expected risk is chosen. Better results may be obtained if the nested cross-validation technique is used instead of the above one, but due to the commodity hardware used during the model evaluation, the required computational power cannot be met.

The K-fold cross validation technique has to be implemented with a careful analysis on the scalability of the overall process.

First of all, the method cannot be called as many times as there are parameteres in the subset: doing so, a splitting of the training set has to be performed on each call, slowing down the performance. A better solution is, indeed, passing as input the subset of all the parameters. In this way, when a validation fold is chosen, a model can be trained on the same $K - 1$ folds for each parameter, yielding the same result as it is done in different calls.

To train the different models, the method described in 3 is internally used: in this way, the computation may scale well when the training set grows. The internal splitting into validation and training folds is done via the method of Spark library. To divide the training set into K folds, it is useful to use the method `randomSplit` where we can supply a vector of K elements, each one with value $1/K$: it returns a list of K differnt RDDs, each one of size m/K, where m is the size of the input training set. The value m may be supplied as an input parameter for the method, avoiding the count of the size of the starting RDD, which requires some time. In turn, each fold is left out to be used as validation set; the remaining ones are merged together with the method `union`.

To further improve the performance of this cross validation method, it may be possible to enable an internal caching of the RDD. Through the Spark method `cache`, an RDD can be stored entirely in main memory, allowing a faster computation. To avoid a saturation of the main memory, only the validation fold may be stored in this way. Once all of the values are computed against the validation fold, the caching is undone by the method `unpersist`, freeing up the memory for the next validation fold.

Once all the computation are performed, the cross validation method returns a dictionary of values, crafted in this way:

- the keys of the dictionary are all the α in the given subset;
- the values are lists of tuples, where each tuple contains the scaled error on one validation part, together with the coefficient of determination, or R^2, of the same validation part.

The caller of this method has to compute the K-fold cross validation estimate of the expected risk, extracting the first element of each tuple in each list, and then computing the mean for each α. The lowest of such values may be used to estimate the risk of the different models among a training set of fixed size.

When the minimun α is found, a final model is obtained, using that value against the original training set.

5 Experiments

The experiments below are all performed on Google Colaboratory [13]. The provided machine has the following specifications:

- operating system: Ubuntu 18.04 bionic
- kernel: x86_64 Linux 4.19.112+
- CPU: Intel Xeon @ 2x 2.3GHz
- GPU: none
- RAM: 13021 MiB

Five different experiments are performed: three of them focus on the choice of a good value for the size of the new space in the PCA algorithm; one explores a different approach in the pre-proccesing part, substituting the missing values of the target label with a specific number, instead of removing them; the last one tries to investigate the impact of the outlier removal from the training set, together with the different missing value substitution strategy.

All of the experiments have in common a specific setting for the global variables, which are:

- `dataset_to_load = "ss13pusa.csv"` , `dataset_sample_fraction = 0.1` and `dataset_sample_seed = 1965`, that determine the dataset to use, the amount of data to load in the starting `DataFrame` and the seed to randomly select the datapoints;
- `target_label = "PINCP"`, `forbidden_columns = ['PERNP', 'WIGP']`: these are the target label to predict and the features that are not allowed to be used in the training of the model.
- `split_weights = [0.9, 0.1]`, `split_seed = 12345`: they determine the fraction of data points to be used as training part and testing part, together with the seed for the random selection of the points; in this case, the values 0.9 and 0.1 means that 90% of the points have to be used as training set, whereas 10% of the points as test set;
- `nan_substitution_strategy = 'mean'`: this variable shows the strategy to replace the missing values in the feature vector;
- `cross_validation_folds = 5`, `cross_validation_internal_split_seed = 1234`, `cross_validation_alphas = [*np.logspace(-8, 0, 7), *np.logspace(0.1, 3, 8)]`: these variables determine the number of folds to use in the K-fold CV method, the seed to randomly split the folds and the set of parameters to work with;
- `number_of_partitions = sc.defaultParallelism * 2`: this variable may help to define a reasonable number of partitions to divide the `RDD` into; the `SparkContext` object (here called `sc`) is in charge to retrieve the number of cores the machine has with the method `defaultParellism`; this value is used to avoid the creation of too many partitions of the `RDD`, which may increase as the number of data point grows, but the host machine may not exploit due to hardware limitation, causing a performance reduction;

- `verbose=True`, which allows the printing of some additional information, at the price of slowing down the execution.

The information collected on each experiment are:

- the retained variance of the new feature vector space;
- the training and test error both for baseline predictor and Ridge Regression predictor;
- the time required to compute the K-fold cross validation method;
- the minimun risk computed by the K-fold CV, together with the corresponding regularization parameter;
- a plot of the dependency of the cross-validated risks on the α parameter of Ridge Regression;
- a plot of the dependency of the determination coefficient on the α parameter of Ridge Regression;
- a plot of how well the best model predicts the values of the test set.

5.1 Experiment 1

This experiment sets `pca_new_space_dimension` to 10, and it is meant to see how a model created with such a great reduction in feature size may perform.

The obtained results are:

1. Retained variance during PCA transformation: 0.4256
2. Time required for K-fold CV: 15 minutes
3. K-fold cross-validated risk for Ridge Regression predictor:

CV risk	α
2.13850E+09	148.398

4. Training and test error

predictor	training error	test error	R^2
baseline	2.99490e+09	3.32687e+09	0.0
Ridge Regression	2.13819e+09	2.38882e+09	0.282

5. Plots:

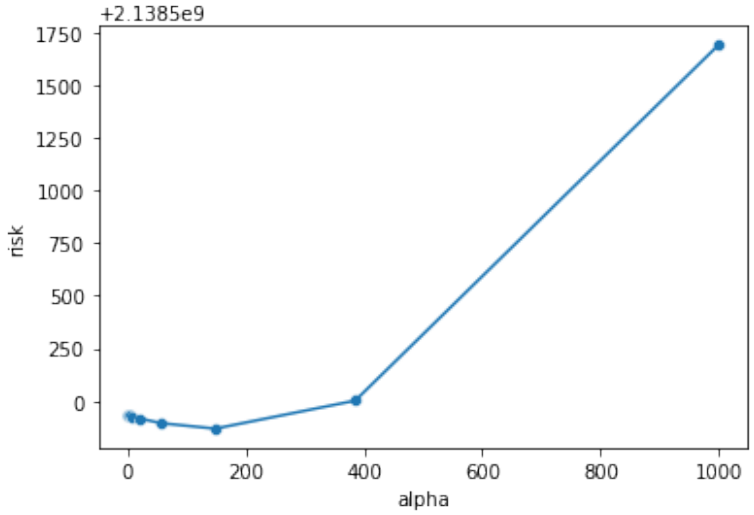

Figure 1: Dependency of K-fold CV risk on α

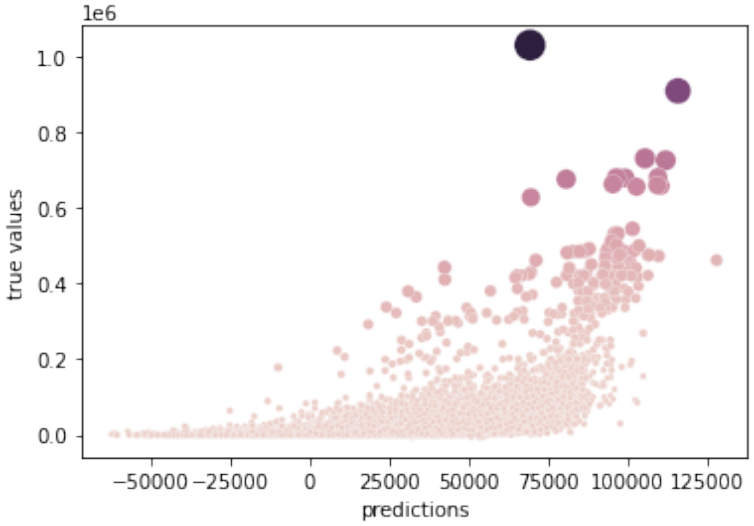

Figure 2: Test prediction of the model

5.2 Experiment 2

The second experiment analyses how a Ridge Regression model performs on a new space of 50 features. The results are:

1. Retained variance during PCA transformation: 0.6507
2. Time required for K-fold CV: 48 minutes
3. K-fold cross-validated risk for Ridge Regression predictor:

CV risk	α
1.16728E+09	148.398

4. Training and test error

predictor	training error	test error	R^2
baseline	2.99490e+09	3.32687e+09	0.0
Ridge Regression	1.16581e+09	1.29150e+09	0.612

5. Plots:

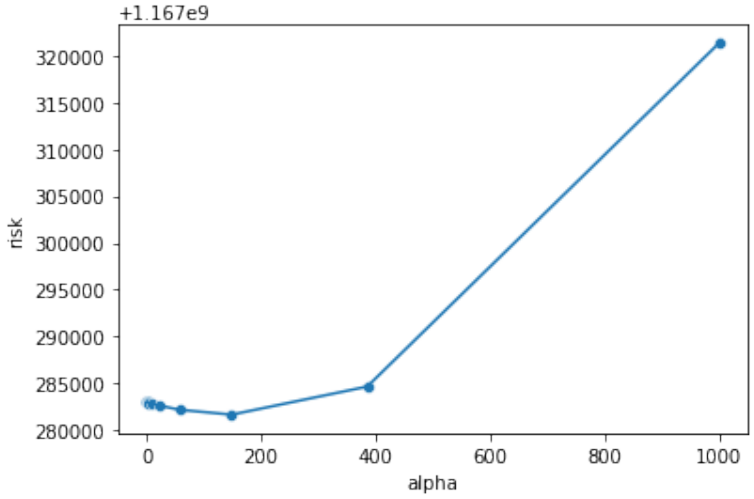

Figure 3: Dependency of K-fold CV risk on α

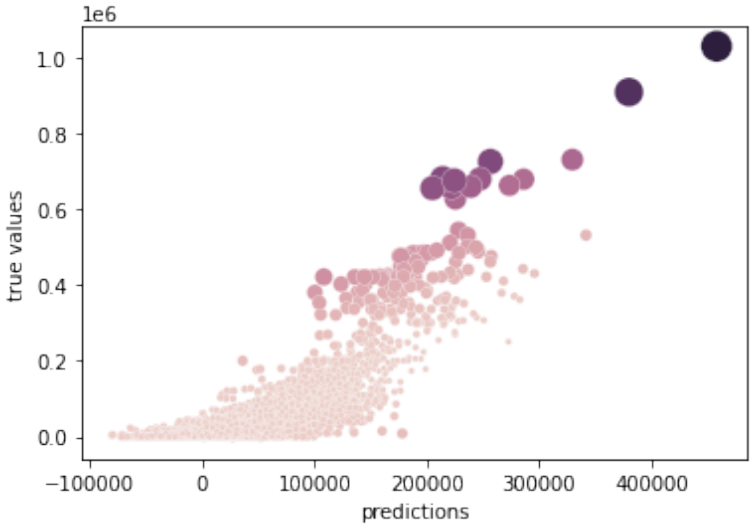

Figure 4: Test prediction of the model

5.3 Experiment 3

This experiment ends the exploration of the number of features to select as PCA output. In this case, the new space has 100 features. The results are:

1. Retained variance during PCA transformation: 0.8094
2. Time required for K-fold CV: 1 hour, 54 minutes
3. K-fold cross-validated risk for Ridge Regression predictor:

CV risk	α
7.30017E+08	57.1667

4. Training and test error

predictor	training error	test error	R^2
baseline	2.99490e+09	3.32687e+09	0.0
Ridge Regression	7.28430e+08	8.43216e+08	0.746

5. Plots:

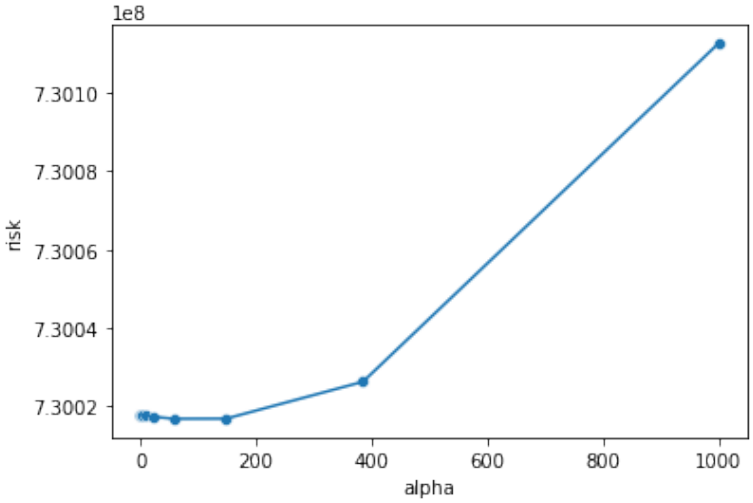

Figure 5: Dependency of K-fold CV risk on α

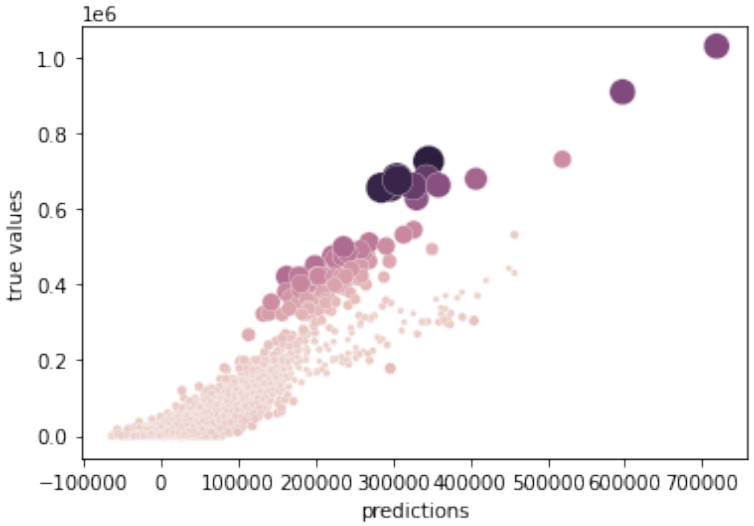

Figure 6: Test prediction of the model

5.4 Experiment 4

In this experiment, a different method of missing value substitution is used; in particualar, the missing **PINCP** labels to predict are substituted by 0, for the reasons discussed in Missing values in target label. The variable `missing_label_substitution` is set to `"zero"`. The results are:

1. Retained variance during PCA transformation: 0.6521
2. Time required for K-fold CV: 1 hour, 9 minutes
3. K-fold cross-validated risk for Ridge Regression predictor:

CV risk	α
9.60927E+08	57.1667

4. Training and test error

predictor	training error	test error	R^2
baseline	2.68978e+09	2.66567e+09	0.0
Ridge Regression	9.59453e+08	9.37526e+08	0.648

5. Plots:

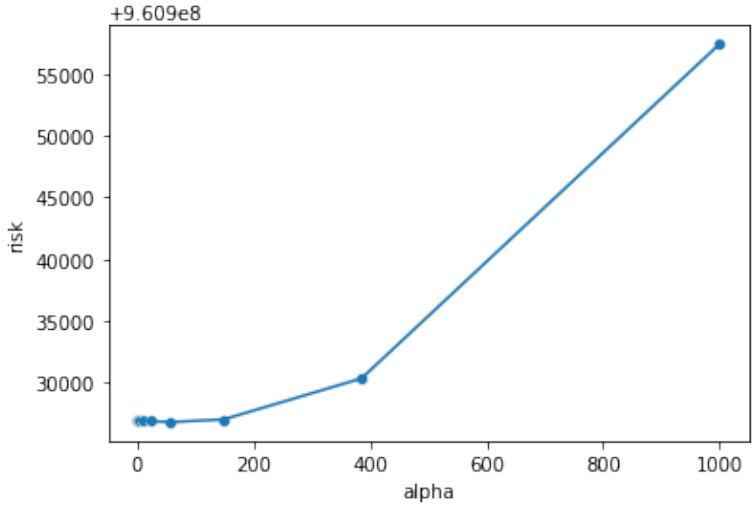

Figure 7: Dependency of K-fold CV risk on α

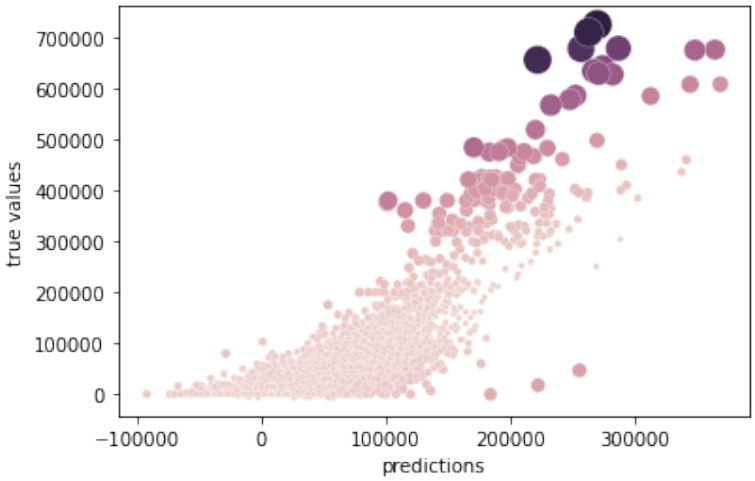

Figure 8: Test prediction of the model

5.5 Experiment 5

In this experiment, in addition to the missing value substitution with 0 for target label (setting `missing_label_substitution` to `"zero"`), the outlier removal procedure is applied. The variable `outlier_strategy` is set to `"iqr"`, meaning that the interquartile range measure is used to detect outliers. The obtained results are:

1. Retained variance during PCA transformation: 0.6540
2. Time required for K-fold CV: 1 hour, 7 minutes
3. K-fold cross-validated risk for Ridge Regression predictor:

CV risk	α
1.60109E+08	148.398

4. Training and test error

predictor	training error	test error	R^2
baseline	5.37118e+08	2.76667e+09	0.0
Ridge Regression	1.59939e+08	1.30941e+09	0.509

5. Plots:

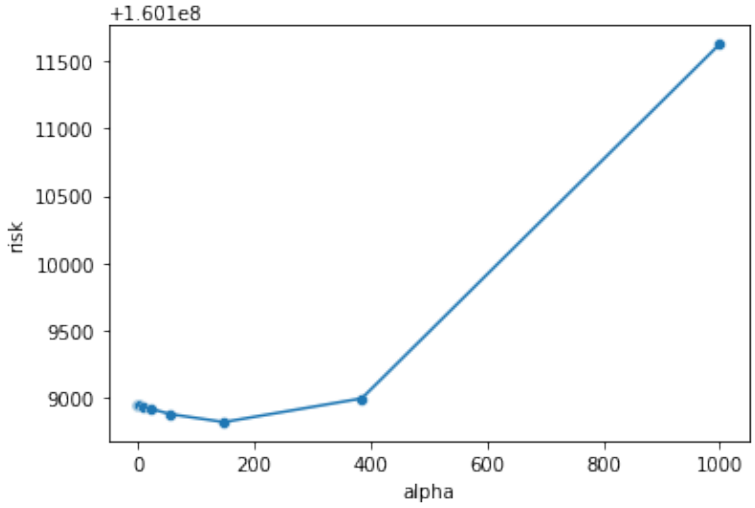

Figure 9: Dependency of K-fold CV risk on α

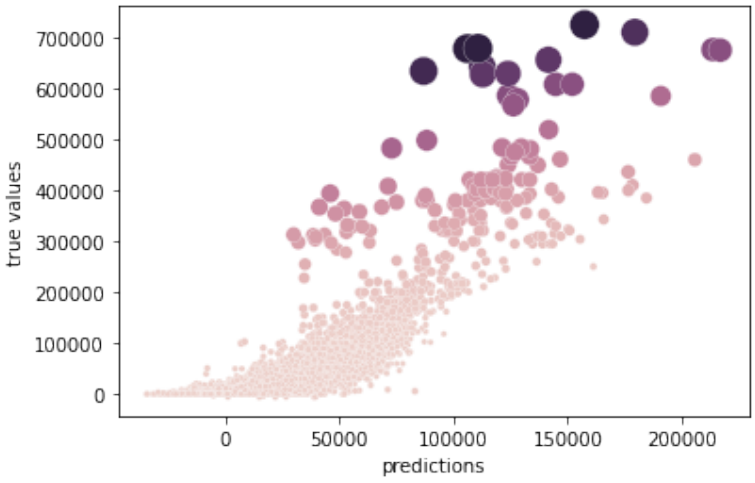

Figure 10: Test prediction of the model

6 Discussion on results

The first experiment clearly shows that a model trained with few features doesn't perform well. Indeed, the retained variance after the PCA algorithm is below 0.5, meaning that lots of information have been lost. This missing is well reflected in the value of R^2, which is 0.282: the model is not the correct one. In addition, both training and test error are extremely high, showing a possible presence of underfitting. Figure 2 shows that the model predicts using a really small subset of values, in particular between 0 and 75000, performing some correct predictions only in this range.

Adding more features to the PCA algorithm output leads to different models with better fitting. Using 50 features halves the K-fold CV risk, and also training and test error are reduced. Even though the order of magnitude of the errors is high, the new model fits better, scoring a 0.612 as R^2 value. Plot 4 shows this improvement, where the range of predicted values starts to increase. Even better results are obtained by using 100 features in the output space of PCA, decreasing the order of magnitude of the errors by 1. This model scores 0.746 as R^2, meaning that the fitting is the best so far.

In the third experiment, a change may be seen in the shape of figure 5. Although the parabolic shape is the same as the previous experiments, the minimun is found in a different position: this fact may be interpreted as the problem starts to be less ill-posed, and the data start becoming more significant. Furthermore, figure 6 shows the model improvements, where the predictions range in a wider subset and the shape of the predictions resembles more and more a straight line.

The fourth experiment shows intresting results. First of all, the cross-validated risk is much lower than one obtained in the second experiment, and also training and test errors, both for baseline predictor and Ridge Regression predictor, are lower. The substitution of missing values in the target label with 0, instead of their removal, leads to a greater number of training data, and so the training time increases. However the resulting model performs better than the one in Experiment 2, meaning that the new observed features allow a more precise prediction.

Finally, the last experiment clearly reveals an overfitting. The target labels kept by interquantile range technique are in the interval $[-11600.0, 95000.0]$, taking away about 25 thousands data point from training set. Besides the lowest CV risk scored, together with a good training error for both baseline predictor and Ridge Regression predictor, the trained model doesn't predict well data point in test set. Indeed, the test error is one order of magnitude greater than the training error: this means that the model overfits the training data, learning some features unique to the training set. Furthermore, figure 10 shows a great number of mislabeled test points, suggesting a lack of diversity in the training set.

What all the experiments have in common is the fact that, once the minimun regularization parameter is found, the cross-validated risk increases fastly. This shows how sensitive the model is in the choice of a correct hyperparameter. Furthermore, there is another aspect to consider: both training and test errors are extremely high. This may be due to an underfitting of the model, suggesting that a wider training set has to be used in order to capture diffent properties of the starting dataset.

7 Further investigations

Due to hardware limitation, computationally heavy tasks cannot be carried. In presence of a more suitable hardware, further trials can be performed, for example:

- use a greater portion of the initial dataset to work with;
- use both population datasets;
- increase the number of folds in the K-fold cross validation procedure;
- inspect a denser subset for the regularization parameter α;
- try to use a greater number of features;

References

[1] Kaggle, 2013 American Community Survey
https://www.kaggle.com/census/2013-american-community-survey

[2] 2013 ACS PUMS DATA DICTIONARY
https://www2.census.gov/programs-surveys/acs/tech_docs/pums/data_dict/PUMSDataDict13.txt

[3] Spark SQL, DataFrames and Datasets Guide
https://spark.apache.org/docs/latest/sql-programming-guide.html

[4] Spark Overview
https://spark.apache.org/docs/latest/index.html

[5] Apache Hadoop, Home
https://cwiki.apache.org/confluence/display/hadoop

[6] Apache Spark 3.0.1, Resilient Distributed Datasets (RDDs)
https://spark.apache.org/docs/latest/rdd-programming-guide.html#resilient-distributed-datasets-rdds

[7] PySpark 3.0.1 documentation
https://spark.apache.org/docs/latest/api/python/pyspark.mllib.html

[8] Apache Spark 3.0.1, ML Pipelines
https://spark.apache.org/docs/latest/ml-pipeline.html#pipeline-components

[9] Wikipedia The Free Encyclopedia, Standard Score
https://en.wikipedia.org/wiki/standard_score

[10] Wikipedia The Free Encyclopedia, Interquartile range
https://en.wikipedia.org/wiki/Interquartile_range

[11] Spark 3.0.1 ScalaDoc, StandardScaler
https://spark.apache.org/docs/latest/api/scala/org/apache/spark/ml/feature/StandardScaler.html

[12] Mining of Massive Dataset, [A. Rajaraman J. Ullman]
Cambridge University Press, 3rd ed.

[13] Google Colaboratory,
https://colab.research.google.com/notebooks/intro.ipynb