Report

M-MLR-900

(Alexandre Guichet, Alexis Auriac, Benjamin Feller)

Machine Learning - Supervised Learning

February 1, 2023

Introduction

We did part 1, 2, 3, and 4 but did not have time to do part 5.
Division of labor:

- Alexandre: part 1
- Benjamin: part 2
- Alexis: part 3 and 4, writing report

Part 1

Problem 1

The goal of this exercise is to work with statistical notions such as mean, standard deviation, and correlation.

Write a file named artificial_dataset.py that generates a numerical dataset with 300 datapoints (i.e. lines) and at least 6 columns and saves it to a csv file or to a numpy array in a binary python file.

The columns must satisfy the following requirements:

- they must all have a different mean
- they must all have a different standard deviation (English for "écart type")
- at least one column should contain integers.
- at least one column should contain floats.
- one column must have a mean close to 2.5.
- some columns must be positively correlated.
- some columns must be negatively correlated.
- some columns must have a correlation close to 0.

Solution.

See `exercise_1/artificial_dataset.py` for the code.

The generated data can be found in `exercise_1/data.npy`, it contains 300 datapoints with 6 columns.

Let's go over each point mentioned in the subject one by one.

All columns must have a different mean

- column 1: 2.58
- column 2: 0.898
- column 3: 1.686
- column 4: 4.707
- column 5: 0.349
- column 6: 10.115

All columns must have a different standard deviation

- column 1: 1.752
- column 2: 0.827

- column 3: 1.998
- column 4: 1.889
- column 5: 2.430
- column 6: 1.352

At least one column should contain integers
Column 1 contains integers.
At least one column should contain floats
All columns except column 1 contain floats.
One column must have a mean close to 2.5
Column 1 has a mean of 2.58.
Columns correlations
Using `numpy.corrcoef` to get a correlation matrix for column 4, 5, and 6 we get this:

$$\begin{pmatrix} 1. & -0.5595593 & 0.66913029 \\ -0.5595593 & 1. & 0.03168539 \\ 0.66913029 & 0.03168539 & 1. \end{pmatrix}$$

Column 4 is **negatively correlated** with column 5.
Column 4 is **positively correlated** with column 6.
Column 5 is **has a correlation close to 0** with column 6.

Part 2

Problem 2

A dataset representing a population is stored in dataset.csv inside the `project/ex_2_metric/folder`.

Define a metric in this dataset, which means define a dissimilarity between the samples, by taking into account all their features (columns of the dataset).

Some features are numerical and others are categorical, hence you can not use a standard euclidean metric, and you need to define a custom metric, like we did in the `code/metrics/hybrid_data/` exercise during the course. Compute the mean dissimilarity and the standard deviation of the dissimilarity distribution that you obtain, and save the dissimilarity matrix to a file (e.g. a npy file).

Importantly, you must define and explain which features are more important with this metric, since you have to balance the contribution of all the features. Your metric should be meaningful in the sense that not all feature values should induce the same contribution to the dissimilarity : the music style ”technical death metal” is closer to ”metal” than it is to ”classical”.

Solution.

The columns in the dataset are age, height, job, city, and favorite music style.

The age and height features are numerical and their dissimilarity can be computed using the euclidean distance.

The job, city, and favorite music style features are not numerical so we must define a custom metric for each of them.

Job

See `job.py` for the code.

Here are the jobs we can find in the dataset: designer, fireman, teacher, doctor, painter, developper, and engineer.

We will assign an art, science, and altruism value from 0 to 10 for each job.

The values given are VERY subjective, we don't mean any offense.

	art	science	altruism
designer	8	3	4
fireman	0	7	10
teacher	4	5	6
doctor	2	9	8
painter	10	2	3
developper	3	6	1
engineer	4	8	2

Using the euclidean distance we get the following dissimilarity matrix:

	designer	fireman	teacher	doctor	painter	developper	engineer
designer	0.000000	10.770330	4.898979	9.380832	2.449490	6.557439	6.708204
fireman	10.770330	0.000000	6.000000	3.464102	13.190906	9.539392	9.000000
teacher	4.898979	6.000000	0.000000	4.898979	7.348469	5.196152	5.000000
doctor	9.380832	3.464102	4.898979	0.000000	11.747340	7.681146	6.403124
painter	2.449490	13.190906	7.348469	11.747340	0.000000	8.306624	8.544004
developper	6.557439	9.539392	5.196152	7.681146	8.306624	0.000000	2.449490
engineer	6.708204	9.000000	5.000000	6.403124	8.544004	2.449490	0.000000

We can see that jobs like painter and fireman have a high dissimilarity (13.2) while painter and designer have a low dissimilarity (2.4).

City

See `city.py` for the code.

Here are the cities we can find in the dataset: paris, marseille, toulouse, madrid, and lille.

We will evaluate cities using 4 metrics: distance (using coordinates), population, country, and if it is a capital.

Here is the data we used:

	coordinates	population	country	capital
paris	(48.8566, 2.3522)	2161000	France	True
marseille	(43.2965, 5.3698)	861635	France	False
toulouse	(43.6047, 1.4442)	471941	France	False
madrid	(40.4168, 3.7038)	3223000	Spain	True
lille	(50.6292, 3.0573)	232741	France	False

To measure distance we use the library geopy we then use log10 so that it doesn't impact the dissimilarity too much.

We compare the population using euclidean distance. We then use log10 so that it doesn't impact the dissimilarity too much.

Not being from the same country adds a dissimilarity of 10.

One being a capital and not the other adds a dissimilarity of 5.

We then compute the square root of the sum of the squares to get the dissimilarity.

Using this method we get the following dissimilarity matrix:

	paris	marseille	toulouse	madrid	lille
paris	0.000000	7.897956	7.986461	11.675366	8.031395
marseille	7.897956	0.000000	5.590724	12.869235	5.798577
toulouse	7.986461	5.590724	0.000000	12.902215	5.378761
madrid	11.675366	12.869235	12.902215	0.000000	12.920325
lille	8.031395	5.798577	5.378761	12.920325	0.000000

Favorite music style

See `music.py` for the code.

Here are the cities we can find in the dataset: trap, hiphop, metal, rock, rap, classical, other, jazz, and technical death metal.

It is hard to find metrics for music styles, we decided to give each pair of music styles a dissimilarity based on personal knowledge.

Some important assumptions that influenced our choices:

- trap, hiphop, and rap are related
- metal, rock, and technical death metal are related
- other is very vague and large, so we gave it a dissimilarity of 10 for all music styles

We get this dissimilarity matrix:

	trap	hiphop	metal	rock	rap	classical	other	jazz	technical death metal
trap	0	3	20	20	5	20	10	15	20
hiphop	3	0	18	17	5	15	10	12	20
metal	20	18	0	5	10	14	10	20	5
rock	20	17	5	0	10	12	10	17	13
rap	5	5	10	10	0	15	10	15	20
classical	20	15	14	12	15	0	10	8	20
other	10	10	10	10	10	10	0	10	10
jazz	15	12	20	17	15	8	10	0	20
technical death metal	20	20	5	13	20	20	10	20	0

Overall dissimilarity

Adjusting means

See `exercise_2.py` for the code.

If we look at the means and standard deviation for each column we get

	mean	std
age	6.456159	4.892174
height	6.000623	4.679549
job	6.259779	3.747122
city	6.950629	5.205568
favorite music style	11.796500	6.390359

The mean isn't the same, which means that because of the way we compute dissimilarity some columns have inherently more value, that's an issue. It also makes standard deviations impossible to compare.

We will try to have all means equal 10 (10 is arbitrary, it makes things relatively readable).

After adjustment we get the following:

	mean	std	adjusted std
age	6.456159	4.892174	7.577531
height	6.000623	4.679549	7.798439
job	6.259779	3.747122	5.986029
city	6.950629	5.205568	7.489348
favorite music style	11.796500	6.390359	5.417165

Deciding feature importance

Age: The age of a person changes a lot of a person, beliefs, physical ability, experience, etc...

$\Rightarrow 3$

Height: Beside appearance and physical ability (in some contexts) this doesn't change much
⇒ 1
Job: job is closely related to knowledge, ability, wealth, status, and more
⇒ 3
City: Geographical location is related to culture, opportunities, language, and more
⇒ 2.5
Favorite music style: As explained before, this dissimilarity is very hard to measure and music styles have a lot of intersections
⇒ 0.5

Result matrix

See `dissimilarity_matrix.npy` for the final dissimilarity matrix.
mean: 58.386
standard deviation: 19.976

Side note: most similar and dissimilar items

Most similar items:

	age	height	job	city	favorite music style
102	27.086348	180.242244	teacher	madrid	jazz
163	26.968458	179.665081	teacher	madrid	jazz

Most dissimilar items:

	age	height	job	city	favorite music style
40	10.851506	169.432515	fireman	lille	jazz
85	46.610179	181.358551	fireman	toulouse	trap

Part 3

Problem 3

We would like to predict the winner of a Basketball game, as a function of the data gathered at half-time.

The dataset is stored in `project/ex_3_classification_NBA/`:

- The inputs x representing the features are stored in `inputs.npy`.
- The labels y are stored in `labels.npy`. If the home team wins, the label is 1, -1 otherwise.

You are free to choose the classification method. However, it is required that you explain and discuss your approach in your report. For instance, you could discuss:

- the performance of several methods and models that you tried.

- the choice of the hyperparameters and the method to tune them.
- the optimization procedure.

Your objective should be to obtain a mean accuracy superior than 0.85 on a test set or as a cross validation score.

scikit-learn model validation and accuracy score

scikit-learn cross validation

Several methods might work, including some methods that we have not explicitely studied in the class. Do not hesitate to try such methods.

Solution.

I tried 3 models for this exercise: KNeighbors, RandomForest and Logistic Regression.

For each model I chose some hyperparameters to tune and then used GridSearchCV or RandomizedSearchCV to tune them and measure the mean accuracy of the model.

GridSearchCV and RandomizedSearchCV both use cross validation.

KNeighbors Classifier

See `kneighbors.py` for the code.

Hyperparameters:

- n_neighbors: can cause overfitting or underfitting if it is too small or too big respectively
- weights: give all points the same or a different importance
- p: distance measure, 1 is for manhattan, 2 is for euclidean

The best hyperparameters found using GridSearchCV were:

```
{
    'n_neighbors': 11,
    'p': 2,
    'weights': 'uniform'
}
```

They give a mean score of 0.804.

RandomForest Classifier

See `random_forest.py` for the code.

Hyperparameters:

- n_estimators: number of trees
- max_features: number of features to consider at every split
- max_depth: maximum number of levels in tree
- min_samples_split: minimum number of samples required to split a node

- min_samples_leaf: minimum number of samples required at each leaf node
- bootstrap: method of selecting samples for training each tree

The best hyperparameters found using “‘RandomizedSearchCV“‘ were:

```
{
    'n_estimators': 400,
    'min_samples_split': 10,
    'min_samples_leaf': 4,
    'max_features': 'sqrt',
    'max_depth': None,
    'bootstrap': False
}
```

They give a mean score of 0.816.

LogisticRegression

See `logistic_regression.py` for the code.
Hyperparameters:

- C: controls regularization strength (prevents overfitting)
- solver: some solvers are better depending on: number of features, size of dataset, number of classes, etc...
- max_iter: maximum number of iterations taken for the solvers to converge

I am confused about the penalty hyperparameter, it seems to be deprecated in favor of solver but I am not sure. Furthermore only some values of solver are compatible with some values of penalty, I am not sure how to take that into account when using GridSearchCV or RandomizedSearchCV. I eventually decided to exclude it from the tuning.

The best parameters found using GridSearchCV were:

```
{
    'C': 0.01,
    'solver': 'newton-cg',
    'max_iter': 1000
}
```

They give a mean score of 0.904.
This classification method achieves a mean score superior to 0.85.

Part 4

Problem 4

We would like to predict the amount of electricity produced by a windfarm, as a function of the information gathered in a number of physical sensors (e.g. speed of the wind, temperature, ...).

The dataset is stored in `project/ex_4_regression_windfarm/`:

- The inputs x are stored in `inputs.npy`.
- The labels y are stored in `labels.npy`.

The instructions are the same as in 3.

Your objective should be to obtain a R2 score superior to 0.85 on a test set or as a cross validation score.

Coefficient of determination

scikit-learn r2 score

Several methods might work, including some methods that we have not explicitly studied in the class. Do not hesitate to try such methods.

Solution.

Similarly to the previous part, we tried 4 different regression models, until we arrived to an accuracy superior to 0.85. The models we tried are Linear regression, Polynomial regression, Ridge regression, and Lasso regression. We briefly considered using Logistic regression before realizing that the dependent variable needs to be binary for it to work. For each model we tried tuning the hyperparameters (except for linear regression and polynomial regression) and checked the result with cross validation.

Linear regression

See `linear_regression.py` for the code.

Mean R2: 0.688

Polynomial regression

See `polynomial_regression.py` for the code.

Mean R2 (degree=2): 0.750

Mean R2 (degree=3): 0.722

The reduced accuracy with degree=3 can be explained by overfitting.

degree=4 is too complex to be tested on our computers.

This method yields better results than linear regression but is still not enough.

Ridge regression

See `ridge.py` for the code.

Mean R2: 0.765

Lasso regression

See `lasso.py` for the code.

Mean R2: 0.887

This is a satisfactory solution.

VIF / multicollinearity

See `data_vif.py` for the code.

The mean VIF of the dataset is: 7.826

Standard deviation of VIF: 0.846

The VIF of the data is moderately high, which is a sign of multicollinearity.

This could explain why Ridge and Lasso regression are better for this dataset.