MACHINE LEARNING

TASK 2 - REPORT

01.06.2022

PART A

To begin with, let us discuss the task.

“The company Tajne-Przez-Poufne has commissioned you to predict a certain experimentally determined value of Target (as the project is top secret, the company has not told you the technical specifications of the projects carried out). Informally, the company only informed you that it is able to influence the value of the Target to some extent -- by modifying the parameters Var_mass and Var_LT, as well as changing the form of the experiment.”

Our main task will be to build a regressor and demonstrate how it works.

1.Basic understanding of the data:

- It is probably some kind of physical experiment, a pendulum or something with movement (sine waves).
- Target value is probably some measure of the distance traveled by the object, or thrust.
- Most likely, the variables are environmental such as wind, temperature, humidity, coating, ball weight, cord length (on which the ball dangles), etc. The relationship between them is such that we have all the information that accompanied pushing the ball, apart from the force with which it was pushed. In addition, the company has influence over (so it probably manipulates in a controlled way) the variables Var_mass and Var_LT.

Index	Var av	Var LT	Var mass	Var1	Var2	Var3	Var4	Var5	Var6	Target
count	399	399	399	399	399	399	399	399	399	399
mean	0.456467	1496.5	1364.73	0.439441	0.481657	0.490752	0.466812	0.430353	0.386084	14.6789
std	0.141583	143.563	61.0903	0.182183	0.156515	0.155704	0.15858	0.161483	0.165331	2.57437
min	0.1027	1050	1251	0.038	0.092	0.093	0.049	0.079	0.015	4.47222
25%	0.36185	1400.5	1324	0.3135	0.376	0.3825	0.363	0.309	0.245	13.2222
50%	0.4553	1499	1363	0.432	0.48	0.495	0.475	0.437	0.384	15.5
75%	0.54255	1586.5	1410.5	0.543	0.566	0.58	0.5705	0.5365	0.5075	15.6944
max	1.3279	1914	1478	1.367	1.29	1.336	1.349	1.356	1.344	26.8611

Index	Var av	Var LT	Var mass	Var1	Var2	Var3	Var4	Var5	Var6	Target
Var_av	1	0.0360257	-0.07105	0.767126	0.893804	0.9291	0.938564	0.856735	0.801954	0.282291
Var_LT	0.0360257	1	-0.0296293	0.182257	0.179205	0.118434	-0.0309082	-0.135792	-0.130143	0.277022
Var_mass	-0.07105	-0.0296293	1	-0.0847183	-0.0863191	-0.081682	-0.0495147	-0.0249083	-0.0541654	-0.0125341
Var1	0.767126	0.182257	-0.0847183	1	0.903599	0.768655	0.554449	0.405036	0.453991	0.3474
Var2	0.893804	0.179205	-0.0863191	0.903599	1	0.898875	0.746419	0.567791	0.531047	0.350299
Var3	0.9291	0.118434	-0.081682	0.768655	0.898875	1	0.843562	0.668073	0.600753	0.250724
Var4	0.938564	-0.0309082	-0.0495147	0.554449	0.746419	0.843562	1	0.88905	0.769386	0.209796
Var5	0.856735	-0.135792	-0.0249083	0.405036	0.567791	0.668073	0.88905	1	0.898119	0.149467
Var6	0.801954	-0.130143	-0.0541654	0.453991	0.531047	0.600753	0.769386	0.898119	1	0.204699
Target	0.282291	0.277022	-0.0125341	0.3474	0.350299	0.250724	0.209796	0.149467	0.204699	1

It is immediately noticeable that two columns need to be normalized. These will be “Var_LT” and “Var_mass”. Also, using correlation matrix we can agree that using all variables to predict “Target” will be understandable.
2.Data cleaning: There is no need for cleaning the data (with a high probability) because we have to deal with physical experience.
3. Preparing data: We normlizied data

Index	Var av	Var LT	Var mass	Var1	Var2	Var3	Var4	Var5	Var6	Target
count	399	399	399	399	399	399	399	399	399	399
mean	0.456467	0.516778	0.50101	0.439441	0.481657	0.490752	0.466812	0.430353	0.386084	0.455884
std	0.141583	0.166161	0.26912	0.182183	0.156515	0.155704	0.15858	0.161483	0.165331	0.114984
min	0.1027	0	0	0.038	0.092	0.093	0.049	0.079	0.015	0
25%	0.36185	0.405671	0.321586	0.3135	0.376	0.3825	0.363	0.309	0.245	0.390819
50%	0.4553	0.519676	0.493392	0.432	0.48	0.495	0.475	0.437	0.384	0.492556
75%	0.54255	0.620949	0.702643	0.543	0.566	0.58	0.5705	0.5365	0.5075	0.501241
max	1.3279	1	1	1.367	1.29	1.336	1.349	1.356	1.344	1

and split it into train and test sets.

X_test	DataFrame	(80, 9)	Column
X_train	DataFrame	(319, 9)	Column
y_test	Series	(80,)	Series
y_train	Series	(319,)	Series

Later, when we decide to choose a method, we will also add a validation set to the training data to determine which parameters will work best for our model.

4.Testing various regressors: We will choose between implemented by hand Linear multiple regression (OLS), LinearRegression built in sklearn, OLS built in scipy and ANN built in keras/TensorFlow.
Note. For model validation, we will use measures such as:

- Adjusted R^2 whis is a corrected goodness-of-fit (model accuracy) measure for linear models.

$$R^2_{adj} = 1 - \left[\frac{\left(1 - R^2\right)(n-1)}{n-k-1} \right]$$

- MAE (mean absolute error)

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n}.$$

- MAPE (mean absolute percentage error)

$$\mathrm{MAPE} = \frac{100\%}{n} \sum_{t=1}^{n} \left| \frac{A_t - F_t}{A_t} \right|$$

- MSE (mean square error)

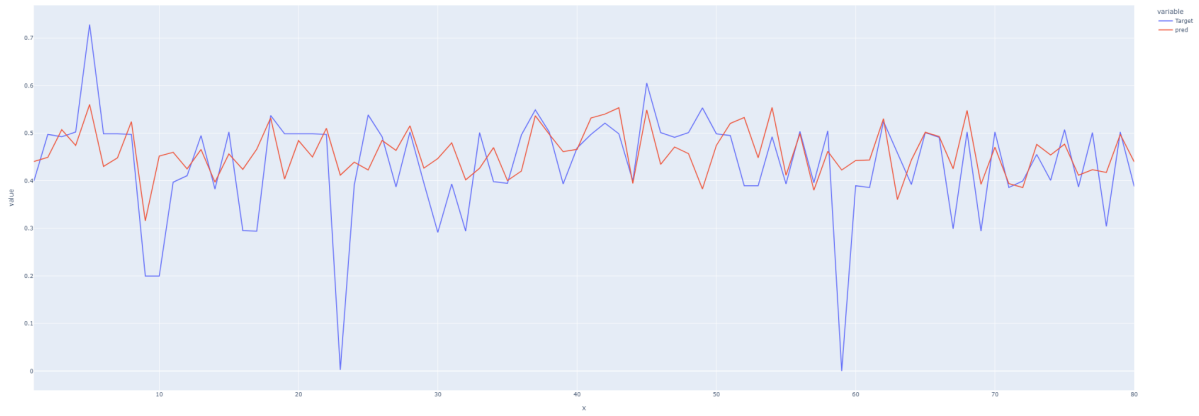
$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

- AIC (Aike information criterion) which is an (best) estimator of prediction error and thereby relative quality of statistical models for a given set of data.

$$\mathrm{AIC} \ = \ 2k - 2\ln(\hat{L})$$

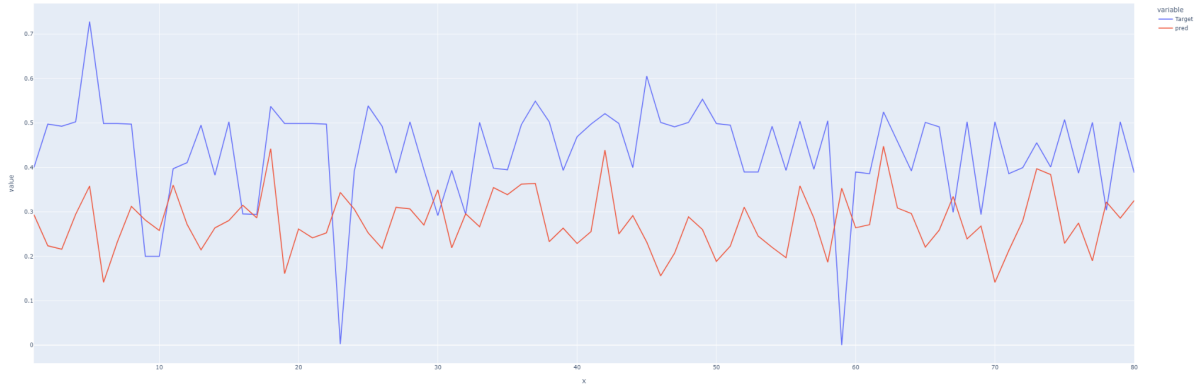
We will focus primarily on the measurement of AIC (the smaller the better).

Linear multiple regression (OLS by hand)



SSE: 0.766200725939506
SSR: 0.24685110459317428
SST: 1.0130518305326803
R^2: 0.2436707551906556
R^2 adjusted: 0.15845055859241952
MAE: 0.06356744865735994
MAPE: 6.876637311090468
MSE: 0.009577509074243824
AIC: -351.8670187148473

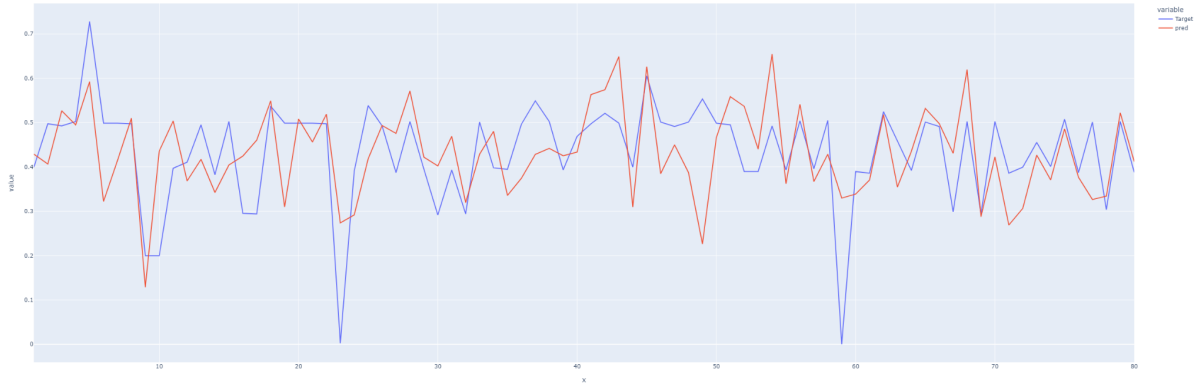
LinearRegression from sklearn



SSE: 3.5544536881930853
SSR: 2.3435568519814076

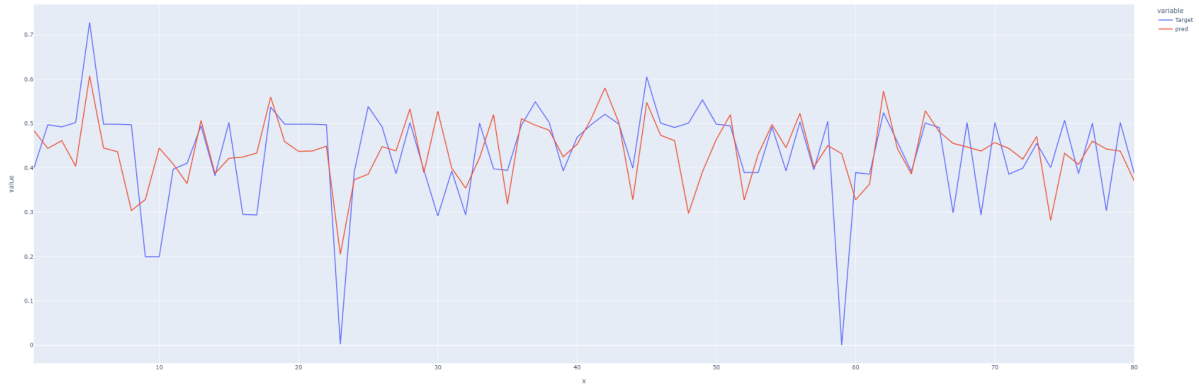
SST: 5.898010540174493
R^2: 0.3973470098125788
R^2 adjusted: 0.32944244753793983
MAE: 0.18261900890501587
MAPE: 5.992398594734831
MSE: 0.04443067110241356
AIC: -229.10602059480465

OLS from scipy.stats



SSE: 0.8795504191504383
SSR: 0.7834472242201771
SST: 1.6629976433706153
R^2: 0.4711054326164058
R^2 adjusted: 0.4115116785450148
MAE: 0.07953817141185435
MAPE: 5.253673764715712
MSE: 0.010994380239380477
AIC: -340.8296819248169

ANN regression



SSE: 0.7636233026232983
SSR: 0.4051587973855373
SST: 1.1687821000088356
R^2: 0.34665041275227815
R^2 adjusted: 0.2730335578511265

MAE: 0.06830103593410526
MAPE: 14.370759963989258
MSE: 0.009545291282791227
AIC: -352.1365844381472

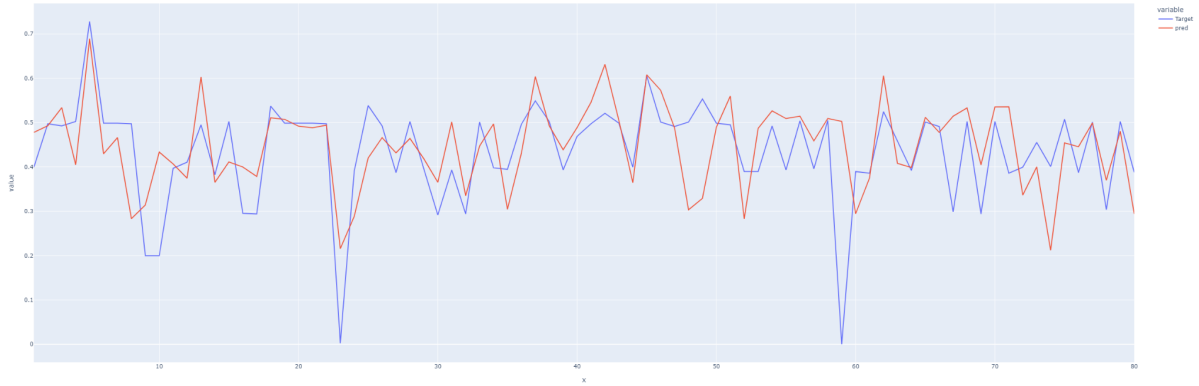
We can easily see that linear regression (sklearn) is significantly underestimated. Multiple regression (by hand) behaves decently. Therefore, the choice will be between the OLS model and the ANN model. Although OLS performs similarly, the ANN is slightly better at capturing outliers and has overall better performance.

5. Training ANN: We will now test different activation functions, optimizer, number of epochs and batch sizes, for our model, to achieve the best possible results.
First, we will further split the data into training, validation and test sets. Now, based on the training and validation sets, with a fixed number of epochs = 100, we will compare performance using measures.

Index	Activation function	Optimizer	Batch size	MAE	MAPE	MSE	AIC
0	linear	Adam	1	0.0688089	0.209676	0.0118959	-334.525
1	linear	Adam	5	0.0740062	0.204413	0.0122939	-331.892
2	linear	Adam	10	0.0725759	0.203924	0.0120633	-333.407
3	linear	Adam	15	0.0697698	0.210136	0.0118598	-334.768
4	linear	Adam	20	0.076663	0.206723	0.0126869	-329.375
5	relu	Adam	1	0.0704509	0.212827	0.0117944	-335.21
6	relu	Adam	5	0.0681356	0.200185	0.0109009	-341.513
7	relu	Adam	10	0.0657031	0.200665	0.0102364	-346.544
8	relu	Adam	15	0.0638738	0.193952	0.0101345	-347.345
9	relu	Adam	20	0.0628812	0.193871	0.00992126	-349.046
10	linear	SGD	1	0.0698826	0.198351	0.0116323	-336.317
11	linear	SGD	5	0.0667734	0.196788	0.0114564	-337.537
12	linear	SGD	10	0.0756173	0.230346	0.0124416	-330.937
13	linear	SGD	15	0.0727314	0.198665	0.0120814	-333.287
14	linear	SGD	20	0.069211	0.200369	0.0115598	-336.818
15	relu	SGD	1	0.0645566	0.194876	0.0102307	-346.589
16	relu	SGD	5	0.0692835	0.199463	0.0117468	-335.534
17	relu	SGD	10	0.0695623	0.198947	0.0123468	-331.548
18	relu	SGD	15	0.0708972	0.200526	0.0128692	-328.234
19	relu	SGD	20	0.0660944	0.18688	0.011716	-335.744
20	linear	<keras.optimizers.optimizer_v2.gradient_des…	1	0.0741051	0.201306	0.0124783	-330.701
21	linear	<keras.optimizers.optimizer_v2.gradient_des…	5	0.0831254	0.214608	0.0136012	-323.808
22	linear	<keras.optimizers.optimizer_v2.gradient_des…	10	0.0678353	0.203658	0.0116439	-336.238
23	linear	<keras.optimizers.optimizer_v2.gradient_des…	15	0.0818466	0.212596	0.0133148	-325.51
24	linear	<keras.optimizers.optimizer_v2.gradient_des…	20	0.0721226	0.198474	0.0121332	-332.945
25	relu	<keras.optimizers.optimizer_v2.gradient_des…	1	0.0700097	0.21983	0.0111535	-339.68
26	relu	<keras.optimizers.optimizer_v2.gradient_des…	5	0.0745373	0.201602	0.0119994	-333.832
27	relu	<keras.optimizers.optimizer_v2.gradient_des…	10	0.0669806	0.192495	0.0119233	-334.341
28	relu	<keras.optimizers.optimizer_v2.gradient_des…	15	0.0672864	0.196221	0.0114313	-337.712
29	relu	<keras.optimizers.optimizer_v2.gradient_des…	20	0.0682099	0.193172	0.0120545	-333.465

Our pick will be activation function = “relu” with optimizer = ‘Adam’ and batch size = 5. Therefore, using the training set, we will train our model and visualize the results from the test data.

Note. The model's overtraining leveling function (EarlyStopping()) pauses no further than the first 10 epochs. Hence, we will teach our model in no more than 250 epochs.



SSE:	0.8741802567836199
SSR:	0.7356550403703723
SST:	1.6098352971539922
R^2:	0.4569753450374257
R^2 adjusted:	0.3957894684219243
MAE:	0.07219897865941312
MAPE:	10.65102481842041
MSE:	0.010927253209795249
AIC:	-341.31962526449774

Summary: In our deliberations, the ANN model proved to be the most effective. Not far behind it is the OLS model from the scipy package. For small amounts of data, the OLS model will be more convenient, while for increasing amounts of data, and thus increasing amounts of information to learn the model, it will be more optimal to reach for ANN. In the case of a hand-written function, i.e. multiple regression, it behaves decently. As for the built-in linear regression function from the sklearn package, it is far behind.

Another linear regression model that we applied to the data is Elastic Net but it was not as efficient as previous models. This is a combination of a Ridge and Lasso regressions, because it uses penalties from both these models. In our model there are multiple attributes which are highly correlated to each other. If we applied Lasso regression, the model would choose just one of the correlated attributes but in the Elastic Net model all of them will be included.

Before we apply the Elastic Net model, we split our dataset to training and test sets and we also standardize our data using StandardScaller(). Then we use ElasticNet() function to create our regresor and predict values for our testing data.

We will select different values of parameters ‘alpha’ and ‘l1_ratio’ which control ridge and lasso regressions penalties. We would like to see which combination of these parameters

will be the most effective. Measures that we will use to define the effectiveness of the model will be:

- standard deviation of residuals,
- convergence coefficient.

alpha	**l1_ratio**	**standard deviation of residuals**	**convergence coefficient**
default (1)	default (0,5)	2,5661	0,9350
0,2	0	2,5961	0,7119
0,2	0,2	2,5982	0,7225
0,2	0,6	2,5829	0,7502
0,2	1	2,5738	0,7755
0,5	0	2,5958	0,7564
0,5	0,2	2,5772	0,7907
0,5	0,6	2,5581	0,8535
0,5	1	2,5781	0,9016
0,8	0	2,5931	0,7880
0,8	0,2	2,5622	0,8416
0,8	0,6	2,5655	0,9217
0,8	1	2,5625	0,9738

The best two settings are the following:

- alpha = 0,2, l1_ratio = 0 - Ridge regression penalty
- alpha = 0,8, l1_ratio = 0,2

but the differences in measures of effectiveness are really small, thus we can conclude that the Elastic Net regression model can be applied with any combination of parameters.

Coefficients that we obtained respectively are:

- 0.21581793, 0.37503251, -0.00282672, 0.46498216, 0.39332233, -0.22264118, 0.13735007, 0.01602662, and we have a warning that objective did not converge,
- 0.14532329, 0.19363816, 0, 0.29939877, 0.26303077, 0, 0.04655672, 0

PART B

In the second part of this task, data which will be classified describe telephones' parameters, such as battery power, clock speed, five g, etc. These telephones are being sold in Anonidas' shop. He would like to have a classifier which will help him define which price range the telephone belongs to.

We will use a random forest classifier to build such a model. This classifier is based on decision trees. It generates uncorrelated decision trees which are really diverse. It is ensured by bagging and future randomness.

Bagging (Bootstrap Aggregation) means that each individual tree randomly selects a sample from the dataset with replacement. It means that if we have a dataset of size n, then the training set for an individual tree is also of size n but instead of the original data we have a sample selected with replacement, so some of the records can be repeated in the training set.

Whereas future randomness means that there is only a random subset of features available from which a condition to split a node is chosen. In a standard decision tree we can choose the most separating future from all of them, but in this model we can choose only from the random subset of all futures.

Now, we are going to describe how a random forest classifier predicts the class for a specific observation. For all trees in the random forest, the class is being predicted as in the standard decision tree. Then, we choose the model's predicted class as the class which appeared in trees' predictions most often.

Before we use the random forest classifier we need to preprocess our data. We standardize all the futures, then all of them will have 0 mean and variation of 1. Because no data is missing in this dataset, we do not have to apply any methods of data completion.

We visualize the result of prediction using a confusion matrix C, which shows in which class the observation was predicted to be and which class it really belongs to. On the place $c_{i,j}$ there is a number of observations that are known to be in class i but were predicted to be in class j. We also use an accuracy score to see how many observations were classified correctly.

In this case only 51 observations were classified incorrectly and 349 were classified to the proper class. The accuracy score is 0.8725 which is quite high and we conclude that this method of classification is effective.

We will apply different values of some parameters to test which combination would be the best classifier for this data. 'n_estimator' parameter determines how many random trees will be generated and 'min_samples_split' defines how many samples have to be in the node to be split it:

n_estimators	**min_samples_split**	**number of samples classified correctly**	**accuracy score**
30	2	349	0,8725
30	5	345	0,8625
30	10	350	0,8750

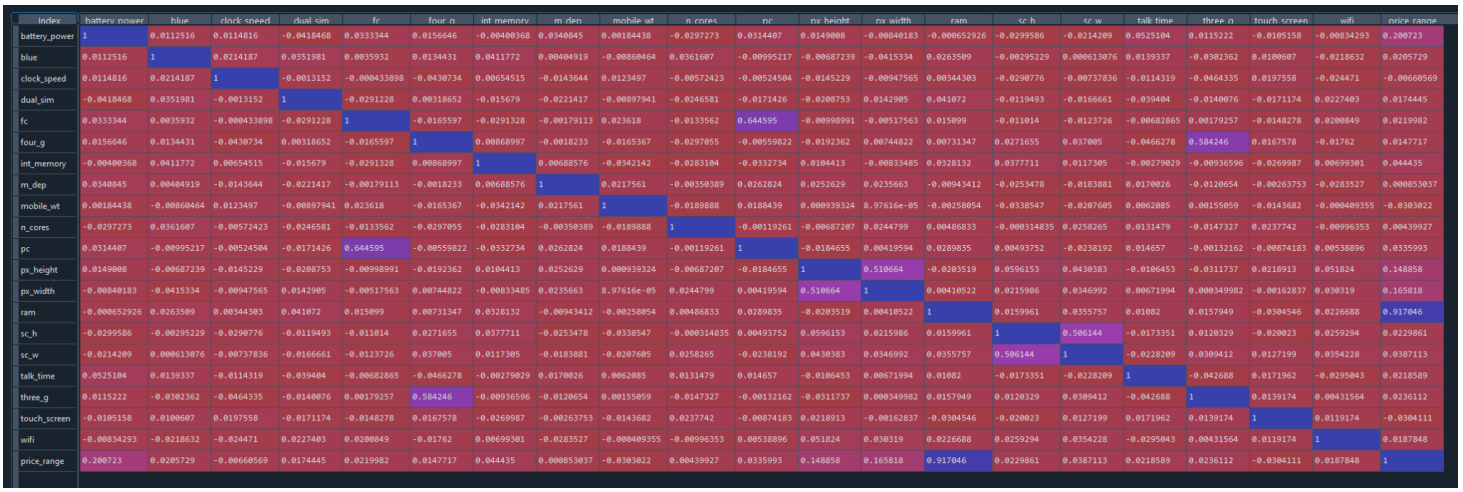
60	2	348	0,8700
60	5	348	0,8700
60	10	347	0,8675
100	2	349	0,8725
100	5	354	0,8850
100	10	354	0,8850

We can clearly conclude that the best parameters' settings are the following:

- n_estimators = 100, min_samples_split = 5,
- n_estimators = 100, min_samples_split = 10,

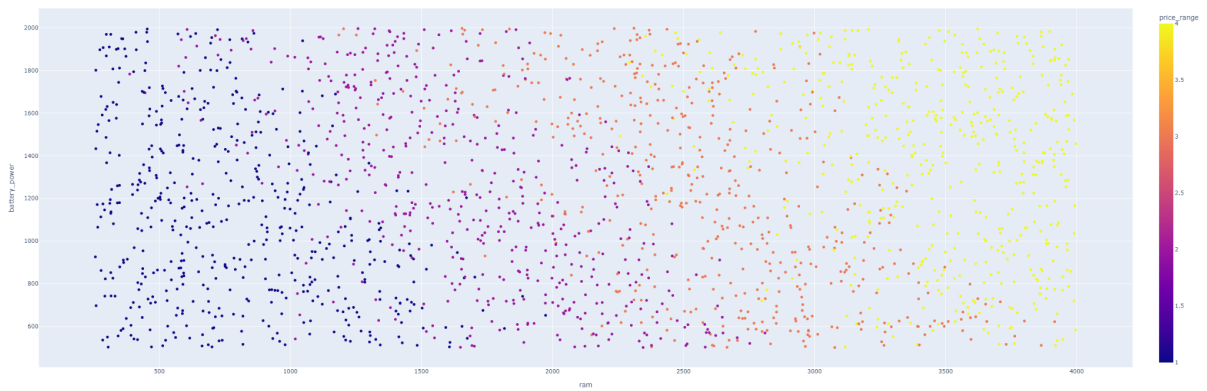
because they result in the best accuracy score and the largest number of samples which are classified correctly.

Now lets take a look at the Support Vector Machine classification method. We will use build in method from the sklearn package. Before we begin, we will take quick peek a correlation matrix:



index	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	pc	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	touch_screen	wifi	price_range
battery_power	1	0.0112516	0.0114816	-0.0418468	0.0333344	0.0156646	-0.00400368	0.0340845	0.00184438	-0.0297273	0.0314407	0.0149008	-0.00840183	-0.000652926	-0.0299586	-0.0214209	0.0525104	0.0115222	-0.0105158	-0.00834293	0.200723
blue	0.0112516	1	0.0214187	0.0351981	0.0035932	0.0134431	0.0411772	0.00404919	-0.00860464	0.0361607	-0.00995217	-0.00687239	-0.0415334	0.0263509	-0.00295229	0.000613076	0.0139337	-0.0302362	0.0100607	-0.0218632	0.0205729
clock_speed	0.0114816	0.0214187	1	-0.0013152	-0.000433898	-0.0430734	0.00654515	-0.0143644	0.0123497	-0.00572423	-0.00524504	-0.0145229	-0.00947565	0.00344303	-0.0290776	-0.00737836	-0.0114319	-0.0464335	0.0197558	-0.024471	-0.00660569
dual_sim	-0.0418468	0.0351981	-0.0013152	1	-0.0291228	0.00318652	-0.015679	-0.0221417	-0.00897941	-0.0246581	-0.0171426	-0.0208753	0.0142905	0.041072	-0.0119493	-0.0166661	-0.039404	-0.0140076	-0.0171174	0.0227403	0.0174445
fc	0.0333344	0.0035932	-0.000433898	-0.0291228	1	-0.0165597	-0.0291328	-0.00179113	0.023618	-0.0133562	0.644595	-0.00998991	-0.00517563	0.015099	-0.011014	-0.0123726	-0.00682865	0.00179257	-0.0148278	0.0200849	0.0219982
four_g	0.0156646	0.0134431	-0.0430734	0.00318652	-0.0165597	1	0.00868997	-0.0018233	-0.0165367	-0.0297055	-0.00559822	-0.0192362	0.00744822	0.00731347	0.0271655	0.037005	-0.0466278	0.584246	0.0167578	-0.01762	0.0147717
int_memory	-0.00400368	0.0411772	0.00654515	-0.015679	-0.0291328	0.00868997	1	0.00688576	-0.0342142	-0.0283104	-0.0332734	0.0104413	-0.00833485	0.0328132	0.0377711	0.0117305	-0.00279029	-0.00936596	-0.0269987	0.00699301	0.044435
m_dep	0.0340845	0.00404919	-0.0143644	-0.0221417	-0.00179113	-0.0018233	0.00688576	1	0.0217561	-0.00350389	0.0262824	0.0252629	0.0235663	-0.00943412	-0.0253478	-0.0183881	0.0170026	-0.0120654	-0.00263753	-0.0283527	0.000853037
mobile_wt	0.00184438	-0.00860464	0.0123497	-0.00897941	0.023618	-0.0165367	-0.0342142	0.0217561	1	-0.0189888	0.0188439	0.000939324	8.97616e-05	-0.00258054	-0.0338547	-0.0207605	0.0062085	0.00155059	-0.0143682	-0.000409355	-0.0303022
n_cores	-0.0297273	0.0361607	-0.00572423	-0.0246581	-0.0133562	-0.0297055	-0.0283104	-0.00350389	-0.0189888	1	-0.00119261	-0.00687207	0.0244799	0.00486833	-0.000314835	0.0258265	0.0131479	-0.0147327	0.0237742	-0.00996353	0.00439927
pc	0.0314407	-0.00995217	-0.00524504	-0.0171426	0.644595	-0.00559822	-0.0332734	0.0262824	0.0188439	-0.00119261	1	-0.0184655	0.00419594	0.0289835	0.00493752	-0.0238192	0.014657	-0.00132162	-0.00874183	0.00538896	0.0335993
px_height	0.0149008	-0.00687239	-0.0145229	-0.0208753	-0.00998991	-0.0192362	0.0104413	0.0252629	0.000939324	-0.00687207	-0.0184655	1	0.510664	-0.0203519	0.0596153	0.0430383	-0.0106453	-0.0311737	0.0218913	0.051824	0.148858
px_width	-0.00840183	-0.0415334	-0.00947565	0.0142905	-0.00517563	0.00744822	-0.00833485	0.0235663	8.97616e-05	0.0244799	0.00419594	0.510664	1	0.00410522	0.0215986	0.0346992	0.00671994	0.000349982	-0.00162837	0.030319	0.165818
ram	-0.000652926	0.0263509	0.00344303	0.041072	0.015099	0.00731347	0.0328132	-0.00943412	-0.00258054	0.00486833	0.0289835	-0.0203519	0.00410522	1	0.0159961	0.0355757	0.01082	0.0157949	-0.0304546	0.0226688	0.917046
sc_h	-0.0299586	-0.00295229	-0.0290776	-0.0119493	-0.011014	0.0271655	0.0377711	-0.0253478	-0.0338547	-0.000314835	0.00493752	0.0596153	0.0215986	0.0159961	1	0.506144	-0.0173351	0.0120329	-0.020023	0.0259294	0.0229861
sc_w	-0.0214209	0.000613076	-0.00737836	-0.0166661	-0.0123726	0.037005	0.0117305	-0.0183881	-0.0207605	0.0258265	-0.0238192	0.0430383	0.0346992	0.0355757	0.506144	1	-0.0228209	0.0309412	0.0127199	0.0354228	0.0387113
talk_time	0.0525104	0.0139337	-0.0114319	-0.039404	-0.00682865	-0.0466278	-0.00279029	0.0170026	0.0062085	0.0131479	0.014657	-0.0106453	0.00671994	0.01082	-0.0173351	-0.0228209	1	-0.042688	0.0171962	-0.0295043	0.0218589
three_g	0.0115222	-0.0302362	-0.0464335	-0.0140076	0.00179257	0.584246	-0.00936596	-0.0120654	0.00155059	-0.0147327	-0.00132162	-0.0311737	0.000349982	0.0157949	0.0120329	0.0309412	-0.042688	1	0.0139174	0.00431564	0.0236112
touch_screen	-0.0105158	0.0100607	0.0197558	-0.0171174	-0.0148278	0.0167578	-0.0269987	-0.00263753	-0.0143682	0.0237742	-0.00874183	0.0218913	-0.00162837	-0.0304546	-0.020023	0.0127199	0.0171962	0.0139174	1	0.0119174	-0.0304111
wifi	-0.00834293	-0.0218632	-0.024471	0.0227403	0.0200849	-0.01762	0.00699301	-0.0283527	-0.000409355	-0.00996353	0.00538896	0.051824	0.030319	0.0226688	0.0259294	0.0354228	-0.0295043	0.00431564	0.0119174	1	0.0187848
price_range	0.200723	0.0205729	-0.00660569	0.0174445	0.0219982	0.0147717	0.044435	0.000853037	-0.0303022	0.00439927	0.0335993	0.148858	0.165818	0.917046	0.0229861	0.0387113	0.0218589	0.0236112	-0.0304111	0.0187848	1

It is immediately apparent that RAM is extremely important for classification. Additionally we can add: battery_power, px_height and px_width.
Plotting RAM against battery power, clearly show possibility of linear separation i.e.



Moving on to the selection of parameters, we will pick between: radial basis function (rbf), linear function (linear) and polynomial function (poly) as a kernel in our clasificator and “one-versus-one” method. Lets compare classification reports for these functions.

(rbf,ovo)

	precision	recall	f1-score	support
1	0.92	0.95	0.94	88
2	0.93	0.92	0.92	95
3	0.88	0.95	0.92	109
4	0.98	0.88	0.93	108
accuracy			0.93	400
macro avg	0.93	0.93	0.93	400
weighted avg	0.93	0.93	0.93	400

(linear,ovo)

	precision	recall	f1-score	support
1	0.94	0.94	0.94	88
2	0.91	0.93	0.92	95
3	0.92	0.93	0.92	109
4	0.96	0.94	0.95	108
accuracy			0.93	400
macro avg	0.93	0.93	0.93	400
weighted avg	0.93	0.93	0.93	400

(poly,ovo)

	precision	recall	f1-score	support
1	0.96	0.98	0.97	88
2	0.97	0.94	0.95	95
3	0.91	0.96	0.93	109
4	0.97	0.92	0.94	108
accuracy			0.95	400
macro avg	0.95	0.95	0.95	400
weighted avg	0.95	0.95	0.95	400

All three of them give decent results. So we need to ask ourselves a question. On which measure should we focus most? Well, we want to minimize losses and maximize profits by allocating phones to the right price shelves. Therefore, we will mainly minimize recalls for categories 3 and 4, as it can be more costly to undercut them, with accuracy in mind.
In this case, the polynomial kernel works best.
Lets run our learned classifier on test data.

```
Confusion matrix
 [[99  1  0  0]
 [ 5 93  3  0]
 [ 0  5 90  1]
 [ 0  0  5 98]]

              precision    recall  f1-score   support

           1       0.95      0.99      0.97       100
           2       0.94      0.92      0.93       101
           3       0.92      0.94      0.93        96
           4       0.99      0.95      0.97       103

    accuracy                           0.95       400
   macro avg       0.95      0.95      0.95       400
weighted avg       0.95      0.95      0.95       400
```

In summary, a very high recall was achieved for categories 3 and 4, with errors more in favor of the seller (single inflated categories) and accuracy at 95%. Thus, it opens up the possibility of possible promotions or sales in the absence of interest without incurring losses.