Report

Mateusz Bąkała, Dominik Rafacz

Metodology

We used the R programming language in the project. The code was prepared using the `renv` package and packages from the `targets` family, which enable reproducible experimental results regardless of the platform used. In order to (more-or-less) reproduce results, use:

```
renv::activate()
renv::restore()
targets::tar_make()
```

in the code directory (it has to be set as working directory).

Additionally, we used packages from the `tidyverse` family to obtain readable, automated code.

The project allowed the use of external implementations of the KNN, LDA and QDA algorithms. We used the KNN implementation from the `e1071` package and LDA and QDA from the `MASS` package.

Selected datasets and initial preparation

We have selected 5 datasets with different characteristics:

- **breast** – medical data, prediction of whether a tumor is malignant or benign; all variables are numeric.
- **creditg** – classification as to whether the person is creditworthy or not; some variables are numeric and some are categorical.
- **wells** – classification of appraisal of oil field well data on the basis of, among other things, rock parameters; has a lot of missing data.
- **amp** – biological sequence data, the aim being to predict whether a protein is an antimicrobial protein on the basis of the amino acid sequence alone .
- **twonorm** – artificial dataset, each of two classes is drawn from multivariate normal distribution (20 dimensions) of unit variance.

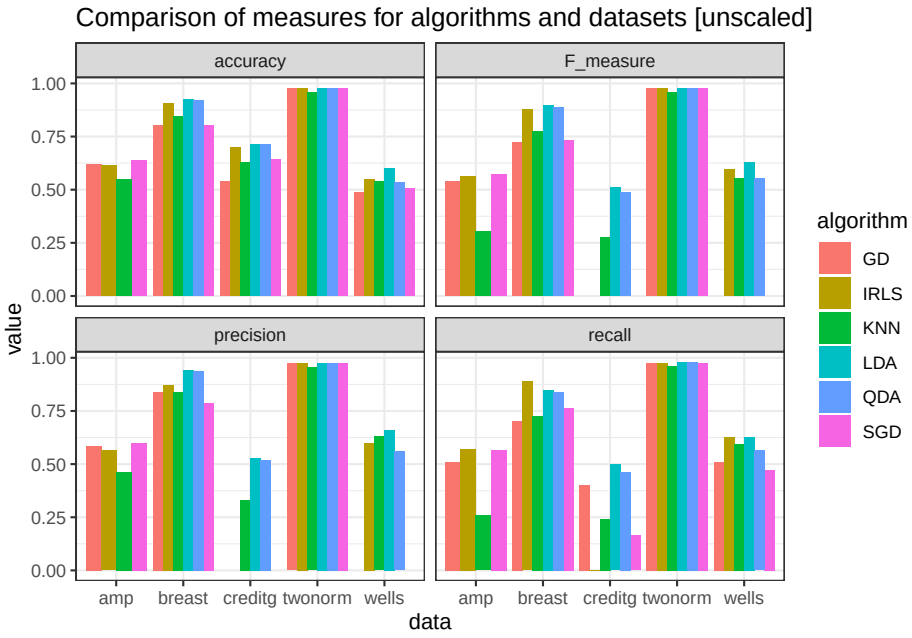
Some datasets required special attention. In the **amp** dataset, the inputs are sequences, stored as character strings. Therefore, here in the data preprocessing we performed ngram counts, i.e. we calculated how often specific subsequences of amino acids (letters) appear in sequences. This frequency matrix must be then filtered to remove uninformative and highly correlated ngrams.

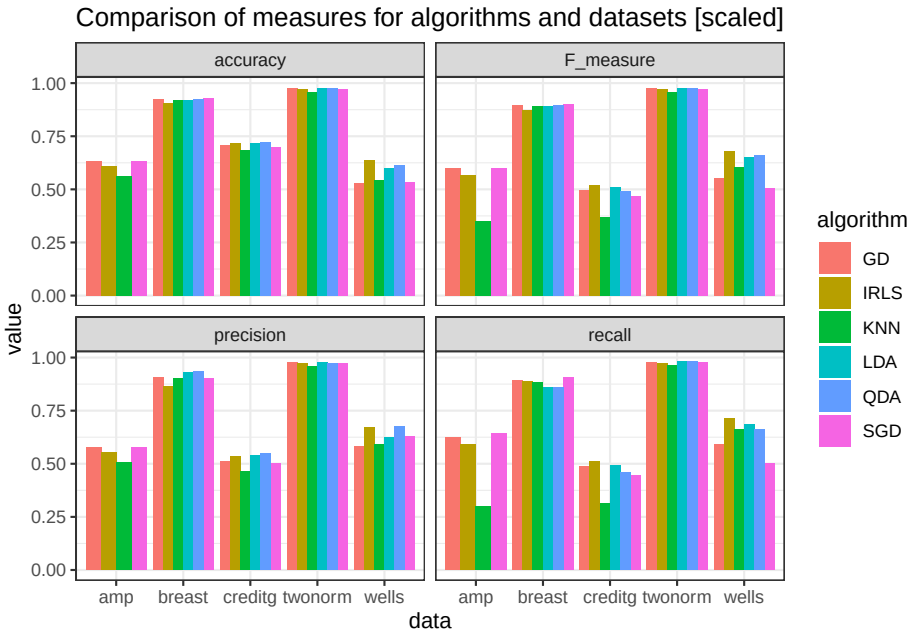
wells dataset contains numerous descriptive variables as names and comments, which were dropped as they were not informative for models. We expected missing values to be Missing At Random so we used median substitution.

All datasets were supplied to algorithms before and after scaling, where scaling means centering all columns and dividing their values by their standard deviation. Thus we could compare whether scaling actually gave any benefit in our case.

Measures

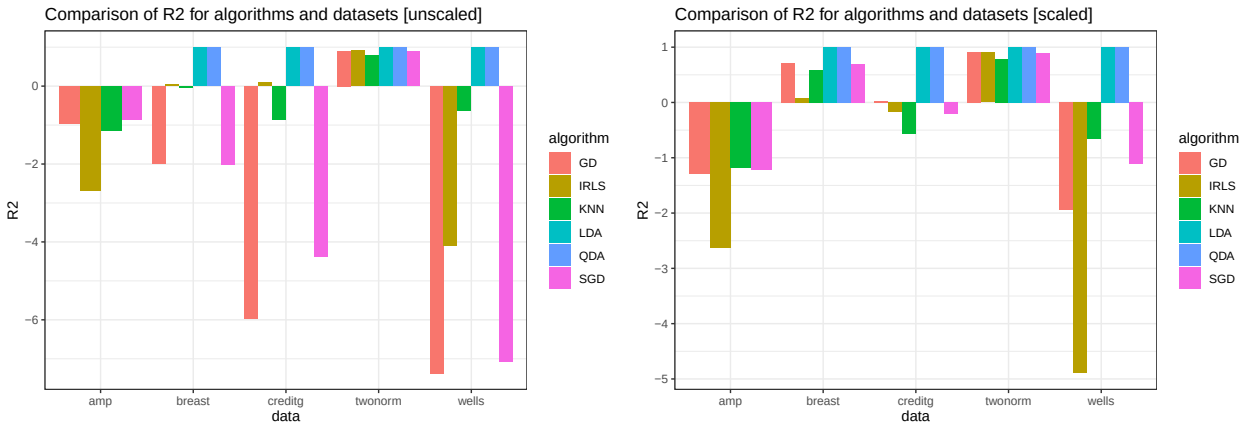
To measure algorithm quality, some measures were necessary. We have implemented measures specified in the task description – accuracy, precision, recall, and F-measure, as well as R-squared (treated separately due to different scale). For each dataset (in both scaled and unscaled version) and algorithm all above measures were computed, resulting in scores shown in the plot below (unscaled and scaled data respectively):





Important point to note is that scores shown are not single scores, but rather means of five runs. Indeed, we performed so-called crossvalidation to train and test algorithm on datasets split in 80:20 proportions multiple times, avoiding random flukes and spikes.

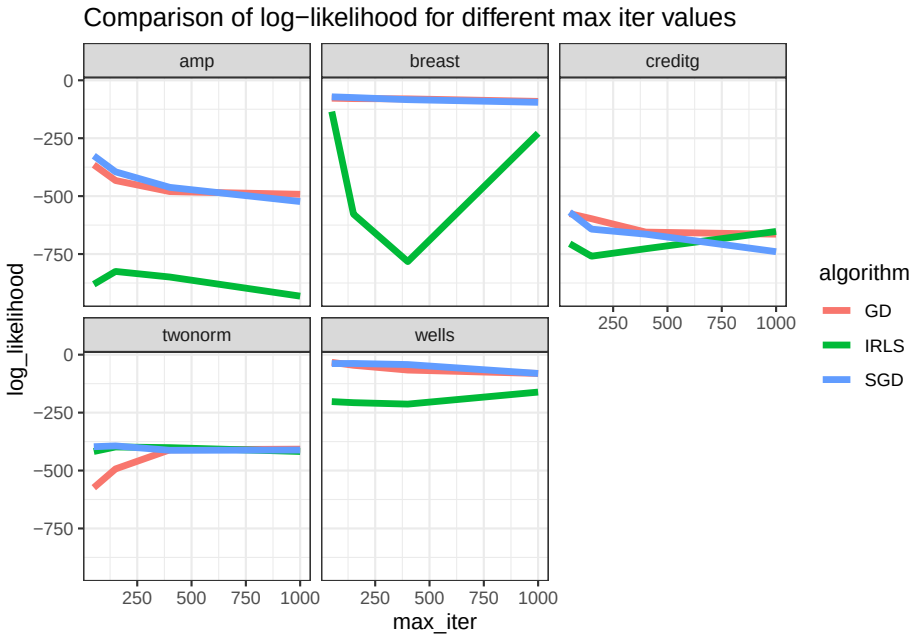
We also include R-squared scores, which turned out to be often concerningly low:



All measures proved that scaling data was worth it, even though benefits were not exactly significant. Gradient descent and its stochastic counterpart were somewhat worse than IRLS, which in turn performed comparably to LDA and QDA. However, LDA and QDA were much more problematic than IRLS, because the user was forced to manually remove perfectly discriminating variables. Also, KNN was significantly worse than other algorithms.

Convergence analysis

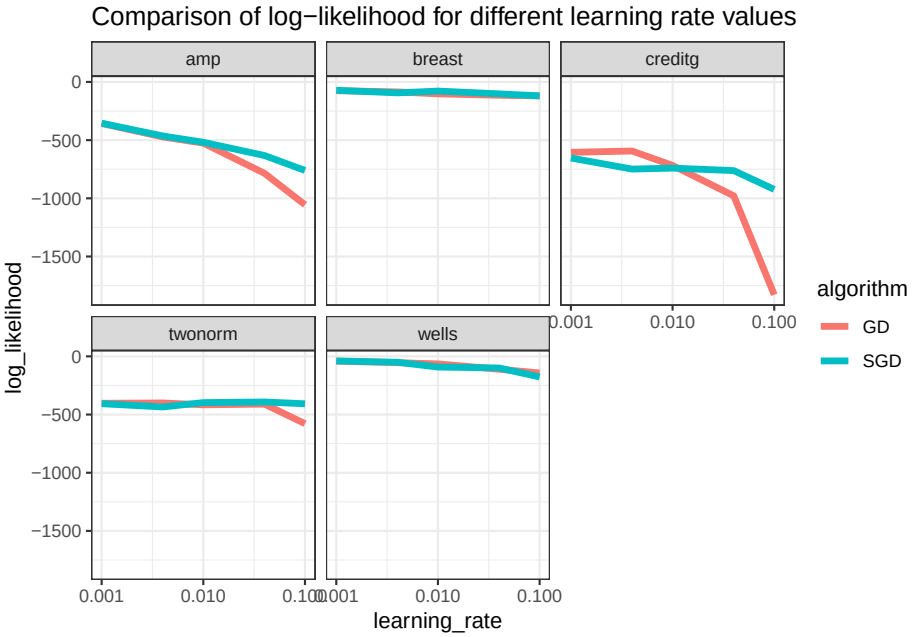
As per task description, our default iteration limit was set to 1000, but it might have been an overkill (or maybe it was not enough?). Thus, we have performed algorithm training with different limit values – 50, 150, 400, and 1000 – to explore and observe possible tendences. Obviously, a thousand words cannot say what five plots can, therefore a plot to bring an enjoyable experience for reader's eyes:



There was no clear tendence, with slight preference towards smaller iteration limit values. However, smaller values are preferred even if due to shorter computation times, thus 50 iterations seem to be enough. Obviously, higher iteration limit might be necessary for more complicated datasets, but there was nothing in our data to support this hypothesis.

Learning rate impact exploration

Gradient descent algorithms can be tuned using learning rate, which determines how aggresively trained coefficients are updated each iteration. We tested the following values: 0.001, 0.004, 0.01, 0.04, and 0.1.



Here, conservative approach was rewarded with better log-likelihood score. No dataset has shown opposite tendency. Conservative approach was working for both GD and SGD, though significantly better for the former.

Additional remarks

The stopping rule we used for all our implemented algorithms was that whenever all absolute values of coefficient changes were lower than (or equal to) some arbitrarily chosen epsilon, then training was stopped. Examples of such epsilon are 0, 1e-7 and 1e-4.