Mandatory Assignment 2

STK-IN4300

Simen Løken

November 2022

Problem 1 Regression

1)

We dichotomize and use a 1/3 2/3 `test/train` split, plotting the observed `LC50` against the predicted `LC50`:

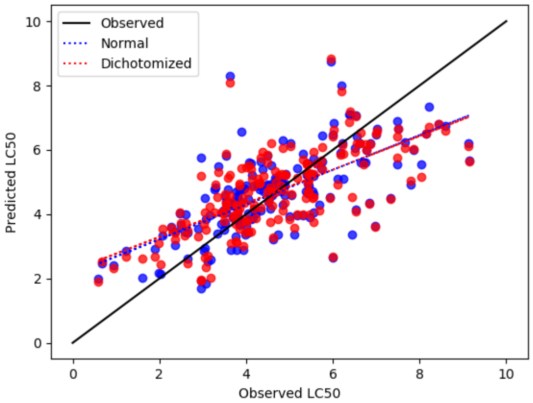

Figure 1: A scatter plot of our predicted values on the y-axis against their observed values on the x-axis. Ideally any values should lie alongside the "observed" or nominal line in black.

As we can see, an undichotomized dataset outperforms its dichotomized counterpart just barely (the blue dashed line is close to the nominal black than the red).
As for the test error, we get:

$$\texttt{MAE} = 0.9172$$

$$\texttt{MAE}_d = 0.9366$$

where the $_d$ signifies the dichotomized error. This tells us that what we already know from Figure [1], namely that our non-dichotomized model performs better than its counterpart.

Additionally, we retrieve the following regression coefficients:

	Normal	Dichotomized
TPSA	1.3635	1.1205
SAacc	-1.0259	-0.8218
H050	0.0694	-0.0734
MLOGP	0.7921	0.8614
RDCHI	0.4530	0.3298
GATS1p	-0.2054	-0.1796
nN	-0.4049	0.0116
C040	-0.0083	-0.0745

Table 1: A table showing our regression coefficients

We see here that, generally, the regression coefficients for the dichotomized model are weaker than their normal counterparts, which is likely what causes the worse fit.

2)

We get:

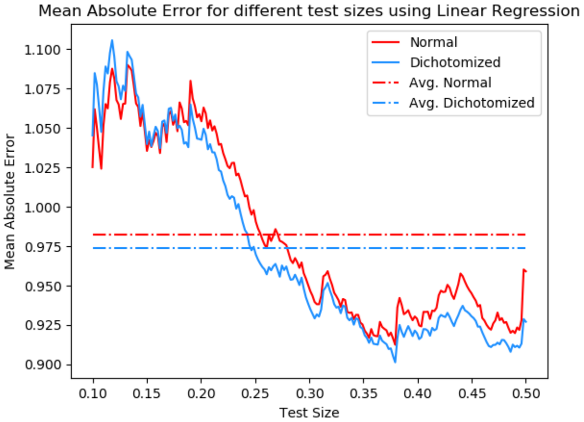

Figure 2: A figure showing the Mean Absolute Error (MAE) for different test sizes of our normal and dichotomized model, plus their average.

which is actually the opposite of what I (and seemingly the sub problem) was expecting. As to why this is the case, I cannot say, it could simply come down to ”bad luck” given the random nature of fitting models, but I can explain why a model is *usually* worsened by dichotomizing variables.
The most obvious one is losing ”resolution”, and by extension information whenever we dichotomize an array. What we're doing, in layman's terms, is postulating that to the model it should only matter *if* there are any hydrogen atoms bonded to heteroatoms, nitrogen in the molocule etc... The amount is irrelevant.
Given that the numbers we've dichotomized for some cases reach all the way up into the 10s, it is hard to justify dichotomizing our data. Perhaps quantizing would be better, at given thresholds.
As it stands now, the result in Figure [2] tells us that the number of the respective atoms do not matter, it only matters that the respective atoms are present.

3)

We examine the the model using stopping criterion and feature selection, and look at R^2 scores. We get 4 features selected, the same for backward and forward, and those are: `TPSA, SAacc, MLOGP, GATS1p`. This is, from what I understand, rather uncommon, and only happens after tuning the alpha hyperparameter of `Lasso`, which was necessary in this case as not tuning it resulted in negative R^2 values for feature selected fits.
We get the following R^2 values:

$$R^2_{AIC} = 0.452 \quad R^2_{BIC} = 0.459$$
$$R^2_F = R^2_B = 0.447$$

R^2_F and R^2_B are of course equal because both forward and backward selection chose the same four variables.

4)

We choose to look at the alpha regularization hyperparameter of Ridge Regression. We let it go from `0.1` to `12`, giving us the following result:

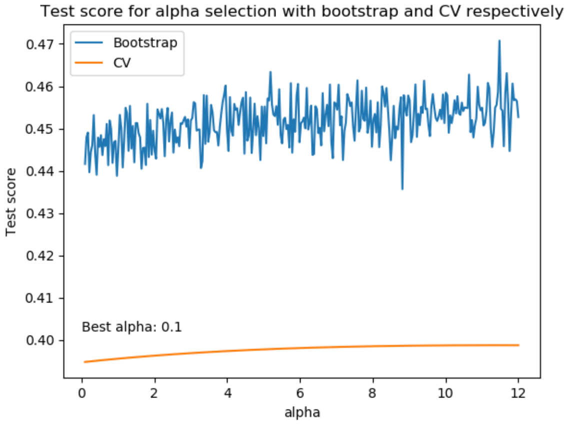

Figure 3: A figure showing how test score varies with alpha selection
It's worth mentioning that this is not using deviance as requested in the assignment text, as I couldn't get a deviance scorer to work for both methods.

The most interesting result here is the variance in the smoothness of the test scores. Whereas the test score for the bootstrap method is strictly better than cross-validation, cross-validation is a lot smoother and more stable. This makes sense when you think about the way cross validation works and how we find the mean of x-folds. You're less likely to have dramatic "jumps" in test scores.

5)

We use three different smoothing splines in:

Spline	MAE
SAacc, MLOGP	0.89
nN, H050	1.12
GATS1p, SAacc	0.97

Table 2: Mean Absolute Error for three different splines using a GAM approach.
Refer to the assignment text [1] for spline variable definitions

We see that we get less than satisfactory results, with mean absolute errors around 1. Additionally, we see that for low variance, low resolution variables like the nN, H050 spline, we get poorer results than with the other splines. This lends further credence to the features selected by forward selection/backward elimination in task 3, in that these variables are less statistically important than the others. You could also argue this could be one of the reasons as to why we saw better performance with these variables dichotomized in 2).

6)

We get the following tree:

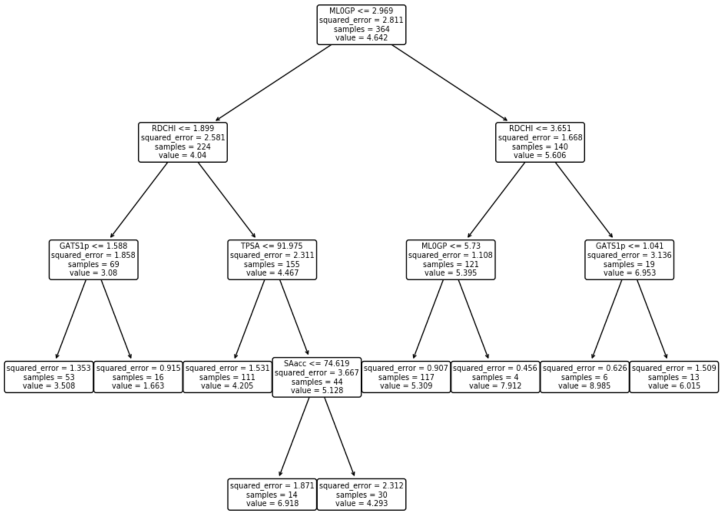

Figure 4: Our tree. Modeled such that, for example at the "root" node, if MLOGP ≤ 2.969, go left, if not go right

As for how we decided this model, we use a tuning parameter alpha which controls how much of the tree we prune to prevent overfitting the model. We simply use the cost_complexity_pruning_path method in scikit-learn to get a range of possible alpha parameters, before finding the one that provides the best test score.

Additionally, we can examine how the tree is pruned as we increase alpha:

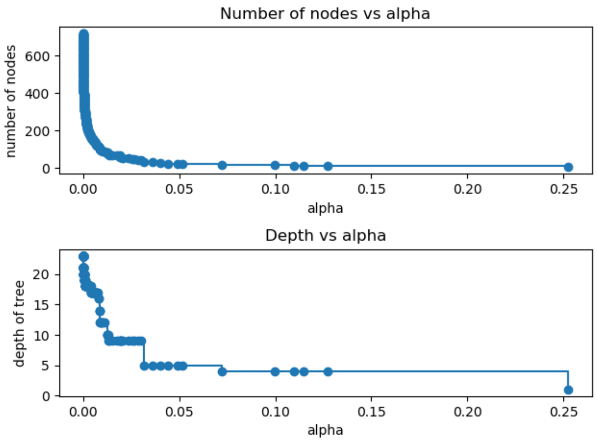

Figure 5: Showing how the depth and total number of nodes of the tree decreases as we increase alpha. In our case the sweetspot is at just over 0.05

7)

Lastly, let us look at the test and training error of all our methods:

	Train Error [MAE]	Test Error [MAE]
1)		
Normal	0.92	0.90
Dichotomized	0.94	0.94
3)		
AIC	0.2	0.81
BIC	0.22	0.79
FORWARD	0.40	0.66
BACKWARD	0.40	0.66
4)		
BOOTSTRAP	0.92	0.93
CV	0.91	0.95
5)		
GAM		0.89
6)		
TREE	0.94	0.85

Table 3: All test and train errors for our models in Mean Absolute Error.
Note that the training error for GAM is missing as I couldn't find a way to retrieve it.

We see that most of the models are pretty bad, most likely because of lack of tuning. However, the greatest performer (by a large margin) is the forward selection/backward elimination solution. This is likely because forward selection/backward elimination is a ”parameter” in itself, that is, this model is not ”raw” in the same sense that the other models are.

Problem 2 Classification

1)

We run the code and find the following plot:

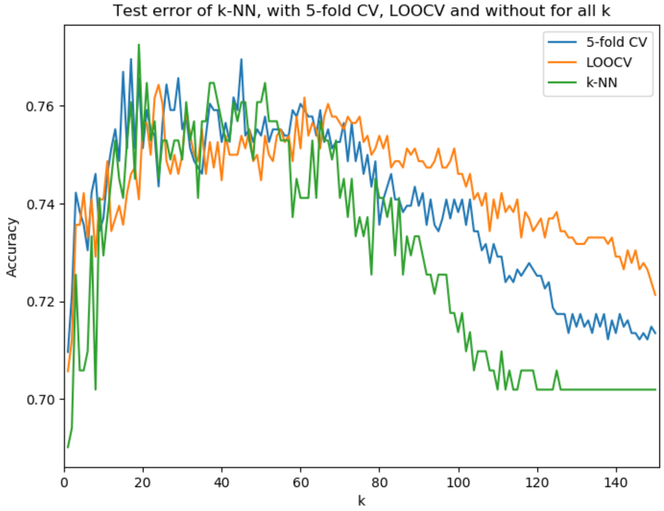

Figure 6: A figure showing the accuracy for all possible k using three different methods.

What's interesting here to me is that the pure `k-NN` actually achieves a higher accuracy than the 5-fold or leave one out cross-validation methods for select small k. For most other k however, `k-NN` is outperformed by the cross-validation methods. As to why `k-NN` outperforms the cross-validation methods at certain ks, this could just come down to statistical luck, given the random nature.

2)

We use a GAM method to examine a given variables effect on the response:

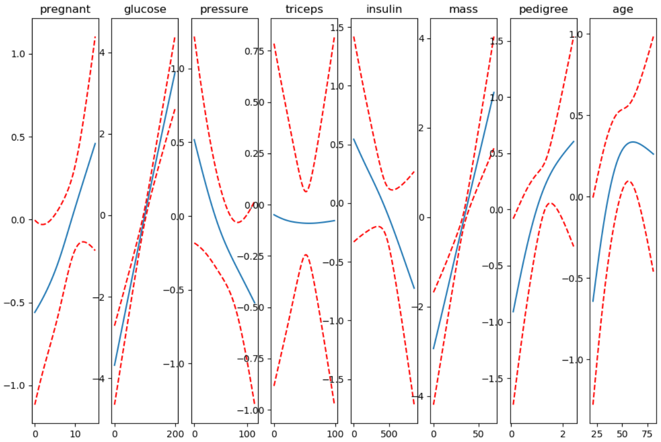

Figure 7: A figure showing how much a variable corresponds with either a positive/negative diabetes classification. The dotted red lines are the uncertainties. Overall accuracy score: 0.77

Unsurprisingly, we see here a very clear correlation between glucose levels, insulin leves, blood pressure and mass, four variables all very commonly associated with diabetes. Additionally, we can also see clearly that women who've not been pregnant trend towards a negative response. The same goes for age, where younger women are less likely to have diabetes. The only variable which seemingly has very little to no correlation with diabetes is the triceps (skin fold thickness), at least for this particular model. Perhaps we could consider culling this variable entirely from our data.

3)

We then find:

	Train	Test
DT	1.0	0.72
PR	0.99	0.75
CO	0.99	0.77
RF	1.0	0.79
AB	0.84	0.76

Table 4: A table showing the training and test error for different ensemble methods
DT: Decision Tree, PR: Bagging w/ Probability voting, CO: Bagging w/ Consensus voting, RF: Random Forest, AB: AdaBoost

We notice that the train error is generally high for all but AdaBoost. This is often indicative of overfitting, which could explain our test scores generally plateauing in the mid-70s, and suggests to us that all models but the AdaBoost model is overfitted. Luckily this doesn't have too much of an effect on our test scores (at least relatively to the not-so-overfitted AdaBoost)

4)

As it stands right now, with just these results and no extra tinkering/tuning I'd say GAM has the most raw analyzing power. In my opinion, I find GAM to be the easiest in terms of reading off which variables matter the most for a given set of data. This in turn of course makes analyzes easier. Of course, that doesn't necessarily make it the best model. There's a case to be made for `k-NN` too, which is a very strong classifier all things considered. Generally most methods are about equal accuracy-wise, although they could likely be substantially improved if I took the time to sufficiently tune parameters. As for the 5 models just above, they're all very overfitted, but this too could in all likelihood be fixed if time was taken to tune parameters.

5)

Let us now examine new results with corrected data:
The updated Figure [6]:

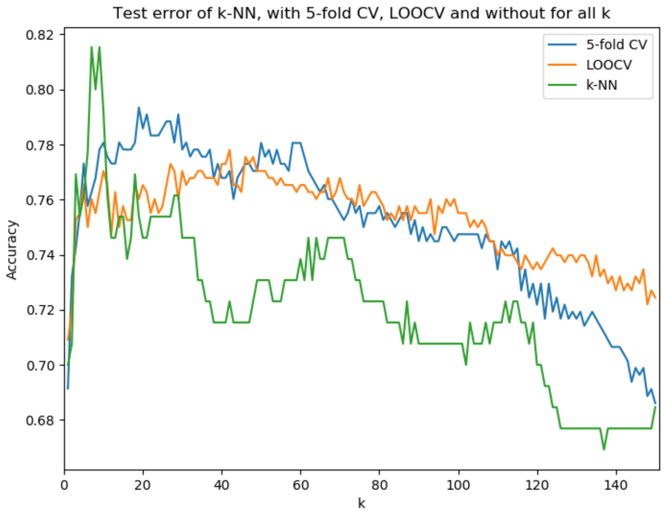

Figure 8: A figure showing the accuracy for all possible k using three different methods.

Again we find that `k-NN` outperforms the other methods early, but then rapidly falls off again. Else we see that the accuracy is higher than in Figure [6], indicating that the false/wrong data "poisoning" was having a significant effect on our model. It is also worth noting that this model is a lot more stable for different k than our previous model.

Let us now examine our GAM results:

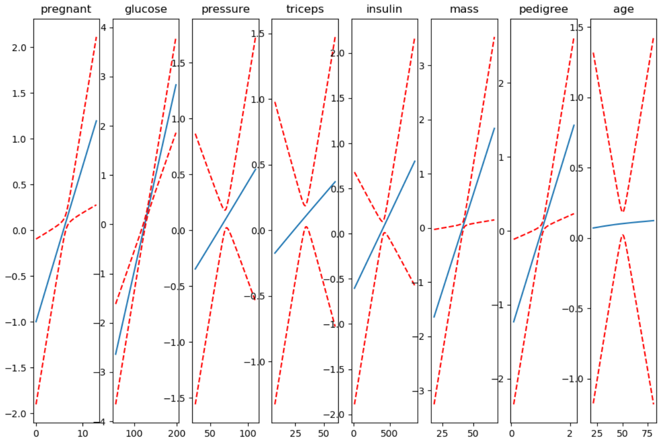

Figure 9: A figure showing how much a variable corresponds with either a positive/negative diabetes classification. The dotted red lines are the uncertainties. Overall accuracy score: 0.77

We see now that our results have changed somewhat. While we now have a lot more clear correlations (notice all the linear relations), we also now have a lot more uncertainty regarding each correlation, except for glucose. We see also now that triceps (which we considered culling from the data in Figure [7] now has a linear relation with diabetes. However, we also see that according to this model high insulin corresponds with diabetes, which is opposite of what we see in reality, where diabetics produce little to no insulin (and thus have high glucose, as seen here).
Lastly the table now becomes

	Train	Test
DT	1.0	0.77
PR	0.98	0.79
CO	0.97	0.76
RF	1.0	0.79
AB	0.95	0.77

Table 5: A table showing the training and test error for different ensemble methods
DT: Decision Tree, PR: Bagging w/ Probability voting, CO: Bagging w/ Consensus voting, RF: Random Forest, AB: AdaBoost

Generally, the train error has had little to no changes whereas we see an improved performance for all test scores but bagging with consensus voting and random forest. The overfitting issue is still not fixed, sadly, and has actually worsened for AdaBoost. Again, we could very likely fix this by applying stricter penalization parameters and general tuning.

References

[1] *Assignment 2*. URL: https://www.uio.no/studier/emner/matnat/math/STK-IN4300/h22/eksamen/oblig_2.pdf.

Appendix - Code

Given the length of the code I thought it'd be nicer to link to a github reposity instead.
The code can be found at https://github.com/simloken/STK-IN4300/tree/main/Assignment2