Exercise 2 - Regression

Machine Learning

2020/2021 | Technische Universität Wien

Grupo 06
Diogo Braga (E12007525)
Enrico Coluccia (E12005483)
Matthias Kiss (01218325)

Datasets

-Seoul Bike Sharing Demand:

Prediction of bike count required at each hour for the stable supply of rental bikes based on weather information and dates

-SGEMM GPU kernel performance:

Measure of the running time of a matrix-matrix product on GPU kernel, where all matrices have size 2048 x 2048.

-Metro Interstate Traffic Volume:

Hourly Interstate 94 Westbound traffic volume for midway between Minneapolis and St Paul, MN.

Seoul Bike Sharing Demand

- 8760 instances
- 14 attributes
- Dependent variable: Bike Count
- No missing values

Data:

- Almost numerical (Rented Bike count, Hour, Temperature, Humidity, Windspeed, Visibility, Dew point temperature, Solar radiation, Rainfall, Snowfall)
- Categorical (Seasons, Holiday, Functioning Day)

SGEMM GPU kernel performance

- 241.600 instances
- 18 attributes
- 4 dependent variables: Independent running time with the same parameters
- No missing values

Data:

- Almost Numerical Data indicated technical settings of the machines/kernel like Stride, Memory Shape, Workgroup Shape etc..

Metro Interstate Traffic Volume

- 48.204 Instances
- 9 Attributes
- Dependent variable: traffic volume
- No missing values

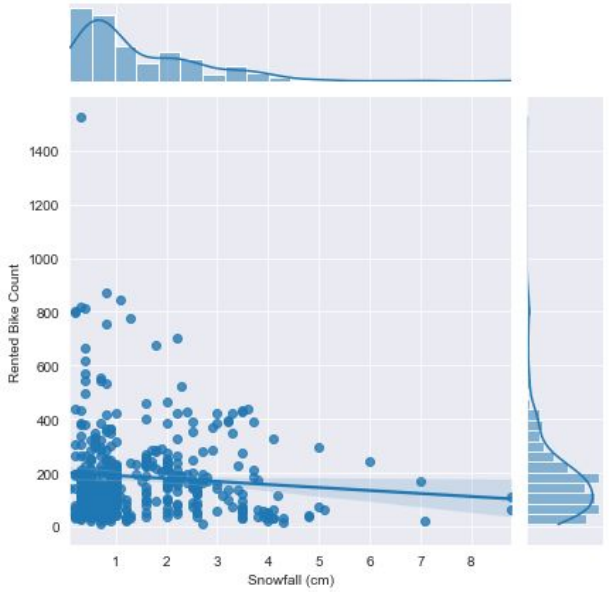
Data:

- Numerical (Temperature, Snowfall, clouds, Datetime)
- Categorical (Holiday, Weather, Weather Description)

Data Pre-processing

Seoul Bike Sharing Demand & Metro Interstate Traffic Volume

1. One Hot Encoding on Categorical Data (Seasons, Holiday etc..)
2. Feature Engineering over the Datetime
    a. Year
    b. Month
    c. Day
3. Outlier Detection and Data visualization



Data Pre-processing

-SGEMM GPU kernel performance

1. No encoding needed (almost numerical data)
2. Added one unique column ‘Run’ as the mean of the 4 Running Time

Data preparation:

- Data splitting with Holdout Method (70/30)
- Standard Scaler to scale the data and to improve the performance over K-NN and GDA

Experimental Results - GDA

	Coefficient of determination	Mean Absolute Error	Mean Squared Error	Root Mean Squared Error	Time (s)
Seoul Bike Sharing Dataset	0.54	323.2	191879.73	438.04	0.01
SGEMM GPU Performance	1.00	4.63e-12	3.24e-23	5.69e-12	0.20
Metro Interstate Traffic Volume	0.15	1608.83	3339779.15	1827.50	0.22

Experimental Results - KNN

	Coefficient of determination	Mean Absolute Error	Mean Squared Error	Root Mean Squared Error	Time (s)
Seoul Bike Sharing Dataset	0.52	303	200380.78	447.63	0.05
SGEMM GPU Performance	1.00	0.65	0.96	0.98	7.65
Metro Interstate Traffic Volume	0.78	606.01	870349.18	932.92	2.75

Experimental Results - Decision Tree

	Coefficient of determination	Mean Absolute Error	Mean Squared Error	Root Mean Squared Error	Time (s)
Seoul Bike Sharing Dataset	0.77	179.96	96960.2	311.38	0.07
SGEMM GPU Performance	1.00	0.22	0.72	0.85	4.84
Metro Interstate Traffic Volume	0.69	606.88	1236429.99	1111.94	0.47

Experimental Results - Random Forest

	Coefficient of determination	Mean Absolute Error	Mean Squared Error	Root Mean Squared Error	Time (s)
Seoul Bike Sharing Dataset	0.88	136.38	52347.05	228.79	3.41
SGEMM GPU Performance	1.00	0.09	0.14	0.38	196
Metro Interstate Traffic Volume	0.83	527.4	670196.24	818.65	32.75

Results Analysis

Linear Regression: this algorithm doesn't perform well on the *Metro Interstate Traffic Volume* and *Seoul Bike sharing* datasets. Even with scaled data the coefficient of determination is very low. This is caused by non-linear association in the data. On the *SGEMM GPU Performance* data performs very well with optimal determination and results. The computational time remains low even with huge data.

k-NN: We can see a big improvement on the metrics for the *Metro Interstate Traffic Volume*. The performances on the *Seoul Bike sharing* dataset are slightly the same, but *Scaling* the data brings to an improvement on the overall performance (COD =0.78, MAE=193.66, MSE=98327.91 RMSE=313.57). It performs well on the *SGEMM GPU Performance*. The computational time increases a lot with huge data sets (above 7,5 seconds for 200k instances)

Results Analysis

Decision Tree Regression: The algorithm performs well on the Seoul Bike Sharing dataset (in respect of the first two algorithms) but it performs worse on the Metro Interstate dataset in respect to the K-NN. Scaling does not improve the overall performance on these dataset. The coefficient of determination on the SGEMM dataset remains optimal. The complexity time grown with the dataset dimension but remains reasonable.

Random Forest Regressor: This algorithm results the best in terms of performance on out dataset but the regression task is very time consuming. It takes 3.4 seconds for the small dataset and more than 3 minutes for the largest one. Even here the effect of Scaling is not pronounced.

Linear Regression - Overview

Goal of linear regression:

Find a linear function *y(x)* that describes the relationship between a all the points in a measurement y_m and one (or multiple) measurement parameters *x*, while minimizing the deviation of *y(x)* from the measured y_m.

Finding the right *y(x)* to represent y_m:

The deviation of *y(x)* to y_m can be measured in multiple ways using so called loss functions. These functions are used to adapt *y(x)* by minimizing the deviation to the measurement in an iterative procedure.

Linear Regression - Theory

Equation to be solved:

- First a set of measurements y_m (e.g. traffic volume) has to be taken and a set of parameters $\{x_1, ..., x_N\}$ (e.g. weather, time, ...) has to be found.
- Then a linear regression function of the following form can be constructed:

$$y(x_1, ..., x_N) = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + ... + w_N \cdot x_N = w_0 + \sum_{n=1}^{N} w_n \cdot x_n$$

- This is the most general form of a linear regression function in N dimensions.
- In order to to represent the set of measurements y_m with this function $y(x_n)$ one must adapt the bias weight w_0 (= function offset), as well as the weights w_n for each parameter x_n.

Linear Regression - Theory

Finding as set of weights I:

- The weights $\{w_0, w_n\}$ are to be chosen such that the function $y(x_n)$ most closely represents the relationship between the measurement y_m and the parameters x_n.
- Therefore the deviation of $y(x_n)$ and y_m has to be minimized in every point of y_m.
- To minimize this deviation we first need to define a measure for the deviation called loss function.
- At their core loss functions are calculated such that the difference $y(x_n) - y_m$ is calculated for every point of measurement.
- Then different mathematical operations can be performed on the difference before summing over all points, resulting in a single loss value for total deviation of $y(x_n)$ from y_m using one set of weights.

Linear Regression - Theory

Finding as set of weights II:

- Once an appropriate loss function has been chosen an iterative procedure can be used to find weights that minimize the loss.
- Therefore starting at iteration *t=0*, a set of initial weights (randomly chosen, set to a constant value, ...) and the resulting regression function $y(x_n, w_n^t)$ are used to calculate the partial derivative of the loss function with respect to each weight $\{w_0, w_n\}$
- This is then used to incrementally update each of the weights: $$w_n^{t+1} = w_n^t - \alpha \cdot \frac{\partial}{\partial w_n} lossFunction(y(x_n, w_n^t), y_m)$$
- The idea here is to follow the gradient of the loss function until the minimum is reached.
- The additional parameter α is called learning rate and determines the rate of change of the weights in each iteration.

Linear Regression - Theory

Finding as set of weights III:

Two important points have to be mentioned when employing this iterative procedure:

1) For the update of each of the weights $\{w_0^{t+1}, w_n^{t+1}\}$ the linear regression function $y(x_n, w_n^t)$ needs to be calculated using the weights from the previous iteration t. Then AFTER updating ALL weights the updated regression function $y(x_n, w_n^{t+1})$ can be calculated.

2) The right choice of the learning rate α is imperative for convergence of the regression function. When α is small only little progress is made each iteration. However if α is too large then the solution will not converge but oscillate around the minimum at an arbitrary distance to the minimum.

Linear Regression - Theory

Loss Functions we investigated and their derivatives: (calculated assuming x_{0k} for w_0 is 1 for all k, where k is the index over all K individual points y_{mk} in the measurement y_m):

- Mean Square Error:

$$\mathrm{MSE} = \frac{1}{K}\sum_{k=1}^{K}\left(y_k(x_n, w_n^t) - y_{mk}\right)^2 \qquad \frac{\partial}{w_n}\mathrm{MSE} = 2 * \frac{1}{K}\sum_{k=1}^{K}\left(\left(y_k(x_n, w_n^t) - y_{mk}\right)\cdot(-x_{nk})\right)$$

- Root Mean Square Error:

$$\mathrm{RMSE} = \sqrt{\frac{1}{K}\sum_{k=1}^{K}\left(y_k(x_n, w_n^t) - y_{mk}\right)^2} \qquad \frac{\partial}{w_n}\mathrm{RMSE} = \frac{\frac{1}{K}\sum_{k=1}^{K}\left(\left(y_k(x_n, w_n^t) - y_{mk}\right)\cdot(-x_{nk})\right)}{RMSE(y(x_n, w_n^t), y_m)}$$

- Mean Absolute Error:

$$\mathrm{MAE} = \frac{1}{K}\sum_{k=1}^{K}\left|y_k(x_n, w_n^t) - y_{mk}\right| \qquad \frac{\partial}{w_n}\mathrm{MAE} = \frac{1}{K}\sum_{k=1}^{K}\left(\begin{cases} +1, & \text{for } y_k(x_n, w_n^t) > y_{mk} \\ -1, & \text{for } y_k(x_n, w_n^t) < y_{mk} \\ \text{undefined}, & \text{for } y_k(x_n, w_n^t) = y_{mk} \end{cases}\right)$$

Linear Regression - Theory

Convergence criterion for the Algorithm:

This being an iterative procedure a criterion is needed for when to stop the algorithm and accept the current weights as the solution of the minimization problem.

We tried three different criteria, where ε is always a small number, and t is any iteration:

1) loss function < ε
2) |loss function (t) - loss function (t+1)| < ε
3) |(loss function (t) - loss function (t+1))/ (loss function (t+1))| < ε

-> Option 1 needs prior knowledge about the loss function value at convergence to set ε.
-> Option 2 does not take the actual value of the loss function into account, that can be very large.
-> Option 3, the relative change in the loss function, turned out to be the best criterion as it is independent of the actual value of the loss function.

Linear Regression - Theory

Learning rate adaption:

Generally it is recommended to adapt the learning rate α during the iterative minimization procedure. When done right this allows for a quick descend along the gradient in the beginning and a slower, finer descend towards later iterations which leads to a more precise solution.

We implemented the following α adaptation algorithm, using an individual α for each weight:

1) check if the loss function value after each weight update is smaller than before
2) if it is smaller do not adapt the learning rate
3) if it is larger divide the learning rate by a set value

This approach proved to be independent of the initial value for alpha (assuming initial alpha was large enough) and led to a faster convergence than decreasing α linearly with iteration number.

Linear Regression - Our Algorithm

Our algorithm's structure in a very simple form:

Set initial learning rate α, weights, convergence criterion ε and type of loss function

Calculate initial regression function and loss function using initial weights

While not converged :

- **Update** weights using old weights
- **Calculate** regression function and loss function using updated weights
- **While** current loss function < previous loss function :
  - **Update** alpha, weights
  - **Calculate** regression function and loss function using updated weights

Return current weights

Linear Regression

On the scikit learn implementation of linear regression:

- SKLearn directly calculates a matrix equation for the derivative of the sum of squares equaling zero and use LU decomposition to solve the matrix inversion problem.
- This approach leads to a very stable and quick solution for the weights of the linear regression function.
  ->The solution should also be exact up to machine precision.
- Therefore their algorithm outperformed our iterative procedure in speed as well as precision.
- We tried out a few simple linear functions we sampled a few points of, were the scikit learn implementation led to the exact solution, while ours always had some deviation.
- We also tested this, by solving the same equation (using scipy.linalg.solve for the matrix inversion) and it also led to exact results for simple functions btu was not stable for real data sets.

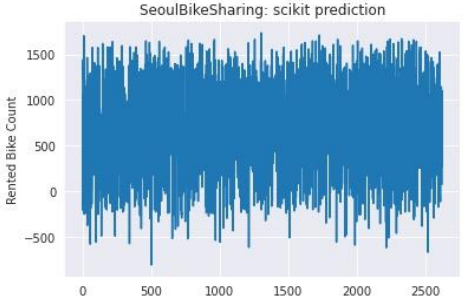
Linear Regression Model - Results Analysis

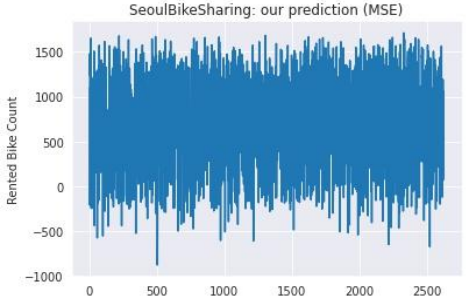
Results for Seoul Bike Sharing:

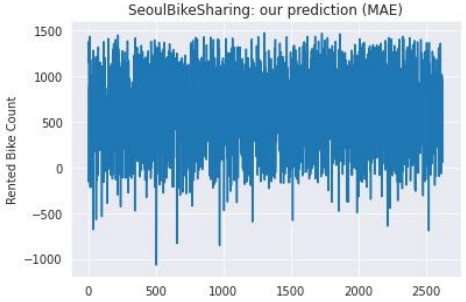
	Coefficient of determination	Mean Absolute Error	Mean Squared Error	Root Mean Squared Error	Convergence ε	iterations	Time (s)
SKLearn	0.56	322.3	190349	436.3	-	-	0.01
MSE	0.55	322.5	191860	438.0	1e-5	854	8.11
MAE	0.52	312.9	205879	453.7	1e-7	4624	49.8
RMSE	0.55	322.5	191860	438.02	1e-8	2470	37.57

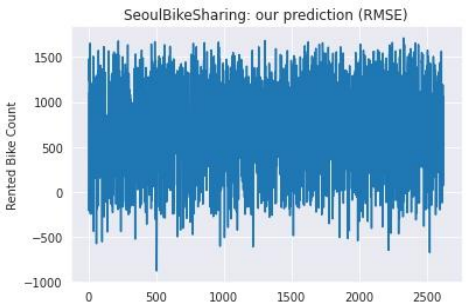
Linear Regression Model - Results Analysis

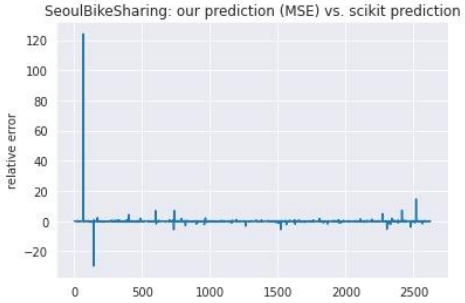
Results for Seoul Bike Sharing:

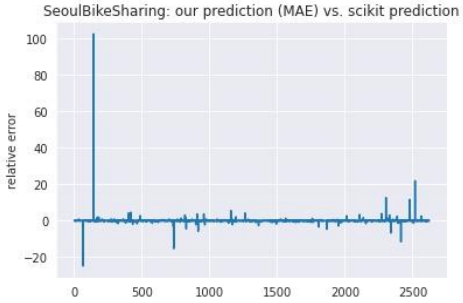


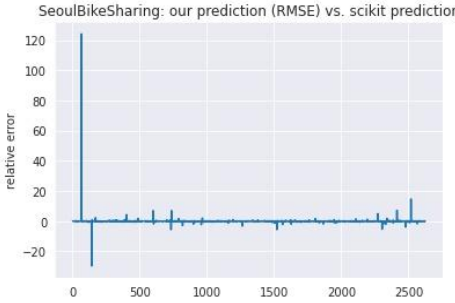












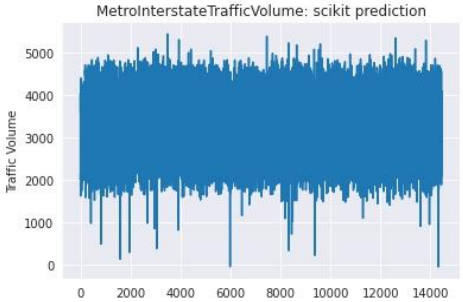
Linear Regression Model - Results Analysis

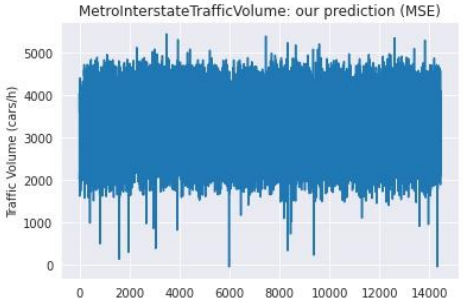
Results for Metro Interstate Traffic:

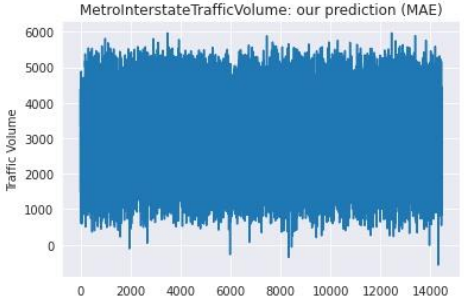
	Coefficient of determination	Mean Absolute Error	Mean Squared Error	Root Mean Squared Error	Convergence ε	iterations	Time (s)
SKLearn	0.15	1617.5	3368315	1835.3	-	-	0.22
MSE	0.15	1617.2	3368213	1835.3	1e-1	1435	716
MAE	0.1	1586.4	3574958	1890.8	1e-4	1947	719
RMSE	0.15	1617.2	3366728	1834.9	1e-4	2182	759

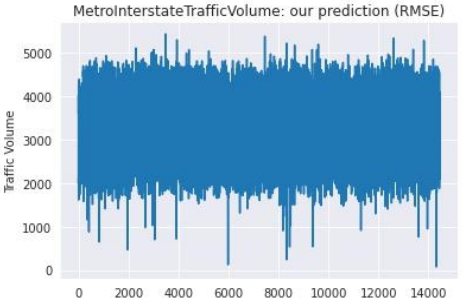
Linear Regression Model - Results Analysis

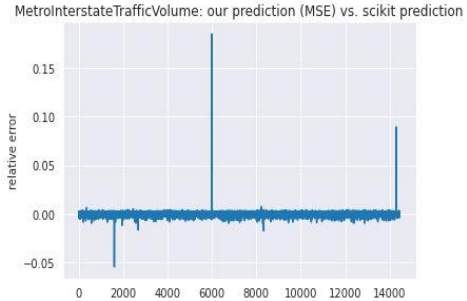
Results for Metro Interstate Traffic Volume:

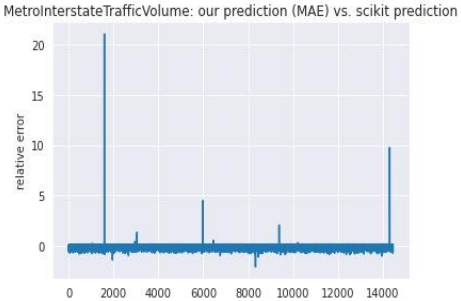


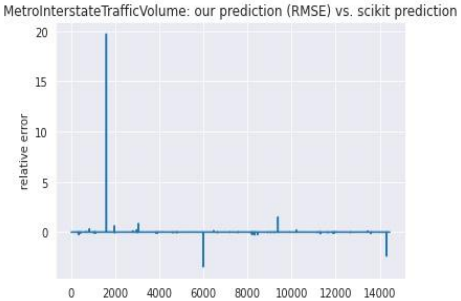












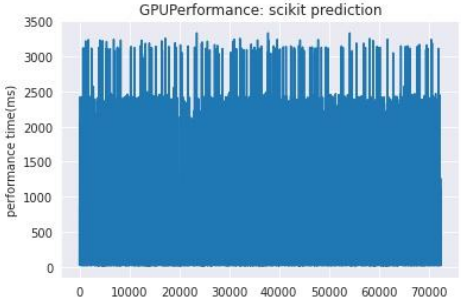
Linear Regression Model - Results Analysis

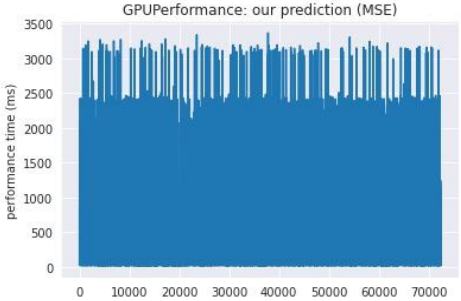
Results for GPU Performance:

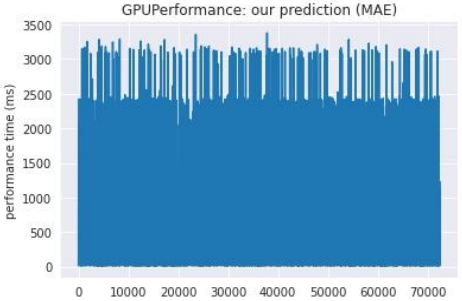
	Coefficient of determination	Mean Absolute Error	Mean Squared Error	Root Mean Squared Error	Convergence ε	iterations	Time (s)
SKLearn	1	7e-14	1e-26	1e-13	-	-	0.22
MSE	1	1.73	20.33	4.51	5e-3	5231	998
MAE	1	0.85	5.88	2.42	1e-5	4130	721
RMSE	1	1.03	5.67	2.38	1e-4	5789	1121

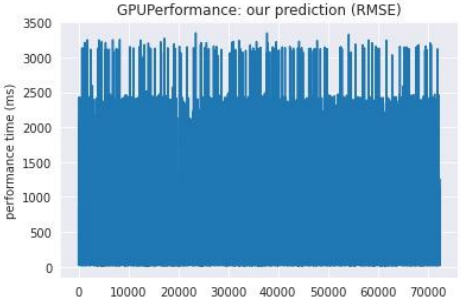
Linear Regression Model - Results Analysis

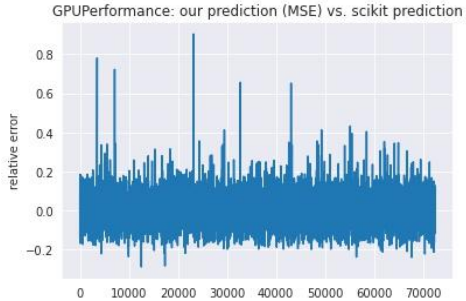
Results for GPU Performance:

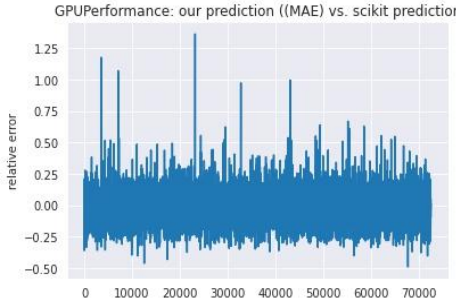


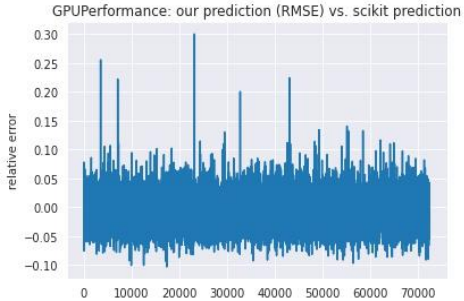












Linear Regression Model - Results Analysis

Comments and interesting observations I:

- It was shown that the results of the iterative gradient descent algorithm are comparable in quality to the ones obtained by the SKLearn approach of solving a matrix equation directly
- The GPU data set is the exception, where the SKLearn solution is basically an analytical solution with all errors being virtually zero.
- The SKLearn algorithm outperformed ours by orders of magnitude despite us using only numpy operations on whole arrays as opposed to using python loops to iterate over the individual elements of each vector y and x_n.
- We also compared this to the same approach using pandas instead of numpy, which led to a tenfold increase in computation time.
- Therefore we are overall satisfied with the performance of our algorithm despite the many possible optimizations that would have been possible still.
- It should also be noted that the precision of our algorithm can be increased arbitrarily by decreasing the convergence criterion ε value.
- However this also increases computation times arbitrarily.

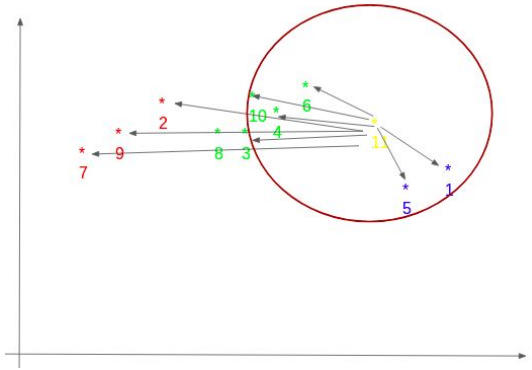
Linear Regression Model - Results Analysis

Comments and interesting observations II:

- The best loss function overally was the Mean Square Error, based on speed. Other metrics were nearly identically with the Root Mean Square Error.
- The Mean Absolute Error generally performed worse than the other two, except for the GPU Performance data set, were the MAE performed better and faster than the MSE.
- Interesting is that the GPU Performance data set is the only one were linear regression would have been an acceptable algorithm.
- For the Seoul Bike Sharing data set the predictions were mediocre at best and should not be used for predictions.
- For the Metro Interstate Traffic Volume data set linear regression performed very poorly.
- It should also be noted that we tried to choose the convergence criterion for each loss function in a way that the Coefficient of determination and sum of relative error were converging without having exceedingly long computation times.
- The computation times seemed to increase exponentially when the convergence criterion ε value was decreased after some point. This was generally also the point where the convergence criteria above were reached.

KNN - Algorithm

- The KNN algorithm uses '**feature similarity**' to predict the values of any new data points.
- This means that the new point receives a value based on the similarity with the points in the training set.
- In a regression problem, the average of the values is considered the final forecast.
- Steps of the algorithm:
    a. First, the distance between the new point and each training point is calculated.
    b. The nearest k data points are selected (based on distance).
    c. The average of these data points is the final forecast for the new point.



Example with two features considered (two dimensions)

KNN - Our Implementation

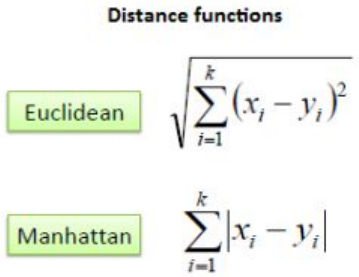
- Conversion of the *pandas dataframes* to *python dictionaries*, in order to achieve greater efficiency.
- Two training data options:
  - All Data (without the target); or
  - Use of the *X_train* & *y_train* from the *train_test_split*.
- Choice of algorithm parameters:
  - **mode:** KNeighbors; RadiusNeighbors;
  - **n_neighbours;**
  - **distance_function**: Euclidean Distance; Manhattan Distance;
  - **radius:** limit to the RadiusNeighbors (0 indicates no radius);
  - **label**: feature to predict;
  - **features**: collection of features to use on the distance function calculation.
- Feature selection using the regression correlation.

KNN - Our Implementation

- Algorithm initialization:
  - **Init (**data, label, features, mode=1, n_neighbours=5, distance_function=1, radius=0, path="")
    - Dictionary data or csv path;
- Execution of the algorithm (forecasting):
  - Get indexes from the test data;
  - For each index:
    - Get target in the dictionary;
    - **Run (**target):
      - Analysis of neighbors, collecting the nearest k elements in the distance calculation;
      - Calculation of the target label associated with neighbors.
    - Addition to the prediction's collection.

KNN - Execution *Run*

- Analysis - For each row in the data:
    1. Calculation of the distance to the target:
        a. **Euclidean** distance function:
            i. If only 1 feature: Absolute value of the difference between the row's feature and target's feature;
            ii. If more than 1 feature: Square root of the sum of squares of the differences between the row's features and target's features.
        b. **Manhattan** distance function:
            i. If only 1 feature: Absolute value of the difference between the row's feature and target's feature;
            ii. If more than 1 feature: Sum of the absolute values of the differences between the row's features and target's features.
    2. (...)



KNN - Execution *Run*

- Analysis - For each row in the data:
    1. (...)
    2. If mode == **KNeighbours**:
        a. Calculation of the **worst distance** in the actual best k-neighbourhood;
        b. If the distance of the actual row:
            i. Creation of a **new element** (with the distance and the row) into the **best k-neighbourhood**;
            ii. Removal of the **worst neighbour** from the best k-neighbourhood.
    3. If mode == **RadiusNeighbours**:
        a. If the distance of the actual row is **less than the established radius** parameter:
            i. Addiction of a **new element** (with the distance and the row) into the **best neighbourhood**.
- Calculation of the target label:
    1. **Mean between targets** of the elements included in the best neighbourhood.

KNN - Results

- **Seoul Bike Sharing:**
  - Dataset size: 8760 instances
  - Distance Function: Euclidean
  - Number of Neighbours: 5

Seoul Bike Sharing	**Attributes**	**Coefficient of determination**	**Mean Absolute Error**	**Mean Squared Error**	**Root Mean Squared Error**	**Time (s)**
SKLearn	14	0.52	307.41	193419.49	439.79	0.02
All Data	4 (Temperature, Hour, Dew point temperature, Winter)	0.68	222.50	129342.91	369.64	87.30
Train Test Split	4 (Temperature, Hour, Dew point temperature, Winter)	0.71	212.55	115539.99	339.91	57.21
Train Test Split	2 (Temperature and Hour)	0.57	265.26	174968.48	418.29	46.63
Train Test Split	1 (Temperature)	0.15	427.92	344168.06	586.66	34.45

KNN - Results

- **Metro Interstate Traffic Volume:**
  - Dataset size: 48193 instances
  - Distance Function: Euclidean
  - Number of Neighbours: 5
- **GPUPerformance:**
  - Dataset size: 241600 instances
  - **Not bearable**

Metro Interstate Traffic Volume	**Attributes**	**Coefficient of determination**	**Mean Absolute Error**	**Mean Squared Error**	**Root Mean Squared Error**	**Time (s)**
SKLearn	71	0.78	607.43	861800.68	928.33	0.54
Train Test Split	6 (Temperature, Clouds, Rain, Snow, Month, Hour)	0.77	619.74	908906.76	953.37	2558.52 = 42 min
Train Test Split	3 (Temperature, Clouds, Hour)	0.76	627.04	955923.09	977.71	1445.74 = 24 min
Train Test Split	2 (Temperature, Hour)	0.76	627.31	956639.09	978.07	1218.48 = 20 min
Train Test Split	1 (Hour)	0.73	629.65	1062031.86	921.49	1218.48 = 15 min

KNN - Results Analysis

- It is necessary to first measure **the correlation in the regression** in order to take into account the **features with the most influence** and obtain the best results, otherwise the execution would require even more time.
- It is **difficult to work with categorical variables** in this algorithm, and converting to binary is not very efficient. Since the neighbors are calculated through distances, the algorithm is more efficient working only with numerical values because they have more varied differences.
- Sometimes **the radius mode can be shown to be more efficient** because it is **difficult to measure which is the best *k*** considering that there may be clusters with very different shapes.
- **Our implementation** is mathematical-oriented, without the usual training/test method in the classifiers. Because of this, it requires more time and is unbearable for larger datasets. However, it **presents better results for smaller datasets** *(Sheoul Bike Sharing - 8760 instances)* in an acceptable time.

Exercise 2 - Regression

Machine Learning

2020/2021 | Technische Universität Wien

Grupo 06
Diogo Braga (E12007525)
Enrico Coluccia (E12005483)
Matthias Kiss (01218325)