

Protocol for "Spezielle Bioinformatik 3" Using Classical Machine Learning Against Pre-Existing Deep Learning Models in PET/CT-Based Local Prostate Cancer Recurrence

Abstract:

Background:

As by far the most common cancer in men, Prostate Cancer (PC) is of great interest for medical research [1]. Of the various established imaging methods, PSMA-directed positron emission tomography (PET) has proven itself as a reliable method for diagnosing and detecting recurrent prostate cancer as well as metastasis. A deep learning model previously trained on [18F] prostate-specific membrane antigen (PSMA)-1007 PET scans to detect local PC recurrence serves as a baseline for comparison of performance [2]. Training another model with classic machine learning methods on the same dataset, the goal was to see how close in performance we could get with as simple an approach as possible.

Methods:

Multiple models based on three different algorithms were trained on the metadata of a dataset including 1404 [18F]-PSMA-1007 PET/CT re-staging scans from patients with histologically confirmed prostate cancer.

Conclusion:

In the presented data, we show that using deep learning as the preferred approach for analyzing this dataset of 1404 scans was a valid choice, as it does not rely on manual feature extraction to function. With the highly simplified approach, we managed to closely match F1 and accuracy, surpassing it in some cases. This leads to the belief that simply the addition of the prostatectomy status has a high degree of influence over the prediction result. Data processing and amount may hold back both approaches from achieving clinically relevant models.

1. Introduction

The goal of this internship is to take a simplified approach to prostate cancer recurrence detection from PET/CT scans, or rather the tabular data derived from patients.

In recent years artificial intelligence (AI) has shown great promise in the medical field by processing large quantities of data that would otherwise require medical professionals to spend a large amount of time on manually analyzing [3].

While deep learning (DL) is popular for complex image analysis such as this, the “black-box” nature of DL poses a problem to medical acceptance in the field.

We want to show that traditional machine learning models, who's mechanism can be further examined, can achieve similar results.

2. Materials and Methods

2.1 Data processing/Study population

The data used for this project was based on the metadata obtained from patient scans using 18F - PSMA-1007 PET/CT imaging from the Department of Nuclear Medicine at the University Hospital Würzburg, conducted between 2019 and 2023. The initially provided patient split was 1016 scans/patients for training, 188 for validation, and 200 as a test set to be conducted at the end of the project.

The metadata used for model training consists of the data obtained from each patient that was referred to primary/follow-up screening due to elevated PSA values.

Per patient there are two randomized IDs for privacy, age of the patient, gender, whether or not it's the primary or restaging of said patient, status of prostatectomy PX (0 = no prostatectomy, 1 = prostatectomy), levels of the PC indicative protein PSA, the label (0 = no cancer, 1 = cancer, 2 = uncertain), and the intended set with either training or validation. The actual images were not integrated into the model training. At later stages, the consideration was made to add derived features from the image data, such as image intensities and potential lesion volumes. Inclusion of the additional features was deemed outside the scope of the initial project and to be looked at in the follow-up project.

The patient data was simplified as much as possible after the initial model training to introduce as few variables or potential issues as possible. For this, any data containing the label 2, rows containing features with N/A and primary staging patients were excluded after initial testing including all data. Following this, only the relevant features of age, PSA and PX were included in further training. Validation/Training masks were created and matching was done via the index to remove both patient IDs.

The overall data was reduced from 1205 viable rows to 872 rows, with X and Y being split for the test and the validation set.

2.2 Data logging/documentation via GitHub and MLflow

For version control, GitHub was used. To learn how to use Git and GitHub, a software carpentry course was completed [4]. In this course, the first lesson was the concept of version control based on checkpoints, as well as conflict resolution with two differing outputs. After setting up the account and setting the basic text editor, nano in our case, we finished up the config.

The first task was to set up a repository and create basic text files to track, manipulate and inspect these changes. The best practices regarding Git commit messages were explained. The concept of the staging area was explained using the example of multiple receipts. While simple in theory, the `.gitignore` file is very important for this project, as we only wanted to track the most important metrics and the section was studied in detail. Afterwards a remote repository was created in GitHub, as this project was not being worked on by multiple people, best practices regarding branching and pushing to main were not explored. For this project a trunk-based branching path was chosen.

When creating and testing various models based on different data subsets, parameters or algorithms, a lot of different models, all with specific performance metrics, will be created. To not accidentally toss a well-performing model or rely on tracking hundreds of small files via GitHub, MLflow was introduced [5]. It allowed fully local storage of any models, including the specific parameters, metrics and environment they were created in. The tracking for the models is highly customizable and manual tracking was preferred over the easy-to-implement automated tracking provided within the documentation. This allowed us to specify to only track the parameters differing from the default parameters, as seen in the documentation of each method in Scikit-learn and the parameters that reflect model performance. The custom parameter and metric tracking allowed for standardized comparison between models by using a confusion matrix involving Accuracy, Recall, Precision and F1-score. These metrics could be visualized in a comprehensive graph view within the locally hosted UI of MLflow, allowing for model evaluation and comparison immediately.

The primary issue across the project was adjusting the tracking for the different purposes, as for parameter tuning, not every run was relevant to keep track of, simply the best one or two parameter configurations were required. With the initial automated tracking that was implemented, a few limited grid search runs would create more than 20 models with almost exact outputs, cluttering the graphs, as well as around 200 files for GitHub to keep track of, as the parameters used weren't accounted for via the `.gitignore` file yet. When the tuning was finished, the best-performing model of each used algorithm was selected and logged internally. This selection process was possible due to the evaluation function MLflow provides, as for some of the initial tuning runs the confusion matrix was not tracked. For proper comparison between all the models, the best-performing grid search models were manually evaluated with the evaluate function MLflow provides. The evaluate function generates not only the confusion matrix for each model but also other insights like ROC-AUC, providing additional detail with which to select the best-performing models.

2.3 Scikit learn

Scikit-learn provides all the algorithms used for this project. For each algorithm used, it provides in-depth documentation and explanation about parameters and attributes used to adjust, analyze and improve the results, as well as basic examples of code and their output. The most important sections for this work consisted of 1.10 "Decision Trees", 1.11 Ensembles: "Gradient boosting and random forests, bagging, voting, stacking" and the whole of section 3: Model selection and evaluation. The latter being used to learn how to evaluate the created models [6].

2.4 Decision tree

The decision tree is a simple, yet powerful classification algorithm [7]. Due to its simplicity, it was chosen as the first model algorithm. It attempts to categorize data based on decision splits, which are based on specific attributes found within the dataset. These splits are referred to as branches, splitting the dataset into nodes with a certain purity. The purity of a node reflects the homogeneity of data characteristics of the node. This continues until the tree reaches end nodes, so-called "leaves", with maximum purity. A leaf with a purity of 1 contains only one specific result or label.

This method of looking for a pattern leading to the purest leaf often leaves to overfitting. That is, it will specialize on the seen data, as it accounts for all the noise or outliers to achieve good results in training with near-perfect accuracy, but will struggle to handle previously unseen data.

For this reason, a method called "pruning" is used to overcome the issue of overfitting. By adjusting certain parameters in the decision tree function, provided by scikit-learn, we can ensure that the tree doesn't develop too many leaves, "cut" off the ones that do not impact the final score too much and are most likely based on noise, or ensure that it doesn't branch up to infinity (or until all leaves are homogeneous).

Another aspect that was considered for the decision tree is removing the provided set split feature altogether and scrambling the train/validation sets in different ways via k-fold cross validation to systematically split the data into smaller subsets and train based on withholding one of these sets. The specific implementation of the k-fold method was provided by scikit-learn [8].

2.5 Random Forest

The Random Forest is an expansion of the decision tree, it tries to solve the prominent issue of overfitting a single tree by using a multitude of, often weaker, trees. These come to a final prediction via majority vote [9].

The most important factor for reducing bias is a function called bootstrapping. Training every tree on the exact same data, even accounting for randomness in training and weak learners, will lead to biased/converging trees. Weak learners in this context are decision trees that do not achieve high accuracy, that can be by limiting their depth (i.e., how many branches they are allowed to create) or heavy pruning. By splitting the training data into smaller subsets, done in a process called "bagging", each tree ends up vastly different from one another. While the individual trees might be overfitting or have a bias, across the entirety of the forest these issues are supposed to balance themselves out.

Two approaches tested for this were a bigger forest with weak learners (100+ trees, max depth 3-10) as the default approach to this problem. The other approach was using a smaller forest (20 trees, no depth limit) with strong learner trees (20 trees, no depth limit) to see if it could compete with the larger forest.

2.6 XGBoost

The Extreme Gradient Boosting algorithm is also an ensemble method, similar to the random forest and based on decision trees. Originally seen in Greedy Function Approximation: A Gradient Boosting Machine, by Friedman [10] and then further developed and enhanced for use by Tianqi Chen [11].

Instead of training multiple trees at once, unlike the Random Forest, XGBoost trains multiple weak trees sequentially to correct the mistakes of the previous trees using something called gradient descent. This process involves calculating the log loss function from the residuals, the difference between the predicted and actual values. To minimize this function, different parameters are scored with changed weights to create a strong classifier.

An integral factor in determining the model's performance is the learning rate, as it scales the step length of the gradient descent procedure, which means it influences the impact of each individual tree. To still ensure sufficient learning, the number of iterations is usually increased to account for a reduced learning rate. Combined with bagging, or subsampling as it's called for XGBoost, it can provide a substantially improved result as shown by T. Hastie, R. Tibshirani and J. Friedman, "Elements of Statistical Learning Ed. 2", Springer, 2009 [12].

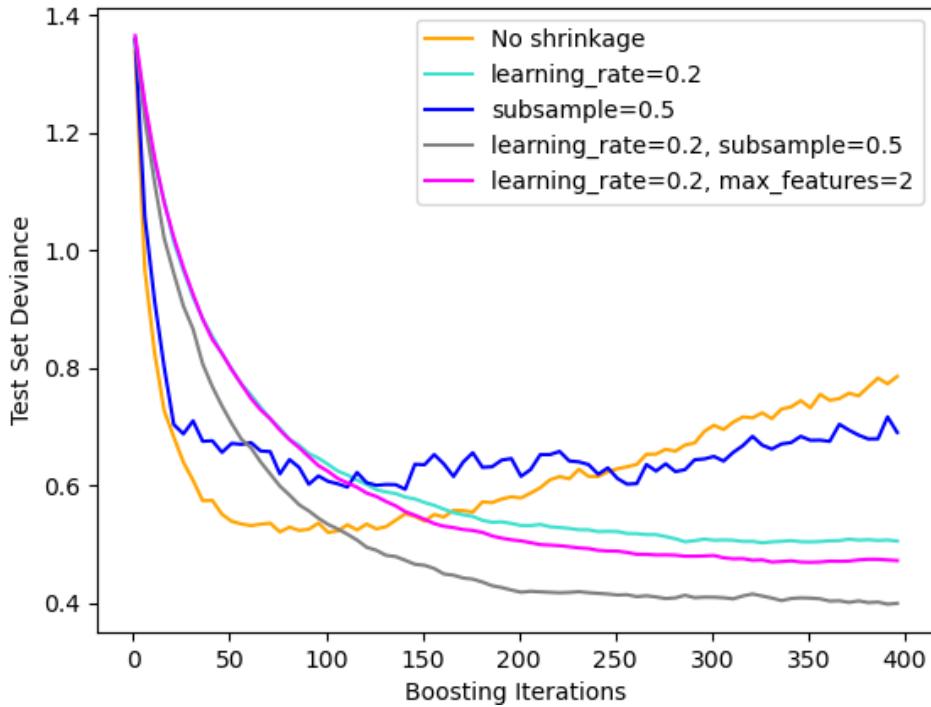


Figure 1: Influence of learning rate and subsampling in multiple example runs of the GradientBoostingClassifier.

Another important feature of XGBoost is early stopping. This method helps to determine the optimal number of iterations to build a model that is not overfitted or exhibits a great amount of bias. Early stopping occurs when the model's performance on the validation set plateaus or worsens across a given number of iterations.

Scikit-learn provides a visualized example of early stoppage and its impact on training and validation errors as well as training time.

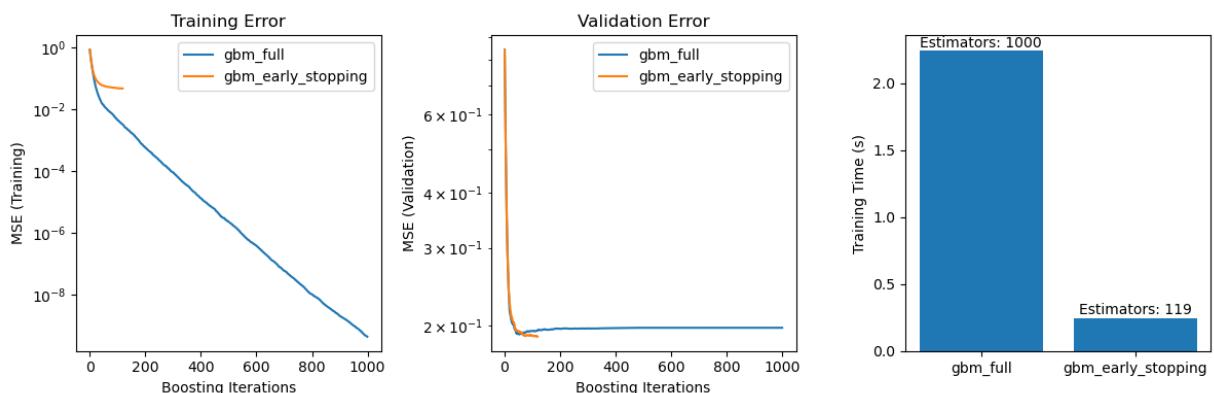


Figure 2: Comparison of validation error and training time in two GradientBoostingClassifiers, running with (orange) and without (blue) early stopping enabled.

2.7 Training, Optimization and Evaluation

2.7.1 Model A: Decision tree

The initial model utilized the default settings of the function provided by scikit-learn and included all the tabular data without any adjustments to establish a baseline. Provided training and validation sets were not considered at this point and a simple 80/20 random split was introduced.

Following this, multiple features deemed not relevant or potentially damaging to the training process were dropped and the given splits into train and validation sets were considered.

The third and final version of the decision tree model used a five-fold GridSearchCV [13] to systematically check for the best parameter combination out of the given values. From this point onwards, random state 42 was introduced as a constant to keep the model consistent and replicable, removing another layer of randomness that could affect results in either way.

Table 1: Parameters included in grid search for DecisionTreeClassifier, with the specific values in brackets.

Parameter (value)	Function
Criterion (gini, entropy)	Measuring the quality of a split based on Gini impurity or shannon information gain.
max_depth (none, 10, 20, 30)	Maximum number of splits the tree is allowed to make. If none, trees will continue splitting until all leaves are pure, or other parameters prevent splitting.
min_samples_split (2, 5, 10)	The minimum number of samples required to split a node.
min_samples_leaf (1, 2, 4)	The minimum number of samples required to be contained within a leaf node, forces every split to at least contain (min_number_leaf) on both sides
min_weight_fraction_leaf (0.0, 0.1, 0.2)	The minimum fraction of samples required at a leaf node, if weight is provided, equal weight is assumed
random_state (42)	Choose the seed determining the randomness of the method. Used to ensure replicable results each time.
max_leaf_nodes (None, 10, 20, 30)	Grow the best tree using the specified number of leaves. Best tree is determined by relative reduction in impurity
min_impurity_decrease (0.0, 0.1, 0.2)	Nodes split must decrease impurity by as much or more.
ccp_alpha (0.0, 0.1, 0.2)	Parameter for minimal Cost-Complexity Pruning. Subtree with largest cost complexity smaller than ccp_alpha will be chosen.

2.7.2 Model B: Random Forest

The initial random forest used the same parameters that were determined to be best for the Decision Tree, with a low estimator number of 20 and a ‘win_weight_leaf’ of 0.1.

Table 2: Parameters included in grid search for the RandomForestClassifier, with the specific values in brackets. Parameters with only data types in brackets were not part of the grid search but highlighted as they are unique to their algorithm.

Parameter (value)	Function
Criterion (gini)	Measuring the quality of a split based on Gini impurity or shannon information gain.
max_depth (none, 10, 20, 30)	Maximum number of splits the tree is allowed to make. If none, trees will continue splitting until all leaves are pure, or other parameters prevent splitting.
min_samples_split (2, 3, 4)	The minimum number of samples required to split a node.
min_samples_leaf (1, 2, 4)	The minimum number of samples required to be contained within a leaf node, forces every split to at least contain (min_number_leaf) on both sides
min_weight_fraction_leaf (0.0, 0.1, 0.2)	The minimum fraction of samples required at a leaf node, if weight is provided, equal weight is assumed
random_state (42)	Choose the seed determining the randomness of the method. Used to ensure replicable results each time.
max_leaf_nodes (None, 10, 20, 30)	Grow the best tree using the specified number of leaves. Best tree is determined by relative reduction in impurity
min_impurity_decrease (0.0, 0.01, 0.1)	Nodes split must decrease impurity by as much or more.
ccp_alpha (0.0, 0.0015, 0.01, 0.1)	Parameter for minimal Cost-Complexity Pruning. Subtree with largest cost complexity smaller than ccp_alpha will be chosen.
n_estimators (20, 100)	The number of trees trained for the ensemble. The prediction is determined by a collective vote of all estimators
bootstrap (bool)	Enables/Disables bootstrapping. If true, a subset of the dataset is used to train each tree, reducing bias.

2.7.3 Model C: Gradient boosting

The first iteration of the GradientBoostingClassifier used default settings with no subsampling, early stoppage and a low learning rate.

Due to the number of variables to consider for each parameter, the grid search had to be done in batches, with the best-performing parameters being used in the next batch. Some parameters were excluded from testing, as they were shown to have a limited impact on the previous optimizations.

The options of early stoppage and warm starts were not considered because of this.

The focus of the grid search was to optimize the Gradient Boosting regularization and the tree itself.

Table 3: Parameters included in grid search for the GradientBoostingClassifier, with the specific values in brackets. Parameters with only data types in brackets were not part of the grid search but highlighted as they are unique to their algorithm.

Parameter (value)	Function
loss (log_loss)	The log loss refers to the binomial deviance in this classification problem. It's the loss function that is being optimized.
learning_rate (0.1, 0.2, 0.5)	Dictates the impact of each individual tree on the loss function. Combined with subsampling, it can be used for regularization.
n_estimators (20, 100)	Number of iterative boosting stages to perform. Due to robustness against overfitting, higher numbers typically lead to better results.
subsample (0.5, 0.8, 1.0)	The fraction of samples used to train each boosting stage. Any value below 1 will result in Stochastic Gradient Boosting, reducing variance while increasing bias.
criterion (friedman_mse)	Measuring function of split quality using mean squared error with improvement score by Friedman.
warm_start (bool)	Enables reuse of a previous solution as a starting point, adding more estimators to the ensemble.
validation_fraction (float)	Only used in case of early stoppage. Fraction of training set, set aside for validation in case of stoppage.
n_iter_no_change (int)	Parameter deciding early stoppage when validation scores across iterations do not improve. When set to none, no early stoppage will occur.
tol (float)	Tolerance determining the minimum improvement for the validation score across n_iter_no_change.

3. Results

Model A.1, using unadulterated inputs with no parameter modifications, achieved an F1-score of 0.53 and a balanced accuracy of 0.65. The F1-score indicates a prediction only minimally better than random chance.

After removing excess data in model A.2, an increased F1-score of 0.57 and a balanced accuracy of 0.57 were achieved.

Performing a grid search to find optimal parameter configurations yielded model A.3 with an F1-score of 0.75 and a balanced accuracy of 0.76. The only change compared to A.2 was the inclusion of the parameter "min_weight_leaf" of 0.1 compared to 0.0.

For model B.1, the initial settings of the Random Forest included the previously determined min_weight_leaf of 0.1 and a smaller estimator size of 20 estimators. This model achieved results closely matching model A.3 with an F1-score of 0.75 and a balanced accuracy of 0.76.

The grid search for the random forest was split into a smaller forest of strong learners (model B.2a) with:

```
('min_weight_fraction_leaf': [0.1], 'n_estimators': [20], 'max_depth': [None, 10, 20, 30], 'ccp_alpha': [0.0, 0.005, 0.01, 0.0015])
```

and a larger forest of weaker learners (model B.2b) with:

```
'min_samples_split': [2, 3, 4],  
'min_weight_fraction_leaf': [0.0, 0.01, 0.1],  
'min_impurity_decrease': [0.0, 0.01, 0.1],  
'max_leaf_nodes': [None, 10, 20]
```

Model B.2a achieved better results with an F1-score of 0.74 and a balanced accuracy of 0.76.

Model B.2b performed worse with an F1-score of 0.70 and a balanced accuracy of 0.72.

The first Gradient Boosting model C.1 used the same modified data and default parameters, resulting in an F1-score of 0.70 with a balanced accuracy of 0.70.

Multiple grid searches combined, while keeping the best determined parameters, resulted in model C.2, these stayed the same as the model using default settings and achieved an F1-score of 0.70 with a balanced accuracy of 0.70.

Model B.2b was chosen for evaluation with the hold-out set that was not used in either training or validation in any models. Resulting in an F1-score of 0.70 with a balanced accuracy of 0.70.

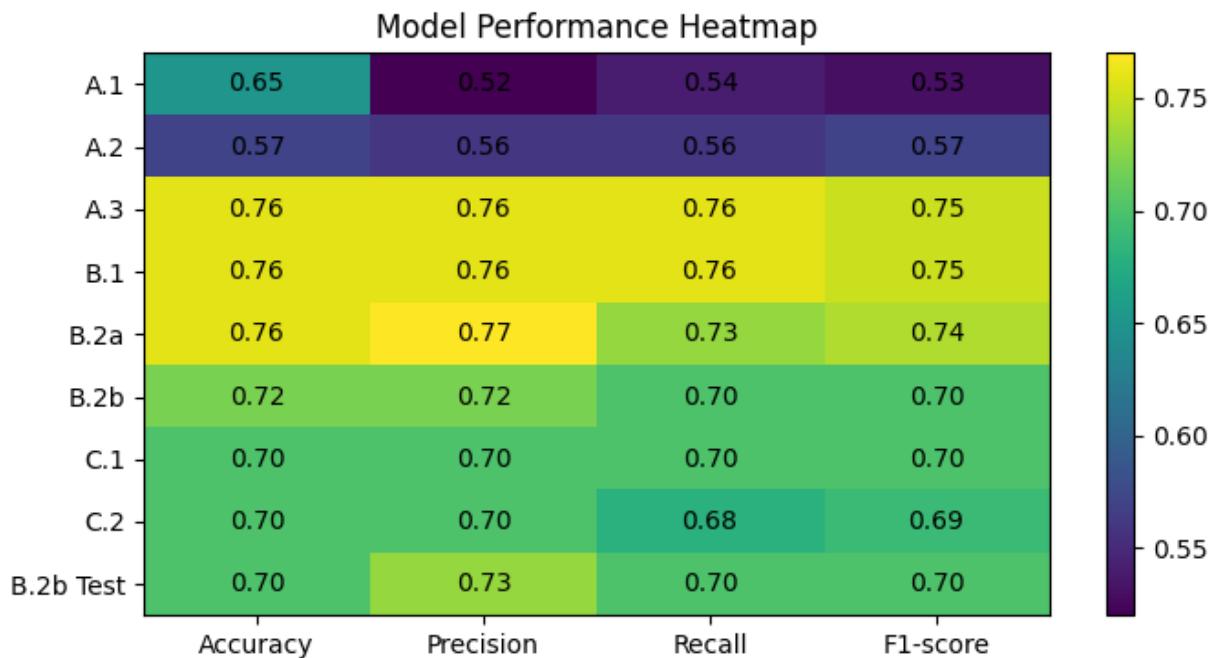


Figure 3: Heatmap of each model's performance. Where yellow/green equal better performance, blue/purple equals worse performance.

4. Discussion

This work's aim was to develop a traditional machine learning model that could predict PC recurrence from only the metadata obtained from patients coming in for re-staging PSMA-PET/CT scans.

Starting from the simplest algorithm with a naive approach utilizing no data trimming, which resulted in mostly random predictions, with an accuracy above expectation. This was due to the inclusion of primary staging patients and some patients coming in for multiple scans. In most cases, these were cancerous, leading to bias. To remedy this, any primary staging as well as patient identifiers were removed, while respecting the set splitting mask, leading to a worse result in Accuracy and consequently F1-score, while slightly improving Precision and Recall.

Interestingly, even after extensive testing, only one parameter seemed to have a meaningful impact on metrics. "min_weight_fraction_leaf" of 0.1 solved the issue of overfitting and filtering out noise. Including or excluding the parameter change from subsequent grid searches showed other parameter combinations that performed better than the default settings but could not reach the combination of default settings with a minimum fraction.

Creating the next models based on a random forest algorithm, but keeping the previous findings in mind. The initial small forest seemed to have mostly copied the individual trees previously made, as the results only started to differ past the third decimal point. Considering the first attempt at a forest was built around a smaller forest with strong learners, this outcome could have been predicted. The parameter tuning that followed was split into two approaches. The approach to refine the "robust" forest showed no improvement over the previous model. While expected, this confirms that the additional parameters available for tuning in a forest model did not affect decision-making.

Model B.2b was chosen for evaluation on the test set due to its larger forest. The concern with better-performing models like B.2a was its small forest size with complex trees, possibly leading to bias. The performance on the test set was overall worse by only a few percent compared to the validation result.

As the most complex and final model, gradient boosting was expected to perform the best or at least similarly to all the previous models. The results, however show a 5% to 10% decrease in performance for the F1-score, in the recall metric, the performance drop is the most significant. While, with the given data, a singular tree created via iterative learning might not have been better than the previous models using a multitude of trees or a single fine-tuned tree, this result was unexpected. A cause for this might be an insufficient exploration of the parameters available for this algorithm. Due to the large quantity of parameters, only a selected number of "more important" parameters were chosen and had to be run in grid search batches, as the exhaustive exploration using the previous method would have taken around 3 years based on rough estimations (using the university's provided Lenovo

Thinkpad). In favor of learn set shrinkage and learn rate reduction, which were explained above, the concept of early stoppage has been neglected here. Early stopping could have allowed for more extensive parameter testing, as it can significantly reduce the train time for most runs. Other ways to more extensively test parameter combinations for gradient boosting could have been Bayesian search instead of the exhaustive grid search or the third-party framework OPTUNA. These issues were brought up and considered but deemed outside the scope of this project. There may be a configuration not explored with better performance.

The rate of true positives could be seen as the clinically most significant metric, as it dictated the rate of actual cancer diagnoses. When comparing the model's confusion metrics, it becomes apparent that the sensitivity is significantly higher than the recall, often balancing out the data. After running a Spearman's rank correlation using Seaborn, the weak correlation between age, PSA levels and label (all calculated separately) becomes apparent. The more significant finding was the comparatively stronger negative correlation between prostatectomy status and label.

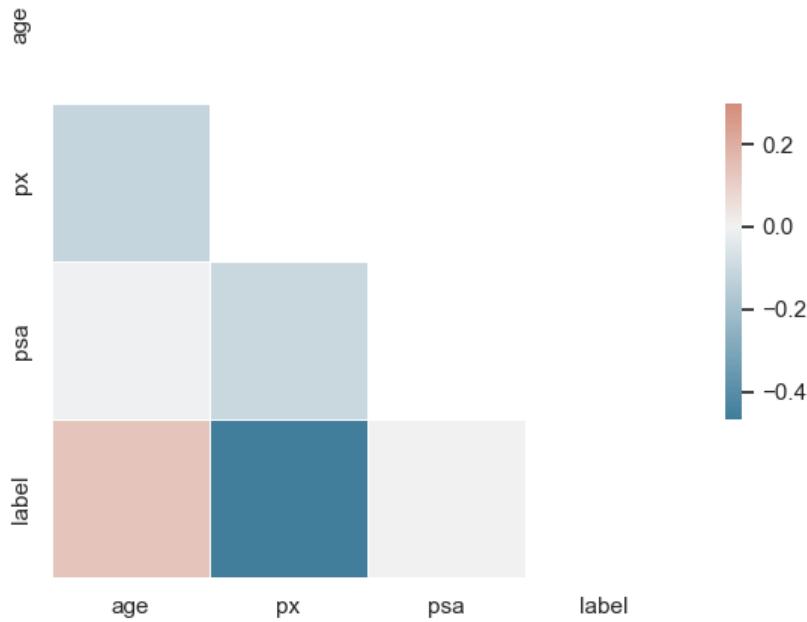


Figure 4: Visualization of Spearman correlation test between the parameters used in training and the label. Marked by a color gradient where red is positive and blue is negative, higher color intensity correlates to stronger Spearman correlation.

This, combined with the low amount of features processed in training, would be an explanation as to why all models across the board struggle with recall while still achieving acceptable numbers compared to the original deep learning model. This seems to be an issue of the original model by Korb et al., as in the final test it achieved only a 56.7% rate of correct predictions for true positives but an 88% rate for true negatives, the previous versions also have a heavy bias towards true negatives. Reinforcing our belief that prostatectomy status plays a heavy role in prediction.

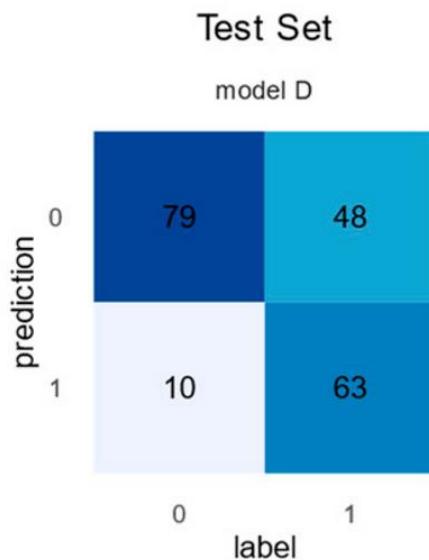


Figure 5: Results of model D, created by Korb et al. when predicting the previously withheld test set. Columns are labels, and rows are predictions. The numbers within the cells are additionally color coded with a gradient from light blue (small numbers) to dark blue (large numbers)

With complex problems like cancer detection, statistical methods will run into a hard limit for performance, also known as Bayes error rate. It describes a specific error limit for any classification problem due to outliers. This in turn means that, no matter how complex and detailed our model is, it won't be able to reach perfect/near-perfect performance. While we don't believe to have reached the overall limit for cancer detection, as shown in other papers. With the current data there may be a hard limit we are not able to surpass [14].

5. Conclusion

After testing multiple algorithms with various parameter configurations and complexities, it seems that despite reaching similar performance to the deep learning model used for comparison, we were unable to reach the same or better performance with the available data.

Considering the small amount of tabular data used to train the models, there is potential to increase performance by carefully selecting and determining significant image features to enter training. These results are in line with the expected outcome of the supervisor and show that, while there are still some potential improvements to be made using classical ML methods, the usage of a DL model for the initial research was not a bad choice.

6. Sources

1. Bray, F.; Ferlay, J.; Soerjomataram, I.; Siegel, R.L.; Torre, L.A.; Jemal, A. Global Cancer Statistics 2018: GLOBOCAN Estimates of Incidence and Mortality Worldwide for 36 Cancers in 185 Countries. *CA. Cancer J. Clin.* **2018**, *68*, 394–424, doi:10.3322/caac.21492.
2. Korb, M.; Efetürk, H.; Jedamzik, T.; Hartrampf, P.E.; Kosmala, A.; Serfling, S.E.; Dirk, R.; Michalski, K.; Buck, A.K.; Werner, R.A.; et al. Detection of Local Prostate Cancer Recurrence from PET/CT Scans Using Deep Learning. *Cancers* **2025**, *17*, 1575, doi:10.3390/cancers17091575.
3. Arita, Y.; Roest, C.; Kwee, T.C.; Paudyal, R.; Lema-Dopico, A.; Fransen, S.; Hirahara, D.; Takaya, E.; Ueda, R.; Ruby, L.; et al. Advancements in Artificial Intelligence for Prostate Cancer: Optimizing Diagnosis, Treatment, and Prognostic Assessment. *Asian J. Urol.* **2025**, doi:10.1016/j.ajur.2024.12.001.
4. Version Control with Git: Summary and Setup Available online: <https://swcarpentry.github.io/git-novice/> (accessed on 16 December 2025).
5. Scikit-Learn with MLflow | MLflow Available online: <https://mlflow.org/docs/3.1.3/ml/traditional-ml/sklearn/guide/> (accessed on 16 December 2025).
6. 1. Supervised Learning Available online: https://scikit-learn/stable/supervised_learning.html (accessed on 16 December 2025).
7. Quinlan, J.R. Induction of Decision Trees. *Mach. Learn.* **1986**, *1*, 81–106, doi:10.1007/BF00116251.
8. An Analysis of Model Evaluation with Cross-Validation: Techniques, Applications, and Recent Advances Available online: https://www.researchgate.net/publication/383955409_An_Analysis_of_Model_Evaluation_with_Cross-Validation_Techniques_Applications_and_Recent_Advances (accessed on 16 December 2025).
9. Breiman, L. Random Forests. *Mach. Learn.* **2001**, *45*, 5–32, doi:10.1023/A:1010933404324.
10. Friedman, J.H. Greedy Function Approximation: A Gradient Boosting Machine. *Ann. Stat.* **2001**, *29*, 1189–1232, doi:10.1214/aos/1013203451.
11. Chen, T.; Guestrin, C. XGBoost: A Scalable Tree Boosting System. In Proceedings of the Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; ACM: San Francisco California USA, August 13 2016; pp. 785–794.

12. James, G.; Witten, D.; Hastie, T.; Tibshirani, R. Tree-Based Methods. In *An Introduction to Statistical Learning: with Applications in R*; James, G., Witten, D., Hastie, T., Tibshirani, R., Eds.; Springer US: New York, NY, 2021; pp. 327–365 ISBN 978-1-0716-1418-1.
13. Ogunsanya, M.; Isichei, J.; Desai, S. Grid Search Hyperparameter Tuning in Additive Manufacturing Processes. *Manuf. Lett.* **2023**, *35*, 1031–1042, doi:10.1016/j.mfglet.2023.08.056.
14. James, G.; Witten, D.; Hastie, T.; Tibshirani, R. Statistical Learning. In *An Introduction to Statistical Learning: with Applications in R*; James, G., Witten, D., Hastie, T., Tibshirani, R., Eds.; Springer US: New York, NY, 2021; pp. 15–57 ISBN 978-1-0716-1418-1.