Practical Work P2

Data Mining GEI, Q2 2024-2025

Diabetes classifying based on health attributes

Alex Beauchamp

Otman Ezzayat Maid

Joan Gómez Català

Matija Jakovac

Álvaro Monclús Muñoz

Delivery date : 26th of May 2025

Index

Description of the original data

Description of data matrix

The data was sourced from Kaggle which consists of the results of a survey from the 2015 Behavioral Risk Factor Surveillance System (BRFSS) published by the user Alex Tabour, who cleaned the data removing any missing entries. The dataset actually consists of three subsets of data, but in this study only the second one was used. The file used is also a subset of the first file with 70,692 entries focused on diabetes-related health indicators and selected with an equal class distribution between diabetic (including prediabetic) and non-diabetic individuals (35.346 each type).

Link: https://www.kaggle.com/code/alexteboul/diabetes-health-indicators-dataset-notebook

Description of metadata

Each row in the dataset represents the health profile of a user who participated in the survey. There are 22 columns in total, the diabetes target column (says if the user has Diabetes or not) and 21 features which combine general and health-related information about the user, such as BMI, smoking status, alcohol consumption, physical activity, sleep duration, and the presence of conditions like high blood pressure and high cholesterol. These features are mostly boolean, due to the preprocessed tasks done by the user who published it, but we also have some qualitative and numerical features. In general, the dataset is well-suited for classification problems aimed at identifying individuals at risk for developing diabetes.

MetaData Table

Variable	Short Variable Name	Modalities	Short Mod Name	Meaning	Type	Range	Role
Diabetes_binary	Diabetes			Diabetes status. (0 = No diabetes, 1 = Prediabetes, 2 = Diabetes)	Quali	[0, 2]	Expl.
HighBP	HighBP			Has high blood pressure	Bool		Expl.
HighChol	HighChol			Has high cholesterol	Bool		Expl.
CholCheck	CholCheck			Had a cholesterol check in past 5	Bool		Expl.
BMI	BMI			Body Mass Index	Num	[12, 98]	Expl.
Smoker	Smoker			Smoked 100 cigarettes in lifetime	Bool		Expl.
Stroke	Stroke			Ever had a stroke	Bool		Expl.
HeartDiseaseorAttack	HeartCond			Coronary heart disease or myocardial infarction	Bool		Expl.
PhysActivity	PhysAct			Physical activity in past 30 days (excluding job)	Bool		Expl.
Fruits	Fruits			Consumes fruit ≥1 time per day	Bool		Expl.
Veggies	Veggies			Consumes vegetables ≥1 time per day	Bool		Expl.
HvyAlcoholConsump	HeavyDrink			Heavy alcohol consumption (14+/7+ drinks per week)	Bool		Expl.
AnyHealthcare	HasHealthcare			Has any health care coverage	Bool		Expl.
NoDocbcCost	NoDocCost			Could not see doctor due to cost (past 12 months)	Bool		Expl.

GenHlth	GenHlth			General health status. (1 = Excellent, 2 = Very good, 3 = Good, 4 = Fair, 5 = Poor)	Quali	[1, 5]	Expl.
MentHlth	MentHlth			Poor mental health days (past 30 days)	Num	[0, 30]	Expl.
PhysHlth	PhysHlth			Physical illness/injury days (past 30 days)	Num	[0, 30]	Expl.
DiffWalk	DiffWalk			Difficulty walking or climbing stairs	Bool		Expl.
Sex	Sex			Sex of user	Bool		Expl.
Age	Age			Age group (13 levels). (1 = [18, 24], 2 = [25,29], ...(5 year intervals), 13 = 80 years old or older)	Quali	[1, 13]	Expl.
Education	Education			Education level (6 levels)	Quali	[1, 6]	Expl.
		Never attended school or only kindergarten	1				
		Grades 1–8	2				
		Grades 9–11	3				
		Grade 12 or GED	4				
		Some college or technical school (no degree)	5				
		College graduate	6				
Income	Income			Income category (8 levels). (1 = <$10,000, 2 = [10,000,15,000), ...(5,000$ intervals), 8 = $75,000 or more)	Quali	[1, 8]	Expl.

* Note that the metadata table is missing the *Measuring Unit*, *Missing Code*, and *Measurement Procedure* columns, as they were empty for all variables.

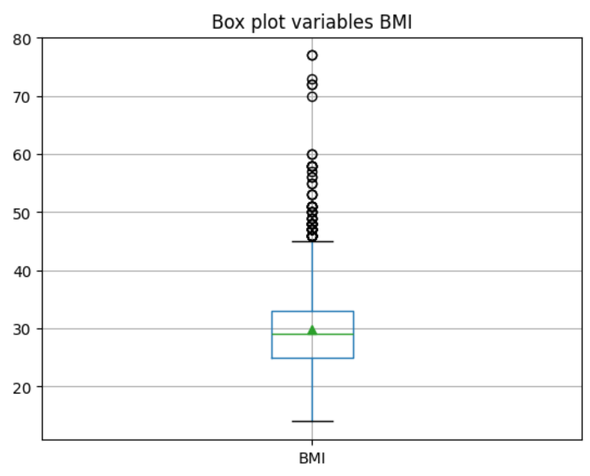
Description of pre-processing of data

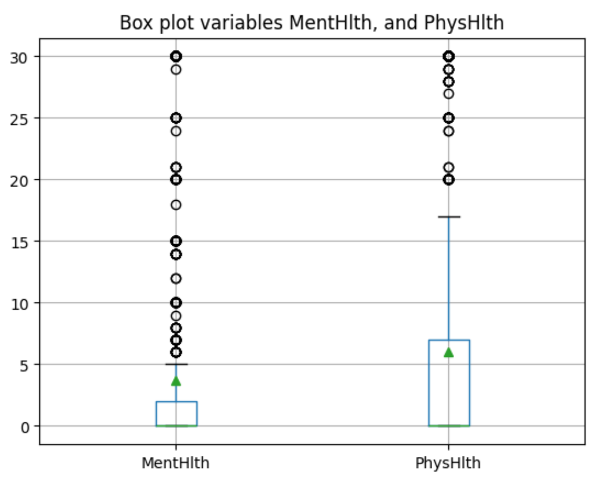
The first step of pre-processing the raw dataset was to select 2000 entries, 1000 each type, from the subset previously described. Reasoning behind selecting that number of entries is performance-wise due to faster training and less memory usage. Some methods such as SVM, k-NN and meta-learning algorithms could scale poorly with larger dataset and it is easier to examine results having less entries. The dataset remains balanced as there are 1000 entries of each type, 0 = no diabetes and 1 = diabetes. Those entries are selected **randomly** to avoid possible biased samples.

There were not any features that would be unnecessary in examining the data. Most common examples of that would be phone numbers, ID numbers, comments and they were not present in the dataset so we did not have to remove them. The data did not have any categorical variable as such variables were already transformed into numerical/boolean variables with values 0 and 1. Age and income which are usually numerical variables were also originally separated in categories described in meta-data.

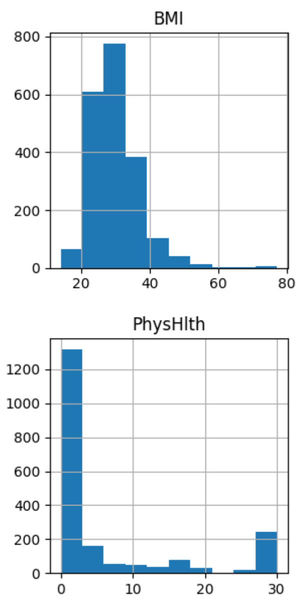
Several name columns were changed because they were too long and could be shortened without losing interpretation of the feature. Some of them were changed because they did not represent the feature as well as its substitution. Examples of changes are `Diabetes_binary` into `Diabetes`, `HvyAlcoholConsump into HeavyDrink`, …

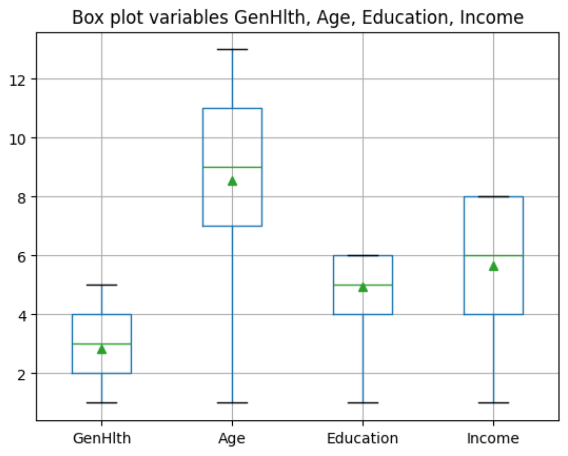
On the numerical variables and qualitative variables a univariate descriptive was performed to potentially observe possible outliers.





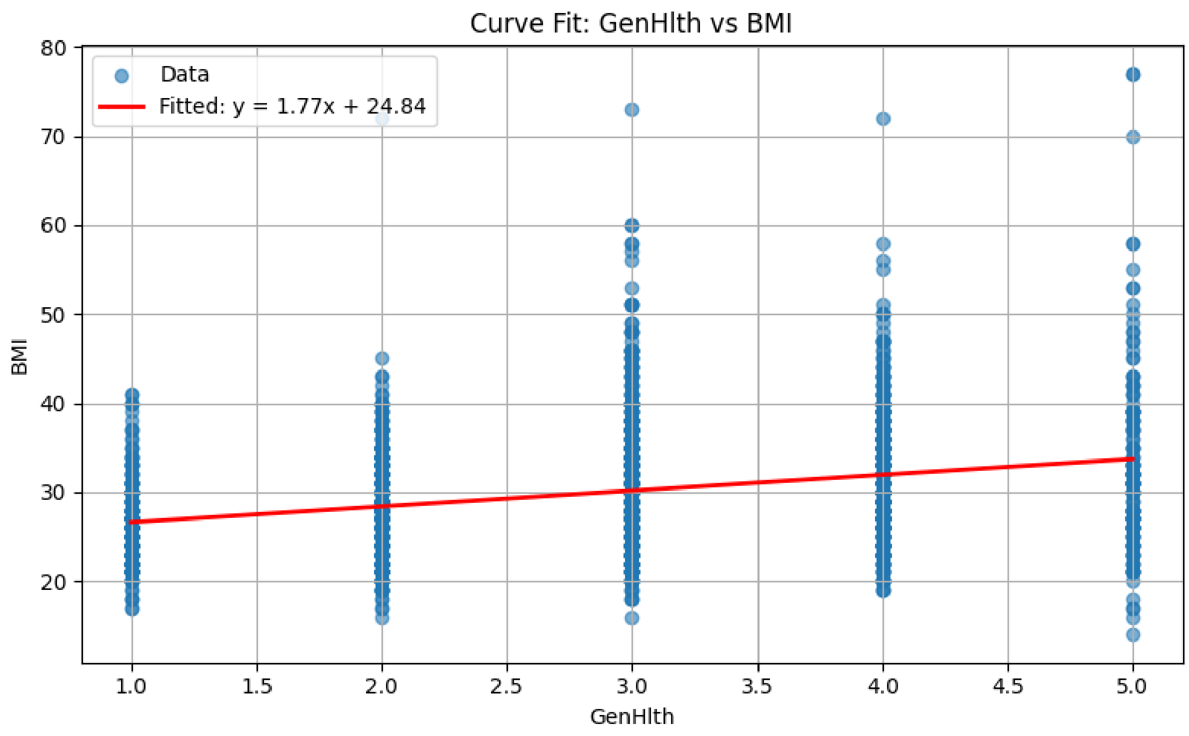
Observing the box-plots of BMI, MentHlth and PhysHlth we can clearly see there are some **candidates** that can be an outlier. Histograms of those variables give us the same conclusion.



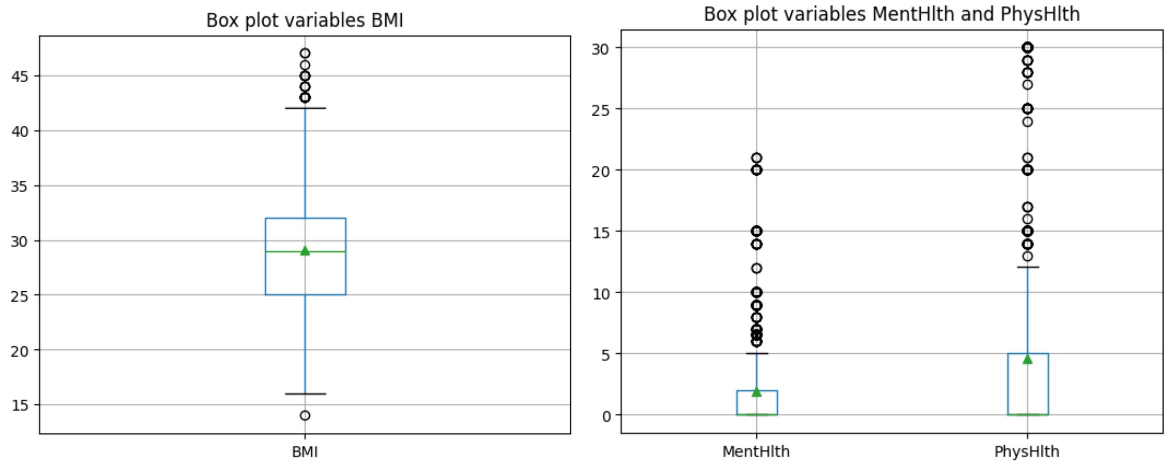


Observing the box plots, we can see that all values for the variables GenHlth, Age, Education, and Income lie within the interquartile range (IQR), with no visible candidates for outliers.

For BMI, MentHlth and PhysHlth outlier detection was performed using bi-variate analysis. Observing the feature we think has an outlier we use a highly correlated feature and apply the linear regression between those two variables. For linear regression, errors follow normal distribution as to why we can apply statistical tests on them. We considered an outlier in normal distribution if it falls in the regions $\mu - 2 * \sigma < x$ or $x > \mu + 2 * \sigma$.



All three variables have the highest correlation with GenHlth and linear regression was applied with it for all of them. After detecting which are the exact values and samples of outliers their substitution was performed by predicted value with linear regression.



Box plots plotted after substituting outliers clearly show that some values were detected as outliers. On the BMI plot there are not any values higher than 50 which is probably not possible for a regular person and those values could be the errors because of calculating weight in pounds. MentHlth box plot is also significantly different after substituting, also confirming there were some outliers in the dataset.

PhysHlth box plot does not look that much different but the same method was performed on the feature.

As there were not any missing values there was no need to perform the same process of substituting the missing values as for outlier values. Most variables are boolean or were already separated into categories so there was no need for normalization as there are no significantly large values in the dataset.

Evaluation criteria of data mining models - Alex

To keep the tests as independent from each other as possible, we have decided to separate the data into train / test right at the beginning so that each of them may use the exact same datasets for training and testing. We believe that doing so should allow us to better compare the accuracy values for each of them. We used a 70/30 split for training and testing, to ensure we have enough data for training while still retaining a sufficient amount for accurate test results.

The last thing to note is that *stratify=y* was used to ensure that the same percentage of each class would be represented in both sets. In our case, this meant that the train sets would contain 700 elements of each class and the test sets 300 elements of each class, since after preprocessing we have a balanced set of 1000 elements of each class.

Next, the main evaluation methods that will be used are accuracy, which is the proportion of correct predictions out of all predictions, and the f1-score, which is based on both recall and precision. It should be noted that since this is a health concern, more emphasis could have been put on recall (out of all the instances which did have diabetes, how many were correctly identified as such). This is because when it comes to health issues, our goal should always be to minimize the number of patients who have to be called back because they are actually diabetic (recall), at the possible detriment of the number of people who have to be told they are not actually diabetic. Since this was a general overview of analysis methods however, and since this health issue is generally not as worrisome as something like cancer, we have decided to stick with the more general methods mentioned earlier.

Finally, the cross validation sets were left for each algorithm to decide. This is because some of them needed more folds (require less data for training like finding the Naive-Bayes threshold) whereas some of them need less folds (more data necessary, like Naive-Bayes itself). Generally the number of folds was 10, but sometimes 5 or 20 was used if necessary. The Stratified cross validation was usually used to ensure a balanced class distribution within our folds.

Execution of different machine learning methods

Naive Bayes - Alex

First of all, the only hyperparameter we were able to identify is the decision threshold used. In the case of a (Gaussian, we have continuous values) Naive Bayes, the decision threshold means the threshold that must be reached for the answer to be considered true. For example, with a default value of 50%, the model must indicate a probability of at least 50% for it to classify something as a class. To decide on which threshold to use, we first separated the data into 20 folds for cross validation, training a model on each of them. Once the model is trained, we then tried each generated probability value as a potential and saved the one which had the highest f1-score for each fold. Once all the folds were done, we simply used the average of the maximum. At the very end, we compare the results with our new threshold against the default of 50%:

- Default: 72% accuracy, 72% average f1-score
- Threshold found: 73% accuracy, 72.5% average f1-score

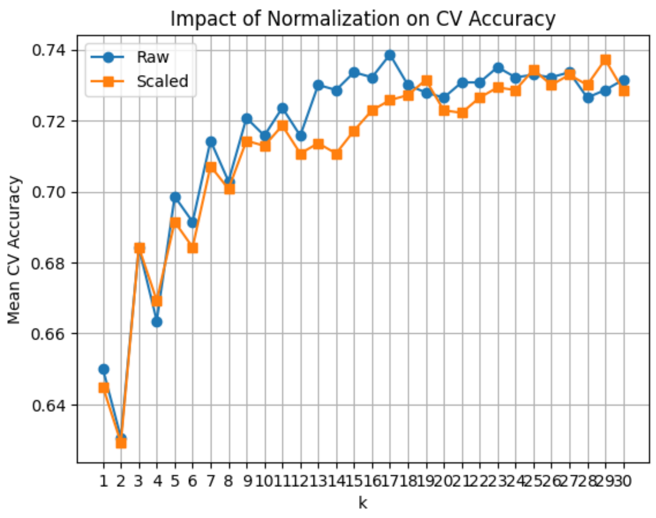
This is not a huge difference but might help if every percent counted, such as may be the case for health issues like diabetes.

These accuracy values, however, are still quite low. This can be attributed to the very essence of a Naive Bayes algorithm: the hypothesis of independence. Here, we have many health attributes, such as high blood pressure, high BMI, etc. One need not be a medical professional to say that these health attributes may very well be dependent on one another. This means that Naive Bayes may not be the most appropriate algorithm for this context.

As for the num ber of elements, this may also be a reason as to why the accuracy values stay relatively low. Indeed, though this may be a balanced (after preprocessing) dataset with 1000 elements of each class, it is important to consider that we are using continuous values. As such, even this number may not be quite large enough to properly represent each type of combination such as is required of Naive-Bayes.

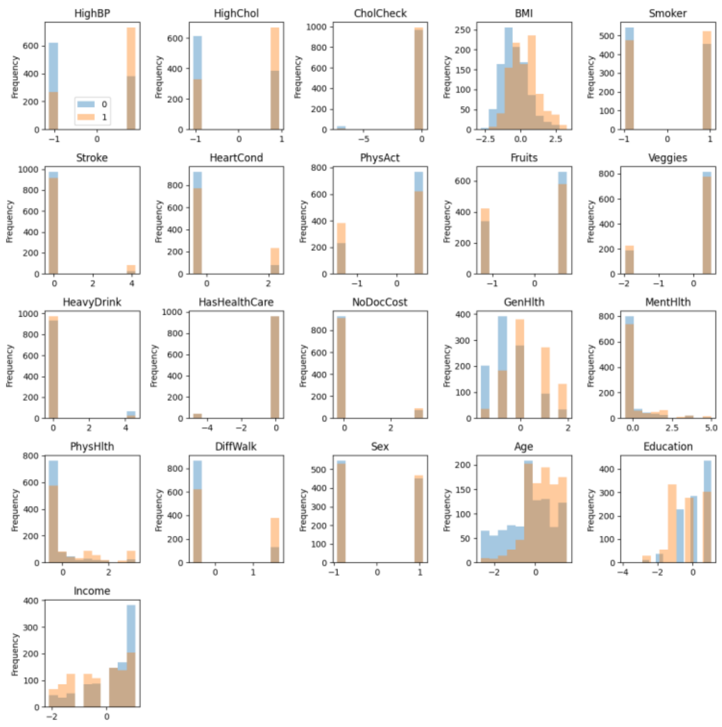
K-NN - Álvaro

The KNN analysis began by creating parallel training and test splits, both on the raw and standarized data in order to isolate the effect of normalization on our dataset. This initial setup allowed us to understand how scaling can affect the optimal k and the resulting cross-validation accuracy.



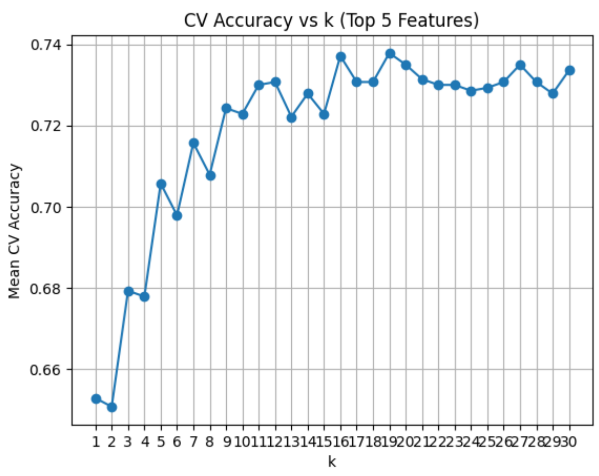
A CV sweep was done for k=1 to 30 to understand which value of k was the optimal for both measures. We found that CV accuracy was relatively similar for most values of k and followed the same trend. Because most variables are either binary (e.g. “HighBP, “Smoker”, “DiffWalk”, etc.) or are narrow integer counts (e.g. “Age”, “BMI”, “GenHlth”, etc.), scaling them to zero mean variance does not change the geometry of the point cloud. This means that normalization is irrelevant for our dataset, as it barely affects CV accuracy.

Because KNN’s distances can be highly influenced by irrelevant features, we next visualized every feature’s distribution in the following grid of histograms.



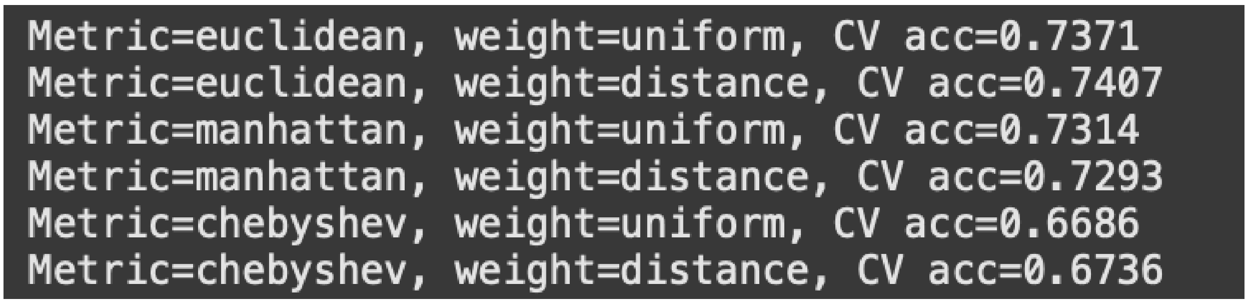
Several binary variables (e.g. “CholCheck”, “Smoker”, “Stroke), showed an almost complete overlab between the two classes, suggesting limited information to be extracted from them. To measure this in a more exact way, we ranked features by mutual information and graphed against CV accuracy. The resulting “Accuracy vs # Features” curve shows a substantial jump from 1 feature onwards with a plateau from features to the full 21, where it hovers between 0.70 and 0.72 accuracy for the most part.

Following these findings, the top 5 features were selected and the CV over K sweep was re-run.



The “CV Accuracy vs k (Top 5 features)” plot has a dramatic increase at k=9, where it remains somewhat stable for the rest. The actual peak is at k=16, with a marginal improvement among the rest of the values of k.

We also tested different distance weighting and metrics to find which offered the highest accuracy. We found the difference between euclidean and manhattan measures were irrelevant, with a slightly better performance of the euclidean metric. We also observed that these two marginally outperformed the chebyshev metric. Euclidean with distance weights remained our default as even though the improvement was marginal, it offered the best results.



```
Metric=euclidean, weight=uniform, CV acc=0.7371
Metric=euclidean, weight=distance, CV acc=0.7407
Metric=manhattan, weight=uniform, CV acc=0.7314
Metric=manhattan, weight=distance, CV acc=0.7293
Metric=chebyshev, weight=uniform, CV acc=0.6686
Metric=chebyshev, weight=distance, CV acc=0.6736
```

Finally, training a KNN with k=16 on the full normalized training set produced an accuracy of 0.7517 when using only the top 5 features and an accuracy of 0.7350 when using all features, showing little noise offered by the irrelevant data.

Decision Trees

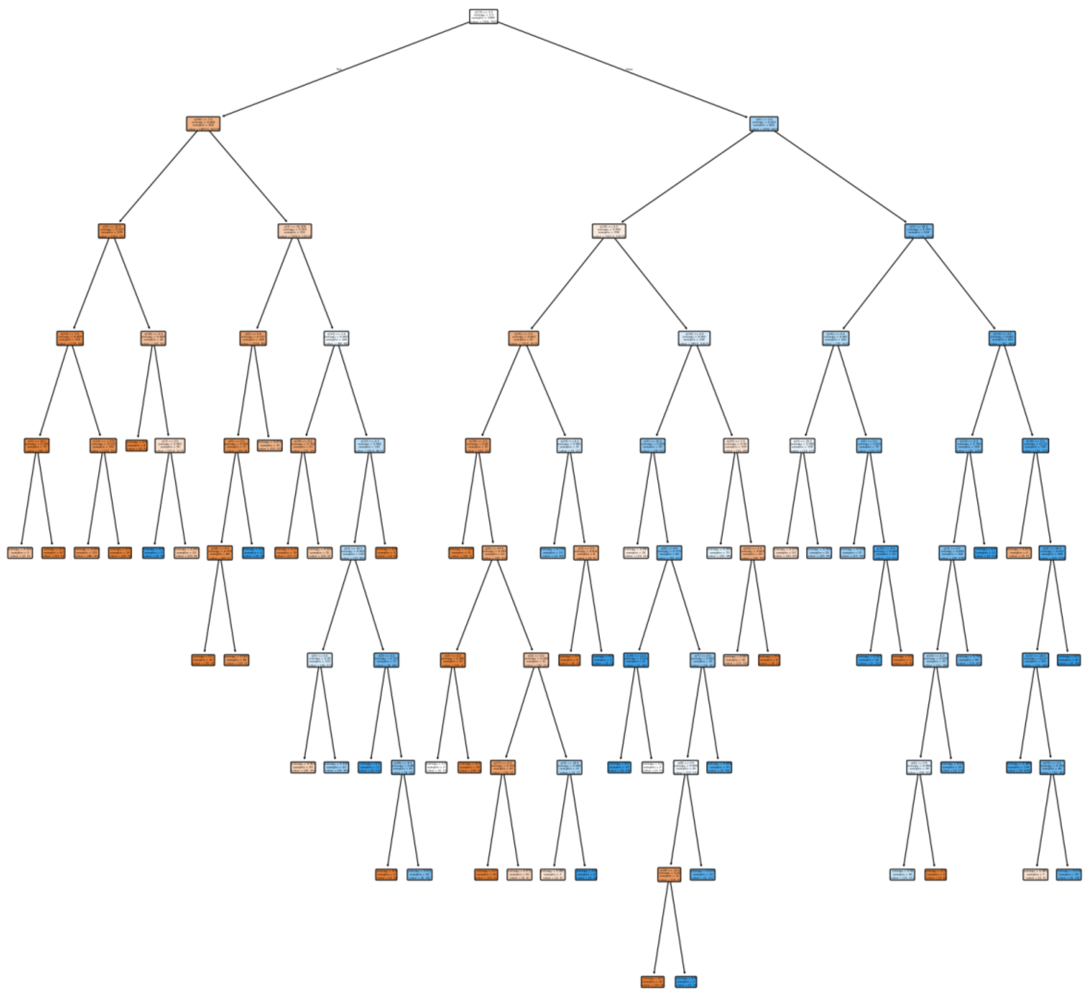
The choice of parameters could determine performance of the decision tree by increasing accuracy and other evaluation metrics. Easiest way to obtain a tree is by using *DecisionTreeClasifier* which has a lot of hyperparameters but three of them are most important and the ones we are going to use. Those three are ***criterion**, **min_samples_split** and **min_impurity_decrease***.

Criterion controls choosing the best split at each node and in every tree we are going to use “entropy” which is computed as $H = -\Sigma p_i * log_2 p_i$. It is used for information gain at every candidate and the split with the largest value of it is taken. It is chosen because it produces splits that maximize “purity”.

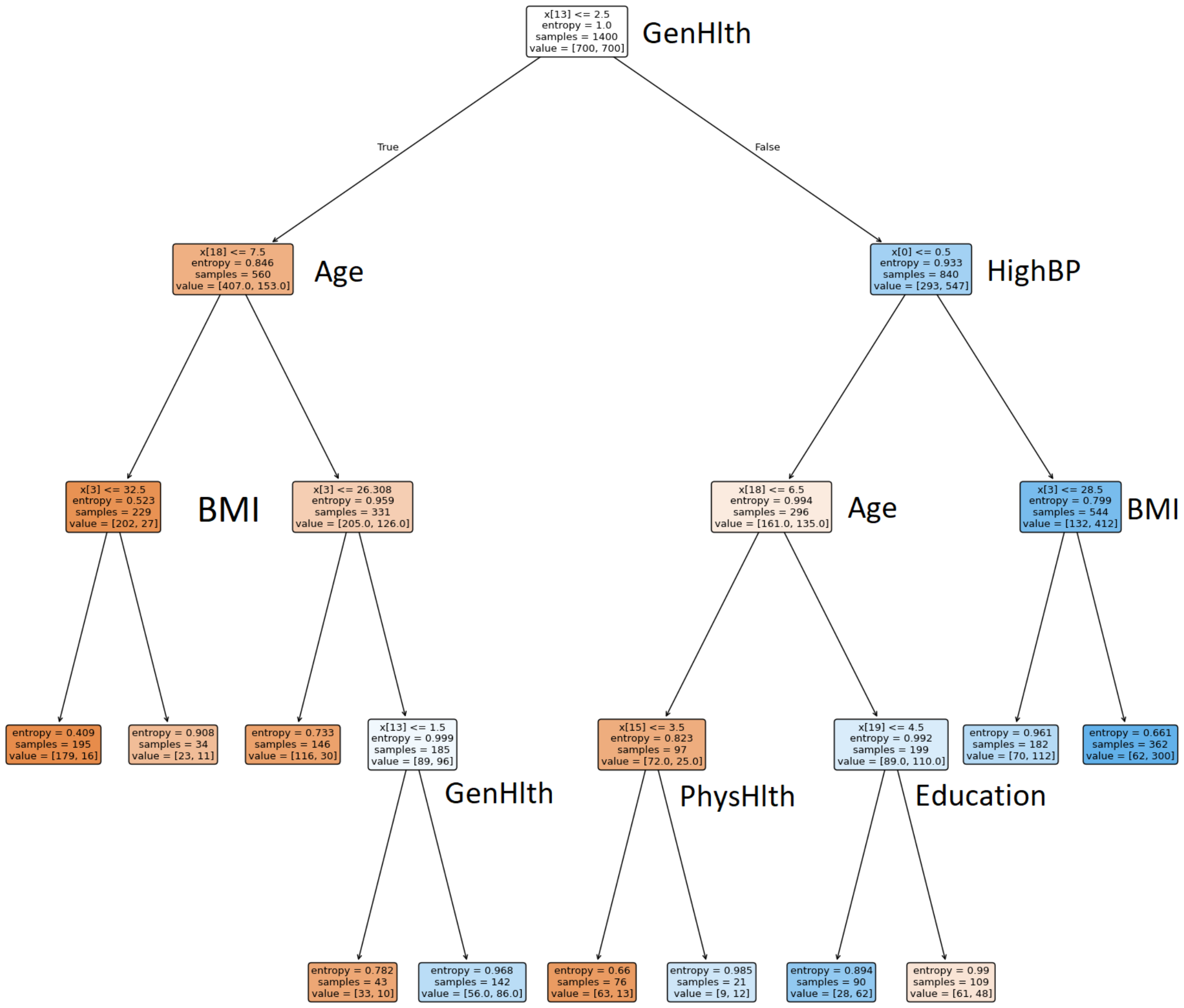
Min_samples_split represents the number of samples node has to have to get considered for further splitting and *min_impurity_decrease* is the pruning threshold by which split must reduce impurity by at least a given amount to be accepted. Those two will be changed during obtaining a tree to find the best performance one.

With only choosing the criterion hyperparameter, the tree obtained had accuracy of 67% with depth of 22 and nearly 700 nodes. It was clear that we need to choose other parameters as well to obtain better performance and interpretable trees.

One way to choose parameters is to manually test combinations and examine the results after each one and in the end choosing the best one. By manual testing with parameters, *min_samples_split=5* and *min_impurity_decrease=0.003*, accuracy increased to 70% and both depth - 9 and number of nodes - 57 decreased.



The obtained tree is still not interpretable and it is “hard on the eye” so further testing is needed. The best and quickest way for choosing parameters is doing **grid search** on the chosen parameters by *GridSearchCV*. The grid search was done from range 2 to 20, only integer values with step=4, for *min_samples_split* and from 0 to 0.5 with 200 evenly spaced values. The tree that performed the best had **accuracy of 72%** on the train set having ***min_samples_split=2*** and ***min_impurity_decrease=0.005025***. Both Naive-Bayes and the obtained decision tree have similar accuracy.



The tree obtained by grid search parameters is much more interpretable with depth of 4 and only 21 nodes as well as better accuracy on the test set which is around 72%. By examining one example we will show some of the rules of the tree and how the tree classifies each example. This example is a true positive one which means its true label is 1 and tree prediction is also 1 which means that this person has diabetes. One of the most important nodes is the **root** of the three which separates starting 1400 samples into two groups of 560 and 840 by considering the **GenHlth** of the sample. If they have less or equal than 2.5 then samples continue their path with the left child and if they have larger value they go to the right child. If the expression is met then the path always goes to the left child and if it is not met it goes to the right one. The path this person follows is:

```
► True label = 1.0,  Predicted = 1.0 -> True Positive
Path through the tree:
   GenHlth <= 2.500 -> sample: 2.0 -> left child
   Age > 7.500 -> sample: 9.0 -> right child
   BMI > 26.308 -> sample: 33 -> right child
   GenHlth > 1.500 -> sample: 2.0 -> right child
```

After the sample reaches the leaf of the tree it is classified in the class that has a larger percentage of representation in the node. That is called the purity of the node and it is better for purity to be closer to 100% as it generally means better classification. The purity of this leaf is as follows:

```
Leaf contains 142 samples: 86 pos / 56 neg
Purity = 60.56%
```

The second examined sample will take the different path from the root having GenHlth score more than 2.5.

```
► True label = 0.0,  Predicted = 1.0 -> False Negative
Path through the tree:pr
   GenHlth > 2.500 -> sample: 4.0 -> right child
   HighBP > 0.500 -> sample: 1.0 -> right child
   BMI > 28.500 -> sample: 33.0 -> right child
Leaf contains 362 samples: 300 pos / 62 neg
Purity = 82.87%
```

The leaf of this sample is one of the purest ones as it has 300 positive samples to 62 negative ones reaching purity of 83%.

The value of BMI appears in the 3 internal nodes which means that BMI gives a lot of information. Two internal nodes that produce the most pure leaves have **BMI** as a criterion for splitting, BMI <= 32.5 and BMI <= 28.5. Those two appear on separate sides of the root. If for the first one criterion is met, the purity of that leaf is 92% to not have diabetes and if the criterion for the second one is not met, the purity of his leaf is 83% for having diabetes.

The + and - examples in the leafs are exactly what determines purity of the leaf and better classification as well. Most errors in the classification come from the leafs that have similar numbers of + and - examples as they are not as pure. Example of that leaf in the tree has purity of 56% in favour of - examples. By reaching that leaf samples have the most percentage of misclassification. It is always desired that the leaves are

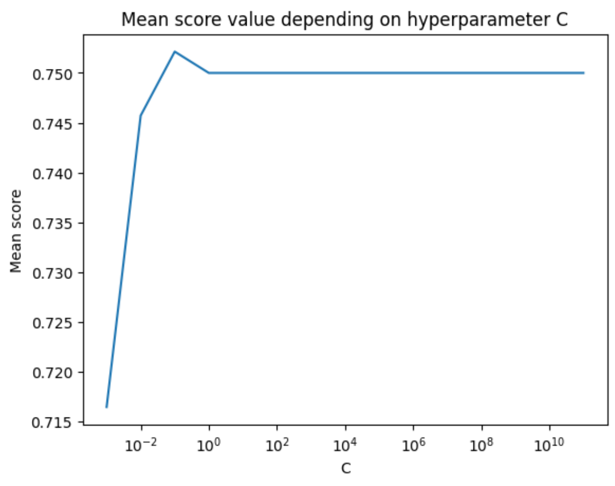
as pure as they can be so that errors happen not as often. Out of 600 samples in the test set 431 of them were classified correctly and out of the remaining 169 there are 89 false positives and 80 false negatives. Most of them probably were wrongly classified in the leaves that are most impure.

Support Vector Machines - Otman

In this method we evaluated 3 types of kernels on our balanced dataset: Linear kernel, polynomial kernel and RBF kernel. The execution time of some of the methods was high as the convergence time with some hyperparameter settings was too high, so for each type of kernel we used a two-step approach to speed the building of the model. First, we did an exploration run with the library's default 10-fold cross-validation, the full log-spaced range of C values and a safety cap of max_iter = 100000 to find the C zone where accuracy peaked. Once that zone was spotted, we used a shorter interval of that accuracy peak and ran the method again to assure the maximum accuracy without the limitation of max_iterations. We also applied other methods to speed up the building of the method such as reducing the rows of the dataset using the preprocessed version and scaling the data.

Linear SVM

In the linear-kernel experiment we skipped the two-step approach because training was already fast. We simply swapped SVC(kernel="linear") for LinearSVC() which reduced the execution time to seconds. For hyper-parameter search we kept the full grid Cs = np.logspace(-3, 11, num=15, base=10.0) and ran 10-fold cross-validation.



```
Acc. 10-fold cross on train data = 0.7492857142857143

Confusion matrix on test set:
 [[215  85]
 [ 64 236]]

Accuracy on test set: 0.7516666666666667

Best value of parameter C found: 0.1

Number of supports: 846 (826 of them have slacks)
Prop. of supports: 0.6042857142857143
```

As we can see, the search picked C = 0.1 and gave a mean CV accuracy of 74.9 %.

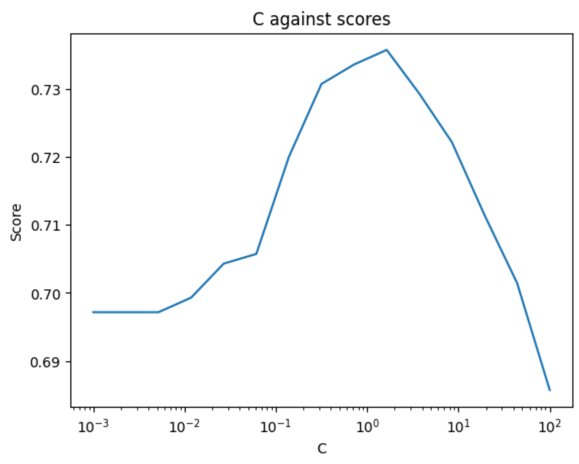
When we applied the same C to the hold-out test set we obtained 75.2 % accuracy, with a precision of around 73 % and an F1-score of 76 % in class 1, which are quite good results. Training the same C with SVC(kernel="linear") showed 846 support vectors, about 60 % of the training data, and most are inside the margin. This tells us a linear boundary with a small C keeps the margin fairly wide, tolerates some mistakes and still fits the bulk of the data well.

Polynomial kernels

For the polynomial experiment, we kept execution time under control by choosing a C grid that balanced speed and accuracy and by running the same two-step approach. First we did a wide-open scan (10-fold CV, max_iter = 100 000) just to see where each kernel had the highest values; then we zoomed in on that C interval, still with 10 folds but without the iteration cap to get the best score.

Quadratic kernel

Using Cs = np.logspace(-3, 2, 15) and cv = 10 the grid-search picked C ≈ 1.64 and reached a mean CV accuracy of 0.736. On the test split the model scored 73% accuracy.

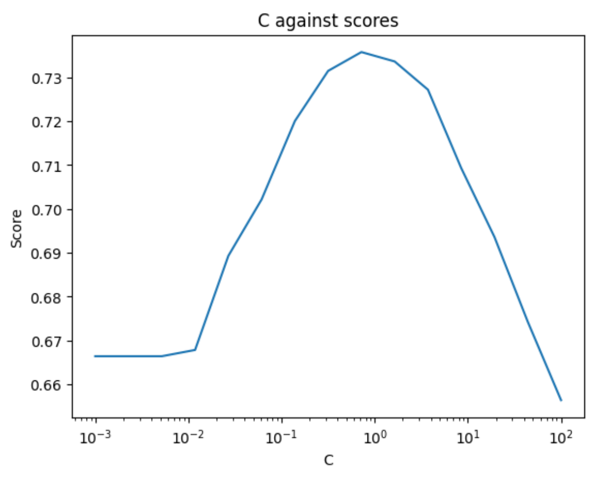


```
Acc. 10-fold cross on train data=  0.7357142857142857

Confusion matrix on test set:
 [[212  88]
 [ 74 226]]

Accuracy on test set: 0.73

Best combination of parameters found:
 {'C': np.float64(1.6378937069540647)}

Number of supports: 851 (754 of them have slacks)
Prop. of supports: 0.6078571428571429
```

Precision for class 1 is about 72% and the F1 74%. The trained model holds 851 support vectors (about 61% of the training data) and 754 of them are slack vectors. In other words, the quadratic surface is just enough to catch more positives but still needs many borderline points to define the margin.

Cubic kernel

Running the same C range with 10-fold CV gave a best C ≈ 0.72 and the same fold-average of 0.736. The hold-out accuracy ticked up slightly to 73.5%.



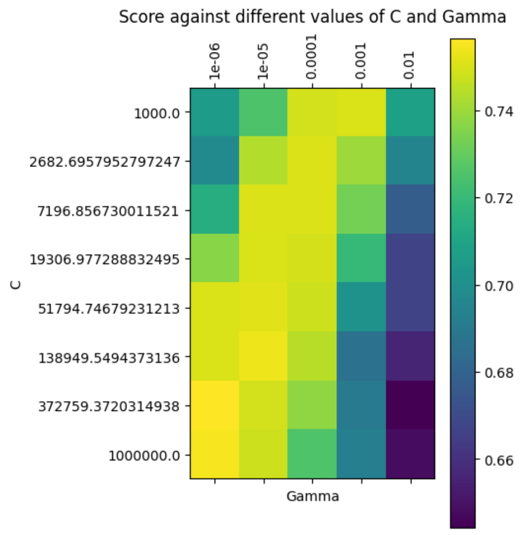
```
Acc. 10-fold cross on train data = 0.7357142857142857

Confusion matrix on test set:
[[211  89]
 [ 70 230]]

Best combination of parameters found:
{'C': np.float64(0.7196856730011522)}

Accuracy on test set: 0.735

Number of supports: 932 (764 of them have slacks)
Prop. of supports: 0.6657142857142857
```

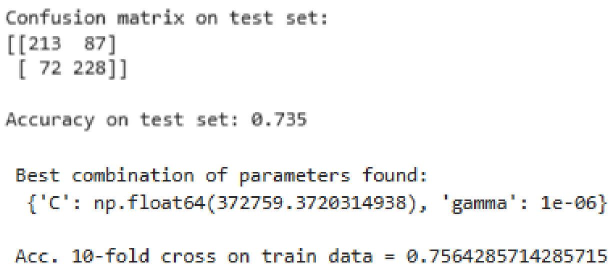
Here the model leans on 932 supports (≈ 67 %), 764 of them inside the margin, showing that the extra flexibility of a cubic term fits a few more true positives but at the cost of a denser, more complex boundary.

Overall, both polynomial kernels behaved similarly with the accuracies in the around the 70 % range and more than half the training points acting as support, so they did not surpass the linear SVM in simplicity.

RBF Kernel

For the RBF, we again used the two-step method, but this time we had to modify two parameters, the C and the gamma. In the first part, we let the model test a broad grid (C from 0.1 to 10^6, gamma from 10^-6 up to 0.01, 10-fold CV, max_iter = 100 000) so the heat-map could show where the scores were highest. In the second part, we kept only that bright zone and removed the iteration limit. The grid-search showed the highest score on C ≈ 3.7×10^5 and gamma = 1×10^{-6}, giving a mean CV accuracy of 0.756. When we trained again with those values we got 75.7 % accuracy.





```
Confusion matrix on test set:
[[213  87]
 [ 72 228]]

Accuracy on test set: 0.735

 Best combination of parameters found:
  {'C': np.float64(372759.3720314938), 'gamma': 1e-06}

 Acc. 10-fold cross on train data = 0.7564285714285715
```

The results show for the positive class a 74% precision, 78% recall, and a 76% F1-score. This model needs 809 support vectors, which represents 58% of the training data with most of them inside the margin.

We also could observe that a very small gamma keeps the RBF surface smooth, while the large C reduces the margin tightly around the relevant points. The combination of these parameters give good accuracy with fewer support vectors than the polynomial runs.

Meta-learning algorithms - Joan

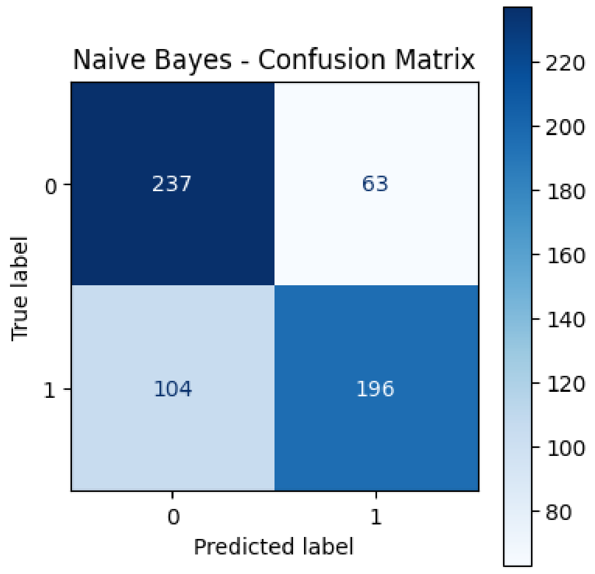
In this section, we will evaluate four different ensemble methods. For each algorithm, we will explain which key settings we chose and why.

Performance Majority Voting

Instead of relying on a single algorithm's prediction, we train Naive Bayes, tuned K-NN, and a Decision Tree. Each model votes for its predicted class, and we pick the class with the most votes. This is called "Hard Voting."
To evaluate this ensemble's performance, we first apply 50-fold cross-validation on the training data to estimate its accuracy, and then we generate a confusion matrix using the testing data.

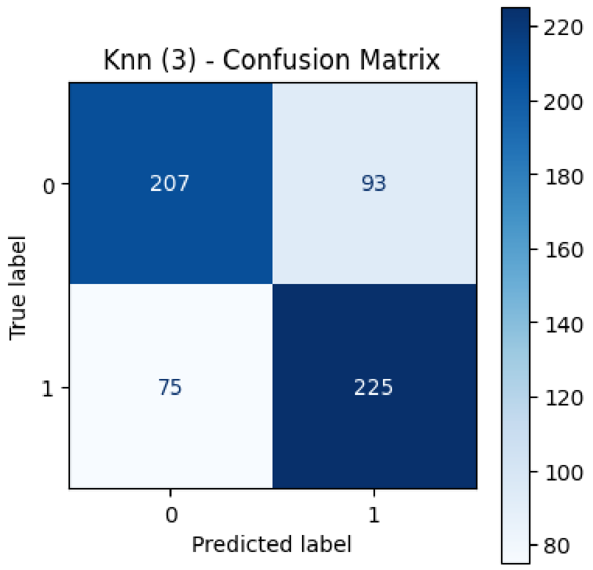
Naive Bayes

```
Train CV Accuracy: 0.715 [Naive Bayes]
```



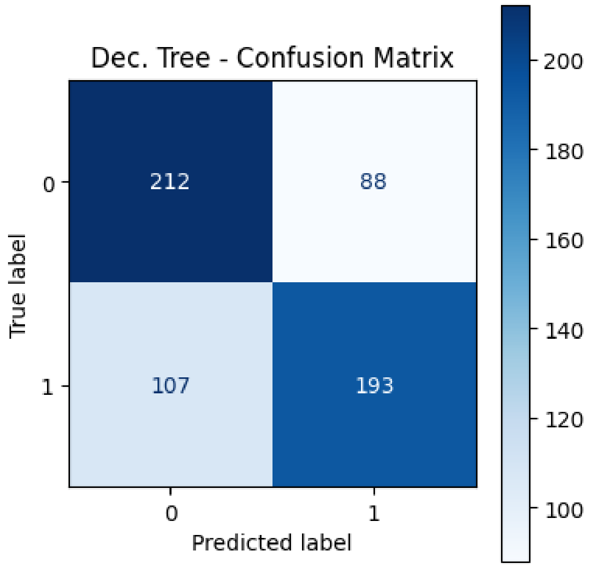
Optimized K-NN

```
Best Params for Knn: {'n_neighbors': 21, 'weights': 'uniform'} -
Accuracy: 0.743
Train CV Accuracy: 0.731 [Knn (3)]
```



Decision Tree

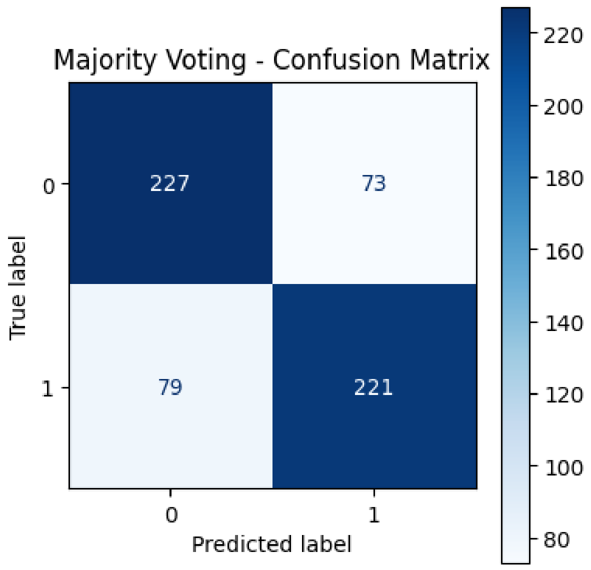
```
Train CV Accuracy: 0.660 [Dec. Tree]
```



Majority Voting

Among the three individual models, the optimized K-NN achieved the highest accuracy. Since the accuracies were similar, we decided not to perform a weighted majority voting classifier. Instead, we combined all three into a single unweighted majority voting classifier:

```
Train CV Accuracy: 0.729 [Majority Voting]
```



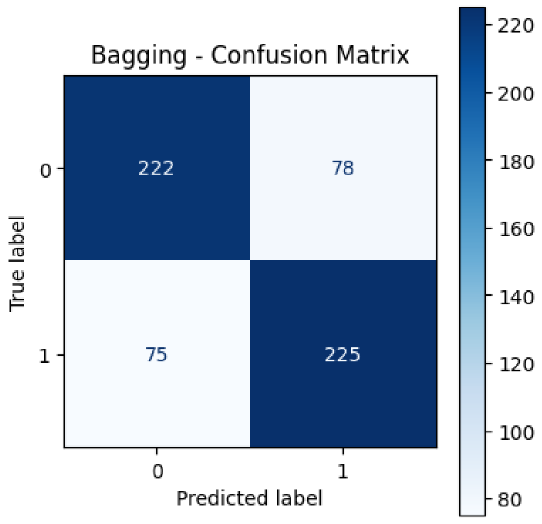
We obtained an accuracy of 0.729, which is slightly lower than the Optimized K-NN accuracy we had before. This could be attributed to the influence of the other two classifiers in our unweighted hard-voting ensemble. When Naive Bayes or the Decision Tree misclassify a sample that K-NN would get right, their votes override K-NN's one and we lose accuracy.

Bagging

For this method, we trained the classifier using a Decision Tree as the base estimator on the training data. We set the number of max_features to 0.35. We calculated the accuracy for different values of estimators using 10-fold cross validation.

```
Accuracy: 0.629 [n° estimators: 1]
Accuracy: 0.627 [n° estimators: 2]
Accuracy: 0.669 [n° estimators: 5]
Accuracy: 0.706 [n° estimators: 10]
Accuracy: 0.722 [n° estimators: 20]
Accuracy: 0.721 [n° estimators: 50]
Accuracy: 0.720 [n° estimators: 100]
Accuracy: 0.732 [n° estimators: 200]
```

We achieved an accuracy of 0.732 using 200 estimators, so we generated a confusion matrix

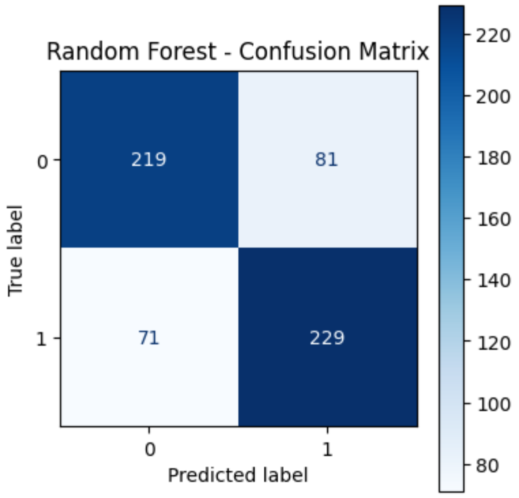


to see how the classification performed on the testing data.

Random Forest

For the Random Forest method, we trained the classifier on the training data using different numbers of trees. Random Forest is an ensemble technique that combines multiple decision trees to improve accuracy and robustness over a single tree.

```
Accuracy: 0.644 [n° estimators (trees): 1]
Accuracy: 0.660 [n° estimators (trees): 2]
Accuracy: 0.693 [n° estimators (trees): 5]
Accuracy: 0.694 [n° estimators (trees): 10]
Accuracy: 0.710 [n° estimators (trees): 20]
Accuracy: 0.726 [n° estimators (trees): 50]
Accuracy: 0.728 [n° estimators (trees): 100]
Accuracy: 0.741 [n° estimators (trees): 200]
```

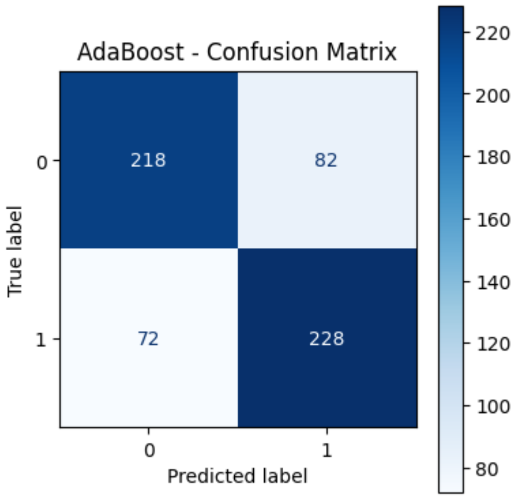


We obtained an accuracy of 0.741 with 200 trees, the best
result among all the techniques tested so far. The confusion matrix was generated with 200 estimators.

Boosting (Adaboost)

For this ensemble method, we trained the classifier on the training data using various numbers of estimators.

```
Accuracy: 0.668 [n° estimators: 1]
Accuracy: 0.668 [n° estimators: 2]
Accuracy: 0.713 [n° estimators: 5]
Accuracy: 0.722 [n° estimators: 10]
Accuracy: 0.731 [n° estimators: 20]
Accuracy: 0.734 [n° estimators: 50]
Accuracy: 0.733 [n° estimators: 100]
Accuracy: 0.736 [n° estimators: 200]
```



We can see how beyond 20 estimators the accuracy doesn’t change much, so for large datasets it would be great to consider using fewer estimators to reduce computation time.
The highest accuracy was achieved with 200 estimators, and the confusion matrix illustrates how the classification performed on the testing data.

Conclusion

After analyzing all of the meta-learning algorithms, we found that they all achieve a similar accuracy, which is around 73%.
That said, if we had to choose one, we would likely go with the Random Forest ensemble method, since it slightly outperforms the others (74.1%).
Because the computation times for all four techniques was minimal, we did not treat runtime as a deciding factor.

Comparison and conclusions - Álvaro

Method	**CV Accuracy**	**95% CI**
Naive Bayes (threshold tuned)	73%	68% - 78%
K-NN (k=16, top 5 features)	75%	70.3% - 80.1%
Decision trees (depth 4, pruned)	72%	66.9% - 77.1%
Linear SVM (C = 0.1)	74.9%	70.3% - 80.1%
Polynomial kernels	73.6%	68.3% - 78.9%
RBF SVM ($C \approx 3.7\times10^{5}$, $\gamma = 1\times10^{-6}$)	75.7%	70.8% - 80.6%
Majority Voting	72.9%	67.9% - 77.9%
Bagging (200 trees)	73.2%	68.2% - 78.2%
Random forest (200 trees)	74.1%	69.1% - 79.1%
AdaBoost (200 estimators)	73.6%	68.6% - 78.6%

The RBF-kernel support vector machine obtained the highest accuracy of 75.7%, with its confidence interval edging over KNN and linear support vector machine. Naibe Bayes seems to suffer from its conditional independence assumption, as many health factors are biologicaly correlated, so there is no real independence. KNN is typically sensitive to irrelevant data or choosing the wrong k, but when selecting the top 5 most informative attributes and iterating over k to find the most appropriate one, it becomes the second better performing method, with an SV accuracy of just 0.7% less than the RBF SVM. Decision trees initially faced overfitting issues, but by pruning, we achieved a simpler tree of depth 4 with a decent accuracy of 72% and good interpretability, yet it was still the worst-performing method in terms of accuracy. Simple bagging and AdaBoost hovered around 73.2% and 73.6% respectively, similar to individual trees and Naive Bayes, staying in the middle of the pack. Random forests gave an accuracy of 74.1%, slightly below other more accurate methods but with the added bonus of feature-importance insights and robustness to hyperparameters. Finally, Majority voting of Naive Bayes, KNN, and a single tree actually under-performed the best base learner (KNN) since all three models tended to miss on similar samples.

While the RBF SVM gave the highest accuracy, the margin above a linear SVM and KNN with the right optimizations (k=16 and top 5 features) is small, of 0.8% and 0.7% respectively. Since SVM's can be sensitive to parameter tuning and less transparent, and KNN can have a higher prediction cost, the need for tuning the parameters and does not give a global picture of which features matter, I would choose Random forests as the most balanced and useful method. It gives a relatively similar performance to our top performers (74.1%) with built-in measures of feature importance, a better interpretability and less parameter sensitivity. Therefore, for a real-world application, I would happily sacrifice some percent in accuracy for a better insight to my dataset.