University of Oslo

Project 3.

Leveraging machine learning models for predicting future vector-borne disease cases.

Ingar Andre Benonisen, Martin Hansen Bolle

FYS-STK4155/Oblig3/Department of Informatics

09.12.2024

Abstract

Vector-borne diseases, such as malaria and dengue fever, remain significant global health challenges, responsible for over 700,000 deaths annually. This project aims to leverage machine learning techniques, including ridge regression, decision tree, random forest, and bagging, to forecast vector-borne disease cases. The models are evaluated through grid-search to optimize hyperparameters and compared against each other to assess their predictive capabilities and find the best predictive model. We found that Ridge regression displayed the best overall result, compared to decision tree (DT), bagging (BG), and random forest (RF) with the optimal parameters of lambda equal 0.01 and a poly degree of 1. We also saw that DT, BG, and RF exhibited different responses to variation in parameter settings. DT had the poorest performance of all methods used. Regarding bagging, increasing features and samples drawn from the training dataset when training the base estimator improved performance. We also found that Time series cross validation had an impact on the performance of RF, in contrast to the other methods. Another important aspect of our analysis showed that our model did, to some extent, capture the presence of outbreaks.

Content

Content 3
Introduction 4
Dataset 4
Methods 5
Ordinary Least Squares 5
Ridge Regression 6
Decision trees 6
Bagging 7
Random Forrest 8
Time-series cross validation. 8
Performance Metrics 9
R2-score 9
Root Mean Squared Error. 9
Code implementation 10
Structure 10
Data preprocessing 11
Results 11
Ridge 11
Decision tree 12
Bagging 13
Random forest 14
Discussion 14
Conclusion and perspectives 15
Future work 15
Literature 16
Appendix 16
Project one. 16
Project two. 16

Introduction

Vector-borne diseases are human illnesses transmitted by vectors carrying pathogens and cause over 700,000 deaths annually. Malaria and dengue fever are the most common diseases. These diseases have afflicted, caused most deaths and been the largest burden on health care systems in tropical and subtropical areas, subsequently affecting the poorest populations the most (World Health Organization, 2020).

There are several concerns that climate change will affect multiple aspects of vector-borne diseases. First, mosquitoes, arthropods and other vectors are ectothermic and strive in warmer climates. Thus, the endemic areas will move to higher altitudes and latitudes and transmission season length will increase. Global trade and travel can disperse vectors to new non-endemic areas which were previously not suitable for vector establishment. This has been the case in southern Europe. Second, a warmer climate and more precipitation will cause higher reproduction rates and subsequently accelerate vector and pathogens evolution (Rocklöv & Dubrow, 2020).

WHO defines climate services as "the provision and use of climate data, information and knowledge to assist decision-making" (Global Framework for Climate Services (GFCS), 2021). Services can help the health sector by e.g. communicating climate related health risks to professionals and vulnerable populations, identify and distribute resources to the more vulnerable areas, identify when climate related health risks are highest and intervene to minimize risks, subsequently making health systems climate resilient (Agache et al., 2022). One service is to predict incidences of vector-borne diseases, like malaria and dengue-fever.

The goal of this project is to use Ridge Regression, Decision Tree, Random Forest and Bagging to forecast vector-borne disease cases. Through grid-search, various hyper-parameters will be tested on the models and give insights into their behavior. The models will also be compared against each other.

The structure of the report begins with an overview of the dataset used in the analysis, describing the characteristics, variables and any preprocessing steps that were necessary. Following this, the "methods" chapter provides the necessary theory for understanding the methods used in the analysis. Furthermore, the report includes a chapter on the code implementation, discussing the architecture, technical workflow and reproducibility of the analysis. Before concluding the research of this report, the results will be analyzed and discussed. Lastly, any relevant material or literature can be found in the appendix and literature list.

Dataset

The dataset used in this analysis was extracted from the HISP research group at the University of Oslo (UiO), specifically from their GitHub repository chap-core within the DHIS2-chap GitHub organization. Even more specifically, we used the file 'historic_data' from the example_data/v0 folder (see Appendix, *Dataset source*).

The dataset includes the columns "time_period", "rainfall", "mean_temperature", "disease_case", "population", "location", "month", and "ID_year". The dataset can be

described as stochastic non-linear time series data, where each row represents a month in Brazil.

Figure 1 shows the sickness cases in Alagoas from March 2001 to June 2016. The graph has some clear spikes, indicating seasonality in sickness cases. It also shows that sickness cases range from 17 500 to near zero.

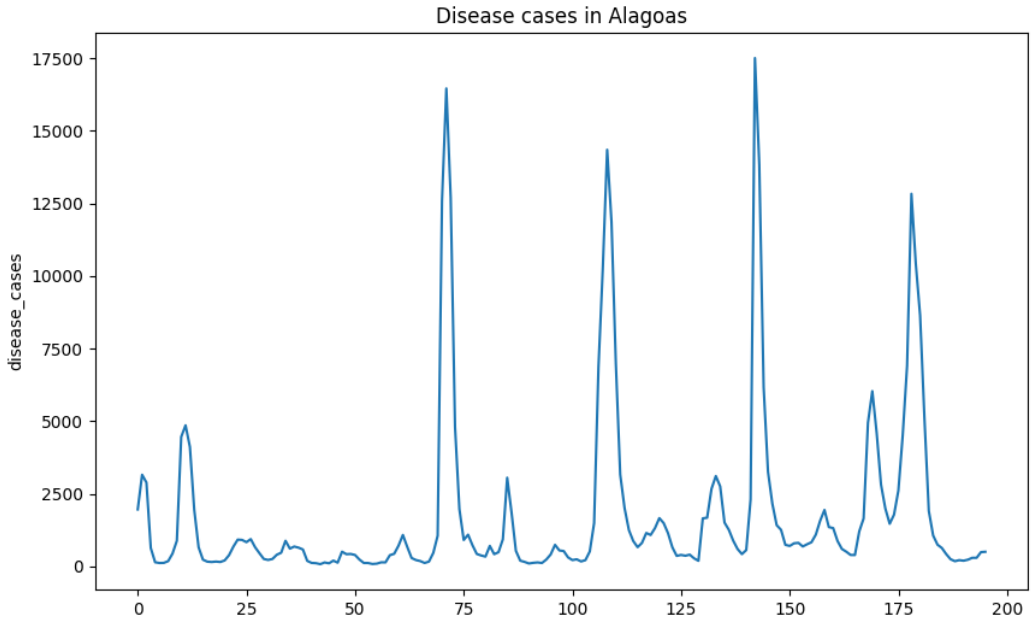

Figure 1 Disease cases in Alagoas from March 2001 to June 2016

Methods

The following sections *Ordinary Least Squares* and *Ridge Regression* are reused from the previous research paper *Project 1 Regression Analysis and resampling methods* Ordinary Least Squares (Ingar Benonisen & Martin Bolle, 2024).

Ordinary Least Squares

Having a basic understanding of ordinary least squares (OLS) is important to understand ridge regression. Hence, we will look further into OLS, before continuing with ridge regression.

OLS is the most basic method for fitting a linear regression model. It aims to minimize the sum of the squared differences between the observed value y and the predicted value $X\beta$ (where X is the input feature and β is the coefficients), which gives us the equation

$$Minimize \sum_{i=0}^{n-1} (y_i - \hat{y}_i)^2$$

With matrix notation we get the loss function

$$C(\beta) = (y - X\beta)^T (y - X\beta)$$

Taking the derivative with respect to β if, and only if, $X^T X$ is invertible, we get the solution for our optimal β

$$\beta = (X^T X)^{-1} X^T y$$

Ridge Regression

Ridge regression aims to find which variables are significant and which variables are not, and then removing the influence of the insignificant variables by setting them close to zero. This is done by adding the shrinkage parameter lambda to both sides of the OLS equation (Hastie et al., 2017)

With our insight from OLS, we can derive the equation for Ridge regression. Ridge regression is ordinary least squares regression with an L2 penalty term on the weights in the loss function. This is what we call the regularization term. This term penalizes large coefficients and helps prevent overfitting. From the loss function from OLS, we get

$$C(\beta) = (y - X\beta)^T (y - X\beta) + \lambda ||\beta||_2^2$$

Here, the λ is a hyperparameter that controls the strength of the penalty, and $||\beta||_2^2$ is the squared L2 norm of the coefficient vector β, which equals the sum of the squared coefficients

$$||\beta||_2^2 = \sum_{j=1}^{p} \beta_j^2$$

By minimizing the loss function above with respect to β we can obtain an analytical expression for the optimal parameter β

$$\hat{\beta}_{ridge} = (X^T X + \lambda I)^{-1} X^T y$$

Where I is the identity matrix and lambda are the penalty term.

Decision trees

A decision trees main idea is to find those descriptive features which contain the most information regarding the target feature and then split the data along the values of these features such that the target feature values for the resulting underlying dataset are as pure as possible.

The process of finding the most informative feature is done until we accomplish a stopping criterion where we then finally end up in a so-called leaf node. These stopping criteria are the minimum samples per split, max depth or impurity criterion. Minimum sample split is the minimum number of data points required in a node before it can be split further. Maximum depth specifies how deep a tree can grow. When it reaches its limit it stops, and the remaining nodes become leaves. Lastly, if a node is pure, it means that all samples have the same value. At this point further splitting is not necessary.

Our task is to use the decision tree for a regression case. This consists of two steps. We split the predictor space, meaning the set of possible values $x_1, , x_2, \dots, x_p$ into J distinct and non-overlapping regions $R_1, , R_2, \dots, R_J$.

For every observation that falls into the region R_J, we make the same prediction, which is simply the mean of the response values for the training observations in R_J. We divide the predictor space into high-dimensional rectangles, or boxes, and our goal is to find the boxes that minimize the MSE given by

$$\sum_{j=1}^{J}\sum_{i\epsilon R_j}(y_i - \bar{y}_{R_j})^2$$

Where $\bar{y}_{R_j}$ is the mean response for the training observations withing box j.

We are using recursive binary splitting when creating the tree. In order to implement this splitting, we start by selecting the predictor x_j and a cut-point s that splits the predictor into two regions R_1 and R_2

$$\{X|x_j < s\},$$

And

$$\{X|x_j \geq s\},$$

So that we obtain the lowest MSE, that is

$$\sum_{i:x_i\epsilon R_j}(y_i - \bar{y}_{R_1})^2 + \sum_{i:x_i\epsilon R_2}(y_i - \bar{y}_{R_2})^2$$

Which we want to minimize considering the predictors. Next, we repeat the process, looking for the best cut-point in order to split the data further so as to minimize the MSE within each of the resulting regions.

Bagging

Bagging, a portmanteau of "bootstrap aggregation", is a general-purpose procedure for reducing the variance of a statistical learning method. It is an ensemble learning technique used to improve the performance and robustness of a machine learning model. For our case, using bagging for regression with decision trees, multiple decision trees are trained on different bootstrapped subsets of the training data, and their predictions are aggregated to produce the final output.

Bagging starts by generating multiple datasets using bootstrap sampling. Given $D = \{x_1, x_2, \dots, x_n\}$, a bootstrap sample D_b is created by sampling n points with replacement from D as

$$D = \{x_{b1}, x_{b2}, \dots, x_{bn}\}, \qquad x_{bi} \sim D,$$

Each bootstrap sample is used to train a separate decision tree regressor, trying to minimize the loss function.

Once all B trees are trained, the final prediction for a new observation x is obtained by averaging the predictions from all individual trees. This is given by

$$\hat{y}_{bagged}(x) = \frac{1}{B}\sum_{b=1}^{B} f_b(x),$$

Where $f_b(x)$ is the prediction of the b-th tree.

The key benefit, like reducing the variance is expressed as

$$Var(\hat{y}_{bagged}) = \frac{1}{B}\, Var(f(x)),$$

Random Forrest

Like bagging, random forest works like an ensemble method as well. In addition to drawing random samples of the data, Random Forest introduces additional randomness by randomly selecting subsets of the features to use when splitting each node in the trees.

Random forest also generates multiple bootstrap samples from the original dataset D like

$$D = \{x_{b1}, x_{b2}, \dots, x_{bn}\}, \qquad x_{bi} \sim D,$$

Then we introduce a random feature selection during the tree construction by selecting a random subset of features.

$$S_b = \{f_{b1}, f_{b2}, \dots, f_{bn}\} \subseteq \{f_1, f_2, \dots, f_n\},$$

And determine the optimal split among the selected features by minimizing the MSE

$$s^* = \arg min_{\, s \,\epsilon\, S_b} \sum_{j=1}^{J} \sum_{i \epsilon R_j} (y_i - \bar{y}_{R_j})^2$$

Where s^* is the optimal split.

Each bootstrap sample D_b, a decision tree T_b is trained using the selected random subset of features at each split. As discussed in decision trees we split based on the stopping criteria or if we happened to be in a leaf node. Once all B trees are trained the final prediction can be expressed as

$$\hat{y}_{Random\ Forest}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x),$$

Time-series cross validation.

Cross validation is a crucial step in evaluating the performance of a model. We use this to avoid overfitting, optimize hyperparameters and make the most out of limited data. It provides a more robust and reliable performance evaluation, helping to ensure that the model is both accurate and reliable.

In forecasting time series data is inherently temporal, meaning that the observations are sequential and depend on the order of events. Standard cross-validation techniques, such as k-gold cross-validation, are not appropriate for time series data because they randomly split the data intro training and test sets, violating the temporal structure of the data. Potentially, this

can lead to data leakage where future data points are included in the training set. We have therefore implemented Time-series cross-validation specifically to preserve the temporal structure of the data. We have used the expanding window approach. In expanding-window cross-validation, at each fold t, we train the model on a training set T_t that expands over time, and test it on a subsequent test set D_t, defined as

$$T_t = \{X_1, X_2, \dots, X_t\}$$

$$D_t = \{X_{1+1}, X_{2+1}, \dots, X_{t+h}\}$$

For each fold, we calculate the performance metric $L(T_t, D_t)$ for the models' predictions $\hat{Y}$, where L represents the loss function. Overall performance of the model is averaged over all the folds

$$L_{avg} \frac{1}{n} \sum_{t=1}^{N} L(T_t, D_t)$$

Where N is the number of folds.

Performance Metrics

To quantify the performance our model and interpret our results we have used well known performance metrics like Root mean squared error and R2-score. This is done to measure the quality of our predictions.

R2-score

For our analysis we reused the coefficient of determination, known as the R2 score, from the previous research paper *Project 1 Regression Analysis and resampling methods* Ordinary Least Squares (Ingar Benonisen & Martin Bolle, 2024)

The R^2 score provides a measure of how well future samples are to be predicted by the model. The best possible score here is 1.0, and the score can be negative. It is the proportion of the variation in the dependent variable that is predictable from the independent variable(s). With the same $\hat{y}$ as the predicted value the R^2 is defined as

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n-1} y_i - \hat{y}_i{}^2}{\sum_{i=0}^{n-1} y_i - \bar{y}_i{}^2}$$

Where we have the mean value $\bar{y}$ defined as

$$\bar{y} = \frac{1}{n} \sum_{i=0}^{n-1} y_i$$

Root Mean Squared Error.

For a better understanding of our results, we have also used root mean squared error (RMSE) to gain a deeper insight of the results. RMSE measures the average magnitude of error between predicted values and actual values, with focus on penalizing large errors more heavily. It is defined as

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Where n is the number of observations, and $(y_i - \hat{y}_i)$ is the difference between actual and predicted values (residual). As our dataset contains outliers RMSE is a well-suited metric for handling this.

Code implementation

Before delving into our architecture, we would like to emphasize that we have leveraged Scikit-learn's built in libraries and frameworks for our code implementation of the various methods (Pedregosa et al., 2011). As a widely used and extensively tested library, scikit-learn provides robust, optimized and computationally efficient algorithms. This streamlines the implementation process by minimizing the risk of errors, reduces the time required to build models from scratch, and allows us to focus more effectively on the analysis and interpretation of results.

Furthermore, we have utilized Matplotlib and Seaborn for plotting, Numpy and Pandas for data processing, and Joblib for storing python objects, such as the trained model.

Structure

We structured the evaluation, training and predicting processes into separate files to ensure clean code. Furthermore, data preprocessing, plotting and other helper functions have been added to a utils folder.

The evaluation process includes grid search over various hyperparameters for the given model. Grid search finds the optimal hyperparameters by comparing the average root mean squared error from the time series cross validation done in the testing process. The optimal hyperparameters are used to predict in the testing process, which gives us the final assessment of the model.

The training process finds the mean absolute error and r2 score through time series cross validation, with hyperparameters defined in the evaluation process. Lastly, the entire training dataset is used to predict disease cases, with the specified hyperparameters, in order to plot the results which are saved under its respective folder in the figures folder.

The testing process uses the trained model with the optimal hyperparameters found in the evaluation process, to predict on the testing dataset. The results are also plotted and saved under their respective folder in the test folder in the figures folder.

To reproduce our results, you can simply run the testing process by executing the predict.py script. Ensure that you include an argument to specify the model to be tested. This will use a pretrained and optimized model. You can also run the evaluation process by executing the evaluate.py script, followed by a specific model. This will run the training and testing process.

Data preprocessing

A critical step in machine learning is the preparation of data before training. Models generally produce better results with well-preprocessed datasets. This reduces noise, handles missing values, and ensures that the data algins with the assumptions of the algorithms used, improving model accuracy and robustness (Gonzalez Zelaya, 2019).

The dataset provided by HISP required significant preprocessing. To address this, we developed helper functions to handle missing values, removal of irrelevant columns, and split the dataset into training and testing sets. Rather than inputting missing values with zeros, a method that could degrade model performance, we opted to use the median of the respective column as a more robust approach. This preprocessing ensured a higher quality dataset of subsequent analyses.

An important aspect of this project is the fact that we are working with time-series. In such cases, the impact of an exposure event may not be immediate but can manifest after a delay. This creates a challenge in modeling the relationship between the timing of an exposure and a series of subsequent outcomes over time (Gasparrini et al., 2010). To address this, we added four new columns to the dataset: moving averages for disease cases, rainfall, and mean over the prior three months, and a feature representing the previous month's disease cases. These additions aim to capture temporal patterns and account for delayed effects in the data.

Results

In this chapter, we will delve into the results from the evaluation and testing of the models, whereas the discussion and comparison of the model performances will be done in the next chapter.

Ridge

When evaluating ridge regression, we tested various lambda values (L2 regularization) and polynomial degrees, as table 1 displays. The results, and figure 2, clearly show that models with polynomial degrees higher than one performed significantly worse than linear models (polynomial degree one). We also saw that, up to some point, increasing the lambda value improved the model performance. After this point (0.01), the performance slightly decreased.

The evaluation process showed that the optimal hyperparameters for ridge regression was {lambda value: 0.01, polynomial degree: 1}. The final prediction on the testing dataset and with optimal hyperparameters, resulted in the scores {RMSE: 1192, r2-score: 0.83}.

Tabel 1 Hyperparameters tested on a ridge regression model

Hyperparameter	Values
lambda	0.0001, 0.001, 0.01, 0.1
polynomial	1, 3, 5, 7, 9

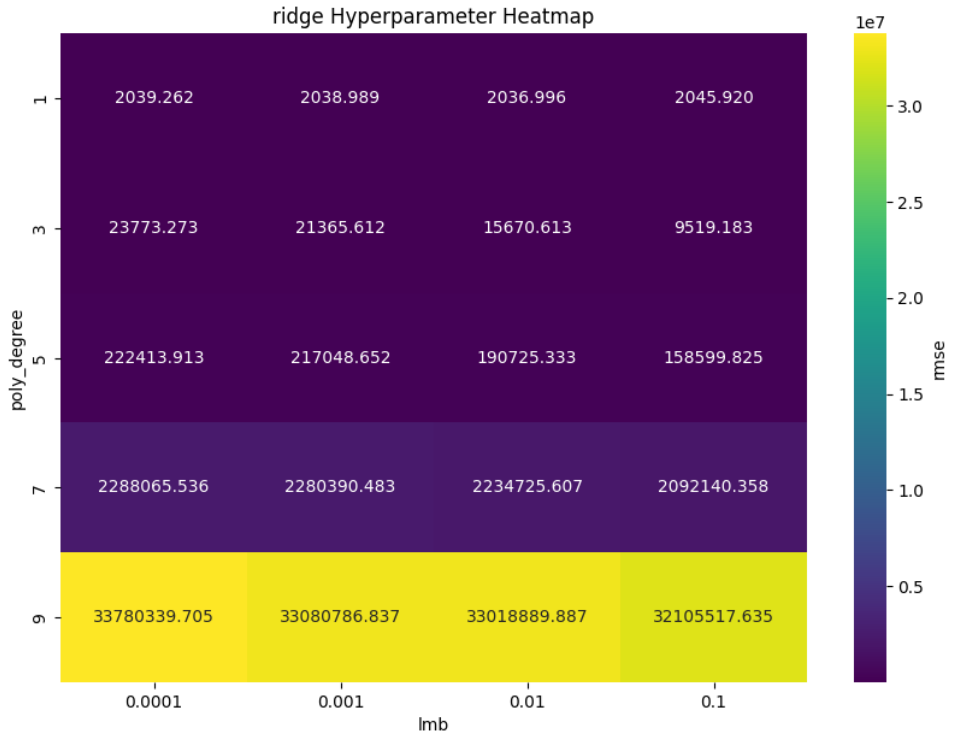

Figure 2 Heatmap of RMSE for ridge regression from TSCV, displaying the combination of polynomial degree and lambda value.

Decision tree

When evaluating decision tree, we experimented with different values for the maximum depth of the tree (max_depth) and the minimum samples required before splitting (min_samples_split). Table 2 gives an overview of the hyperparameters, and the values tested.

Looking at the results from the grid search (Figure 3), we found that higher min_samples_split resulted in a better performance. Furthermore, we also found that increasing max_depth was insignificant.

The optimal hyperparameters proved to be 5 for max_depth and 15 for min_samples_split. These hyperparameters resulted in an RMSE of 2349 and a r2-score of 0.36, when predicting on the testing dataset.

Tabel 2 Hyperparameters tested on a decision tree model

Hyperparameter	Values
max_depth	5, 10, 15
min_samples_split	5, 10, 15

Figure 3 Heatmap of RMSE for decision tree from TSCV, displaying the combination of min_samples_split and max_depth

Bagging

When evaluating bagging, we tested different values for the number of trees to be created (n_estimators). We also experimented with using fraction of the samples from the training dataset to train each base estimator (max_samples). The same fractions were applied to the number of features to be drawn from the training dataset to train the base estimators (max_features). See table 3 for an overview of the hyperparameters and the values. Also, the bagging model used the optimal decision tree we found previously, as the base estimator.

The results (see Appendix, *Bagging Results*) from the evaluation process show that increasing max_samples and max_features improves performance. When n_estimators is above 50, the performance is better when max_samples is 1.0. However, when n_estimators is 50 or below, the performance is better when max_samples is 0.75.

The optimal hyper parameters proved to be {max_samples: 0.75, max_features: 1.0, n_estimators: 100}. Predicting disease cases with the testing dataset gave the performance metrics {RMSE: 1647, r2: 0.68).

Tabel 3 Hyperparameters tested on a bagging model

Hyperparameter	Values
max_samples	0.5, 0.75, 1.0
max_features	0.5, 0.75, 1.0

n_estimators	10, 50, 100, 200, 300

Random forest

The random forest model employs the same stopping criteria as the decision tree, specifically the hyperparameters max_depth and min_samples_split. Additionally, similar to bagging, the hyperparameter n_estimators is introduced to control the number of trees in the ensemble. Table x shows the hyperparameters and the values tested in the grid search.

From the results (see Appendix, *Random Forest Results*), we see that a lower min_samples_split improves performance. Changing the max_depth and the number of trees seems to be insignificant.

The evaluation process has also been executed without time series cross validation, to see if there was any difference. For ridge regression, decision tree and bagging, there was little difference, but for random forest we saw an improvement of the r2-score by 10%.

The optimal hyperparameters ended up being {max_depth: 5, min_samples_split: 15, n_estimators: 50}. With these hyperparameters, the model achieved performance scores of {RMSE: 1399, r2: 0.77}.

Tabel 4 Hyperparameters tested on a random forest model

Hyperparameter	Values
max_depth	5, 10, 15
min_samples_split	5, 10, 15
n_estimators	10, 50, 100, 200, 300

Discussion

We see from the results, that ridge regression outperformed decision trees, random forest and bagging. This is most likely due to the small dataset we are working on, where model complexity often results in overfitting. It becomes even more apparent when we look at the evaluation of ridge regression itself, where the linear model outperformed models of polynomial degree above one. As the model complexity for ridge regression increases, the model performance decreases, indicating overfitting. Increasing the lambda value also improves the performance, which also mitigates overfitting.

Decision trees performed better when the minimum samples required for splitting were high, which makes sense as it reduces the complexity of the tree, which mitigates overfitting. Decision trees had the worst performance among the different models. This makes sense as decision trees are prone to overfitting, especially on smaller datasets, whilst the other models mitigate overfitting in different ways.

We saw the same trends in random forest, as in decision tree. A lower min_samples_split resulted in less model complexity, lower RMSE and variance. However, random forest improved the RMSE by 40%. This improvement can be explained by looking at how random forest mitigates decision tree's weaknesses and amplifies its strengths. The randomness in

feature selection helps avoid focusing too heavily on specific features. The ensemble can capture more intricate patterns, by combining diverse models. Averaging the results from the trees makes it less affected by outliers and noise, and reduces overfitting.

Bagging performed better when the number of samples drawn from the training dataset was high, as it allows each base estimator to better capture the underlying pattern, effectively mitigating underfitting. The model performed best when the hyperparameters max_samples and max_features were the highest, suggesting that the model performs best when all samples and features are available for training the base estimators.

The number of trees had little effect on random forest and bagging. There comes a point when increasing the number of trees in an ensemble has little to no improvement in performance. This is because we reach a sufficient number of trees and the variance in the ensemble stabilizes. At this point, increasing the number of trees would only add computational cost, as the variance will remain the same regardless of number of trees.

Conclusion and perspectives

The analysis demonstrates that ridge regression outperformed ensemble methods like bagging and random forest, as well as individual models like decision trees, in predicting disease cases. This result is likely attributable to the characteristics of the dataset, which was relatively small in size. Linear models are well-suited for situations with limited data because they incorporate regularization to prevent overfitting and ensure that the model generalizes well to new data.

Among the ensemble methods, random forest performed better than bagging. Decision trees showed the poorest performance. This highlights the robustness of random forests, which combine predictions from multiple trees with randomized feature selection, reducing variance and improving predictive accuracy. In contrast, bagging does not randomize features, potentially leading to correlated trees and reducing performance. The poor results of decision trees further emphasize their susceptibility to overfitting, particularly in small datasets with complex non-linear relationships common in disease prediction. These findings underscore the importance of selecting models suited to the dataset's size and complexity.

As discussed in the previous chapter, limited data poses several challenges to the reliability and generalizability of our predictive models, as their performance heavily depends on the quality, quantity and representation of data. Insufficient data often fails to capture crucial patterns and relationships leading to overfitting, where the model performs well on training data but struggles with unseen cases. Furthermore, health data related to disease cases is typically highly skewed. This imbalance can disproportionately influence the model, as the dataset lacks sufficient "normal" cases to mitigate the impact of outliers.

Future work

We explored regression techniques, like ridge, decision trees and ensemble methods for predicting future disease cases. While more complex architectures, such as deep neural networks or hybrid approaches, could potentially capture intricate patterns, their effectiveness is contingent on the availability of sufficient data. Collecting additional data would undoubtedly improve model training, regardless of the technique employed. However, in

many developing countries, limited resources often result in poor weather observation and incomplete datasets, making the collection of large-scale data challenging and unrepresentative of real-world conditions. Ideally, the goal is to develop models capable of identifying meaningful patterns and delivering reliable predictions, even with limited data availability. This could be done by stress testing models on simulated data. This is an ongoing research field at UiO's research group HISP, and a part of the authors' master's thesis.

Literature

Gasparrini, A., Armstrong, B., & Kenward, M. G. (2010). Distributed lag non-linear models. *Statistics in Medicine*, *29*(21), 2224–2234. https://doi.org/10.1002/sim.3940

Gonzalez Zelaya, C. V. (2019). Towards Explaining the Effects of Data Preprocessing on Machine Learning. *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, 2086–2090. https://doi.org/10.1109/ICDE.2019.00245

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. The MIT Press.

Hastie, T., Tibshirani, R., & Friedman, J. H. (2017). *The elements of statistical learning: Data mining, inference, and prediction* (Second edition, corrected at 12th printing 2017). Springer.

Ingar Benonisen & Martin Bolle. (2024, October 7). *Regression analysis and resampling methods.* https://github.com/martin-og-ingar/FYS-STK4155-Project1/blob/main/report/Project_1_report.pdf

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*(85), 2825–2830.

Appendix

Project one.

https://github.com/martin-og-ingar/FYS-STK4155-Project1

Project two.

https://github.com/martin-og-ingar/FYS-STK4155-Project2

Dataset source

https://github.com/dhis2-chap/chap-core/tree/dev/example_data/v0)

Bagging Results

Graphs:

https://github.com/martin-og-ingar/FYS-STK4155-Project3/tree/main/figures/bagging

RMSE:

https://github.com/martin-og-ingar/FYS-STK4155-Project3/tree/main/figures/rmse/bagging

R2:

https://github.com/martin-og-ingar/FYS-STK4155-Project3/tree/main/figures/r2/bagging

Graph from final test:

https://github.com/martin-og-ingar/FYS-STK4155-Project3/blob/main/figures/test/bagging.png

Random Forest Results

Graphs:

https://github.com/martin-og-ingar/FYS-STK4155-Project3/tree/main/figures/rf

RMSE:

https://github.com/martin-og-ingar/FYS-STK4155-Project3/tree/main/figures/rmse/rf

R2:

https://github.com/martin-og-ingar/FYS-STK4155-Project3/tree/main/figures/r2/rf

Graph from final test:

https://github.com/martin-og-ingar/FYS-STK4155-Project3/blob/main/figures/test/rf.png