Fetal Acidosis Classification

Dominik Fischer

Abstract—**In this text, four machine learning models, namely the Logistic Regression, Support Vector Machine, Random Forest and Mulit-Layer Perceptron classificator, were configured and tuned. Said classifiers were designed for detection of fetal acidosis, which is a medical term referring to high levels of toxicity in an unborn baby's blood. The classifiers were veryfied on real data and multiple performance measures, such as ROC curves, AUC, geometric mean etc. were used to determine, which of the classifiers exhibits best behaviour and which will therefore be used as a representant to showcase its performance on external data. All of the models showed to be comparably accurate and each one prevailed in a different tested aspect. The best model in terms of achieved average geometric mean score was found to be Support Vector Machine with the average geometric mean of 0.699 from 50 testing runs.**

I. Assignment

The goal of the semestral work was to create a machine learning model for fetal heart rate classification that would be able to predict whether fetuses suffer from acidosis. The classification procedure is as follows. Based on the provided features, the model ought to predict whether the baby's blood will have pH > 7.05 and thus be classified as an *acidosis case* or vice-versa.

The performance of proposed classifiers will be mainly measured using geometric mean (*gmean*), which is defined as

$$gmean = \sqrt{SE \cdot SP}, \tag{1}$$

where $SE = TP/(TP + FN)$ denotes *sensitivity* and $SP = TN/(TN + FP)$ is *specificity*. TP and FN represent the number of *true positives* and *false negatives* respectively.

The requirements on peformance of the final chosen and submitted model were that it should achieve *gmean* ≥ 0.67 on external data that are not known in advance. [1]

II. Introduction

Some form of machine learning can nowadays be found in the vast majority of different fields. Its application ranges from steering autonomous cars to forecasting weather. In recent years the ability of machine learning algorithms to process large amounts of data and to make precise decisions based on said data has allowed diferent kinds of not only classification algorithms to find their use in medical diagnosis.

Acidosis is a medical condition where the subject suffers from high toxicity of blood (pH $>$ 7.05). This can also affect unborn children, i.e. *fetuses*. Desipte monitoring of the fetal heart rate to assess the fetal health status during delivery, it remains a challenging task to reliably detect fetal acidosis, which would in turn allow for relevant decisions for operative delivery to be made. [1]

In this text the framework of machine learning was be applied to the above described problem and the capability of different classification models to precisely diagnose fetal acidosis was be measured and compared.

The text is organized as follows. In Section III, the available dataset is described together with preprocessing procedures which were performed. Then the designed classifiers are described and their properties are discussed. In Section V the performance of the classificators is tested and the results are presented. Here, also the most accurate model is selected. Section VI offers the discussion of achieved results and finally section VII concludes this text and recapitulates the main results.

III. Data

The dataset available for training and evaluation of designed models consists in total of 4546 intrapartum recordings, which were acquired between April 2010 and September 2017 at the obstetrics ward of the University Hospital in Brno, Czech Republic. [1] Together with the pre-delivery recordings, the information about the pH level of the born child is present to allow for assessment of the accuracy of predictions.

Furthermore the whole monitored labor process is divided into a number of segments, each representing 20 minutes of recordings during different labor stages.

A. Used Features, Segments and Labor Stages

Here, the decision process behind the choice of which part of the dataset i.e. what labor stage and how many recoding segments, would serve best as the information source for the models, is described. There are two parameters that allow for selection of distinct subset of the provided training dataset, namely `select_stage` which allows for labor stage selection and `nr_seg` for controlling the number of data segments used.

1) Labor Stage Selection: When selecting the labor stage, there are three possible values of `select_stage` to choose from, namely 0 - all stages, 1 - first stage and 2 - second stage. According to the performed experiments, data from the first stage do not lead to any convincing results when compared to the other two choices. It was established that if the model uses data only from the second stage of labor, the results were comparable, if not better as when the data from all stages were used. Besides that, the amount of data needed for comparable correctness of prediction is much lower when using only the second stage. Thus for further tuning of the model, the variable `select_stage` was set to `select_stage = 2` and only data from the second stage were used. Claims of this subsection as well as the of III-A.2 are based on data in Table I.

Segments \ Stage	All	1st	2nd
2	0.65	0.55	0.69
4	0.67	0.62	0.70
6	0.64	0.59	0.72
8	0.62	0.52	0.76
10	0.59	0.60	0.67
12	0.62	0.60	0.58
14	0.60	0.59	0.62
16	0.59	0.59	0.62

TABLE I: Comparison of MLP *gmean* values on testing data split based on labor stages and number of segments.

2) Number of Segments Selection: In order for the data to represent a longer period of the selected stage, the number of used segments was inceased to 8. This is done by setting the `nr_seg` variable to `nr_seg = 8`. This value was found to be a good tradeoff between computation time and performance of the model. Further increasing the number of segments used did not lead to any substantial improvements of the model.

B. Data Preprocessing

The loaded features `X` and labels `y` could not be used for model tuning directly but needed to be slightly preprocessed.

First, records with some missing values were discarded.

This was followed by standartization of the features. This means that they were centered around origin and scaled so that they have an unit variance. These operations were performer using the `sklearn`'s `StandardScaler()`.

Then it was needed to deal with the imbalanced nature of the dataset. The problem with imbalanced dataset arises from the fact that one class is heavily more represented that the other class, in this case the majority class represents healthy babies i.e. those with normal pH. Fitting the model directly to such a biased data could in turn lead to biased classificator that could apriori lean towards classifying examples as negative class. One possible solution to this problem is to *over–sample* the training dataset. This means that new samples of the under-represented class are artificially generated and added to the training dataset. Over–sampling was performed using the `python` library `imblearn`[1].

The algorithm used for over–sampling was the Synthetic Minority Over–sampling Technique. [2] This algorithm generates new samples based on k nearest neighbours as

$$x_{new} = x_i + \lambda \left(x_{zi} - x_i \right), \tag{2}$$

where x_i is the considered sample, x_{zi} is an arbitrary sample from k nearest neighbours of x_i and $\lambda \in [0, 1]$ is a random number. The over–sampling was performed with the help of the `imblearn.SMOTE()` function.

IV. Design of Classificators

This section describes the proposed classifiers, namely the Logistic Regression, Support Vector Machine, Random Forrest and Multi–layer Perceptron classifiers. Each of the aforementioned classifiers is first briefly described in general which is followed by the description of each model's parameters.

[1]`https://imbalanced-learn.org/stable/index.html`

A. Logistic Regression Model

Logistic Regression Model belongs to classifiers which use a discriminant function to decide how an observation should be classified. This discriminant function can be an arbitrary function $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$, corresponding to a single class. The discriminant functions are then used to create a decision rule, which assigns a class to an observation $\hat{y} = \arg\max_k f_k(\mathbf{x})$, where $\hat{y}$ is the predicted class.

The Logistic Regression Classifier uses a nonlinear transformation of values of a linear function as its discriminant function, namely

$$f_{\mathbf{w}}(\mathbf{x}) = \phi(\mathbf{x}\mathbf{w}^\top) = \frac{1}{1 + \mathrm{e}^{-\mathbf{x}\mathbf{w}^\top}}, \tag{3}$$

where

$$\phi(z) = \frac{1}{1 + \mathrm{e}^{-z}} \tag{4}$$

is the logistic function, see Fig. 1. Owing to the shape of the logistic function, the decisions of logistic regression are much less influenced by examples which are far from the boundary.

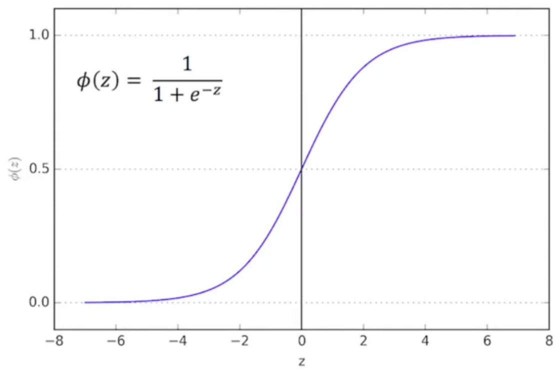

Fig. 1: Logistic function.

1) Tuned Logistic Regression Model: The `sklearn`'s implementation `LogisticRegressionCV()` offers numerous adjustable parameters which can be used for tuning the final model. Some of the hyperparameters were altered and some were left at their default values.

The strength of regularization can be controled by setting the `Cs` parameter. This takes in a list of floats and the model uses it internally to select the optimal value using a grid–search–like technique. The values used as candidates were from the range of 10^{-4} to 10^3.

Coupled with this is the hyperparameter `scoring` which allows for non–default scoring function to be used when internally evaluating the model's performance. Because the main performance indicator is the *gmean* score a custom scorer `our_scorer`, see the following code snippet, has been designed and passed as value of the `scoring` parameter.

Listing 1: Custom scoring function.

```
our_scorer = make_scorer(
                utils.g_mean_score,
                greater_is_better=True)
```

Further, the maximum number of iterations was restrained to `max_iter = 500` and the internal solver was chosen `solver ='newton-cg'`.

The complete model is defined in the following listing

Listing 2: Logistic Regression Model

```
log_clf = LogisticRegressionCV(
        Cs = C_vals, max_iter=500,
        scoring = our_scorer,
        solver = 'newton-cg')
```

The final model was integrated into a pipeline which first scales the input data accordingly, then up–samples them and finally, the Logistic Regression Model is applied. The final pipeline is in the listing below.

Listing 3: Final assembled pipeline.

```
clf = make_pipeline(StandardScaler(),
        SMOTE(), log_clf)
```

B. Support Vector Machine Model

In number of classification methods, the goal is to separate the given classes with some hyperplanes. An optimal separating hyperplane then maximizes the distance to the nearest training sample of any class. In general, parameters of any hyperplane can be written as a linear combination of *support vectors*. Support vectors are points that are closest to the decision boundary. In order to facilitate separation of the data points into classes, SVM uses *basis expansion* i.e. mapping to a higher dimensional space. Furthermore they introduce special *kernel functions* $k(\mathbf{x}_1, \mathbf{x}_2)$ which define dot product in the higher dimensional spaces. The separating hyperplanes are then selected as a set of points whose dot product (or rather kernel function applied to them) with a vector in the given space is constant. The discriminant function of SVM can then be written as

$$f(\mathbf{x}) = \operatorname{sign}\left(\sum_{i=1}^{|T|} \alpha_i y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x}) + \omega_0\right), \tag{5}$$

where $\alpha, \omega_0 \in \mathbb{R}$ are constants.

1) Tuned Support Vector Machine Model: Due to the principle behind SVM, one of the most influential parameters on the performance of the model is its kernel function. The `sklearn`'s `SVC()` offers the choice between 'linear', 'poly' standing for polynomial, 'rbf' standing for Radial Basis Function, 'sigmoid' or 'precomputed' kernel. The chosen kernel for this model was 'rbf' (`kernel = 'rbf'`), which is defined as

$$k(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(-\gamma|\mathbf{x}_1 - \mathbf{x}_2|^2\right). \tag{6}$$

From equation (6) proceeds that another important tunable parameter is γ. The learning curve of the SVM model for this parameter is shown in Fig. 2. From the learning curve it can be assessed that the optimal value of the hyperparameter γ lies somewhere between 10^{-3} and 10^{-1}. Therefore the final value of γ is found using the `GridSearchCV()` using values from this range when training the model.

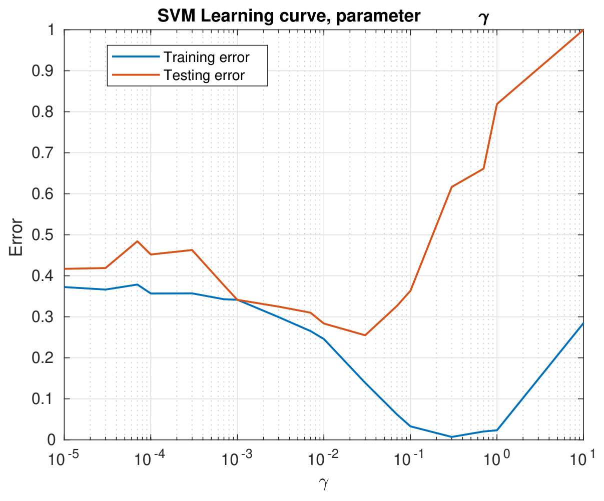

Fig. 2: Learning curve for parameter γ.

Last parameter that has been changed from its default value is the regularization parameter `C`. Again, for a rough estimate of said parameter its depending learning curve was computed, see Fig. 3. Seemingly, the best value lies somewhere around 0.2. As with the previous parameter, `GridSearchCV()` is performed on values `C`$\in [0.1, 0.2, 0.3]$ to find the most suitable parameter for given data while training the model.

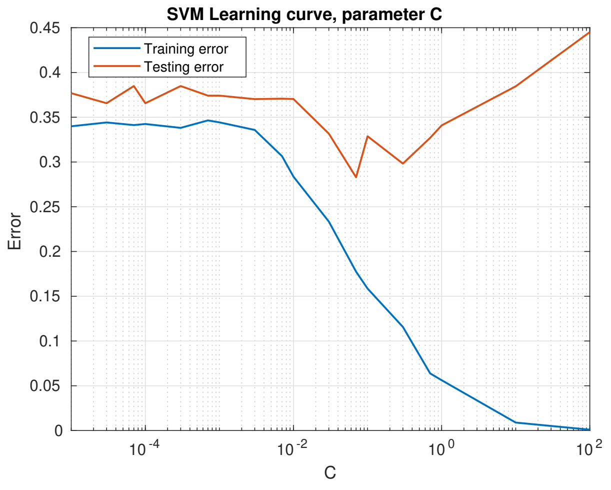

Fig. 3: Learning curve for parameter `C`.

The definition of the final model is in the following listing.

Listing 4: SVM model.

```
svm_clf = SVC(kernel = 'rbf')
clf = make_pipeline(StandardScaler(),
                  SMOTE(), svm_clf)
params_selection = {
        'svc__gamma' : [1e-3,1e-2,1e-1],
        'svc__C' : [0.1, 0.2, 0.3] }
clf = GridSearchCV(clf,
        param_grid = params_selection,
        scoring = our_scorer)
```

C. Random Forrest Model

Ensamble methods create multiple models and then combine their results to provide an improved estimate or prediction. Random Forrest Classifier uses a set of Decision Trees as the internal model and in general it performs averaging of these deeper Decision Trees which are each trained on a different subset of the training dataset also known as the Bootstrap Aggregation or *Bagging*.

1) Tuned Random Forest Model: An important hyperparameter that needs to be paid attention to is the number of internally used Decision Trees. This is adjustable via the `n_estimators` parameter. In Fig 4 is the learning curve corresponding to this parameter. As can be seen from the both subplots, increasing the number of estimators does not bring any significant improvement in performance of the model. On the contrary, it only results in overfitting on the training dataset and increasing time needed for training. Therefore it is not worth tuning this parameter using `GridSearchCV()` and for the sake of computation time, only a static value of `n_estimators = 40` was set.

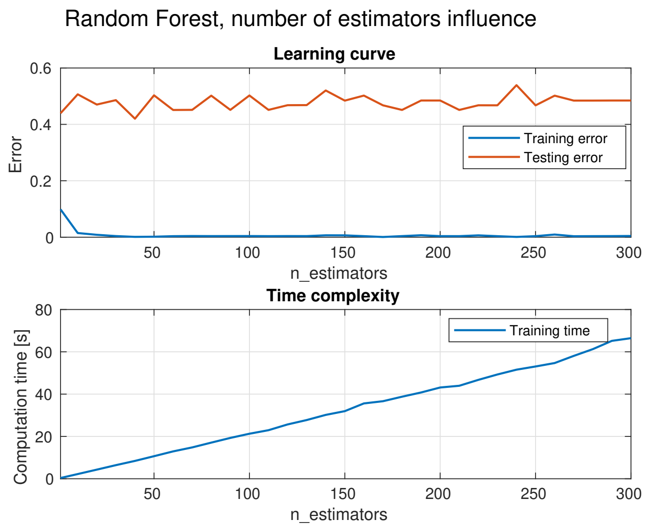

Fig. 4: Influence of number of estimators on the performance of Random Forest model.

Another parameter, that influences the performance of underlying Decision Trees structure is `max_depth`. This sets the maximal depth of one internal estimator. The procedure of determinig its value is very similar to the described in the previous paragraph. According to the learning curve in Fig.5, the optimal value of this parameter lies somewhere around 5. Therefore `GridSearchCV()` is again used for testing of the interval $[3, 7]$ when training the model.

The last tuned hyperparameter was `criterion`, which sets the measure used when establishing the quality of performed split. Thanks to the finite set of possible values, the parameter was selected manually and set as `criterion = 'entropy'`.

The following listing shows the complete Random Forest classifier definition.

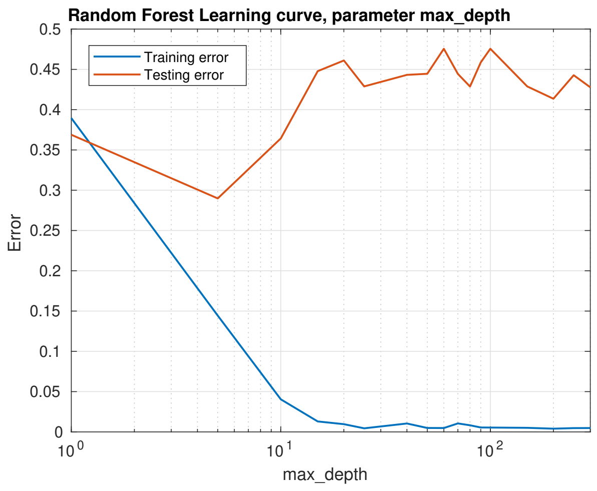

Fig. 5: Learning curve of the Random Forest model for `max_depth` parameter.

Listing 5: Random Forest model.

```
rf_clf = RandomForestClassifier(
        criterion = 'entropy',
        n_estimators = 40)
clf = make_pipeline(StandardScaler(),
                  SMOTE(), rf_clf)
params_selection = {
'randomforestclassifier__max_depth' :
                  [3,4,5,6,7]}
clf = GridSearchCV(clf,
        param_grid = params_selection,
        scoring = our_scorer)
```

D. Multi–layer Perceptron Model

Multi–layer Perceptron (MLP) Model falls into the category of feedforward networks, meaning that the signal is propagated from inputs to outpus only. MLP consists of multiple layers of neurons and each neuron j transforms the features $\mathbf{z}_j$ using the inner product with its weights vector $\mathbf{w}_j$

$$a_j = \mathbf{z}_j \mathbf{w}_j^\top, \tag{7}$$

where a_j is the output of the neuron, see Fig. 6. The outputs z_j of a neuron are computed using the activation function $g()$ as $z_j = g(a_j)$.

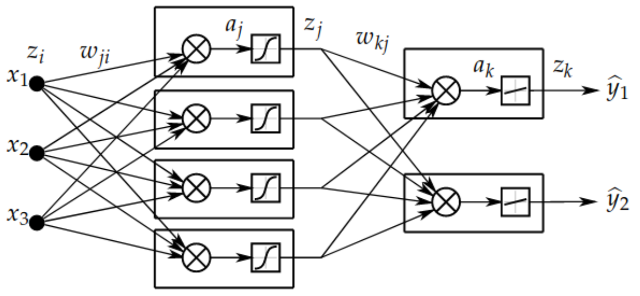

Fig. 6: Structure of a MLP classifier. [3]

1) Tuned Multi–layer Perceptron Model: There are number of parameters that needed to be changed in order for the classifier to behave accordingly. A very important hyperparameter which highly influences the performance of any given MLP is its activation function. The `'relu'` function was observed to function reasonably well for this classifier. It is also the default value of the `activation` parameter, so nothing had to be done there.

There are a couple more parameters which were selected manually, namely the `solver = 'adam'` and `learning_rate = 'adaptive'`. *Adam* is a stochastic–gradient based optimizer for the internal weights optimization, which performs well on larger datasets and `learning_rate` is used for selecting the size of step in the gradient descent optimization.

When assessing the regularization strength (L2 penalty) parameter α, the learning curve in Fig 7 was used again. The parameter has no susbtantial influence on the performance of the model, however value around 10^{-4} still seems like the most optimal parameter and thus α was set to this value.

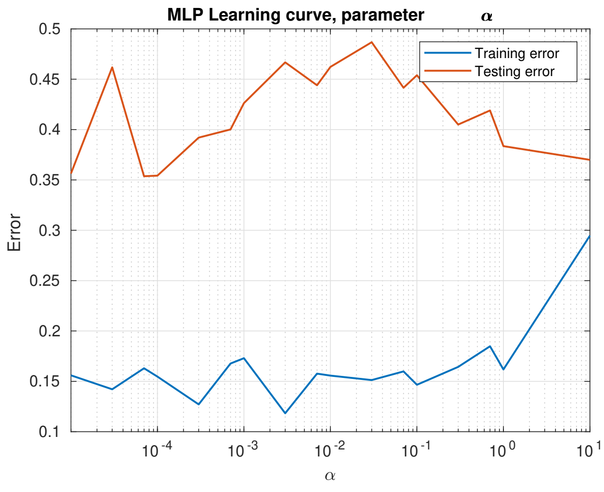

Fig. 7: MLP learning curve of parameter α.

The structure of the classifier i.e. the number and size of hidden layers is given by parameter `hidden_layer_sizes`. The best number of layers was found to be 2, since a high number of layers led to over–fitting rather than improving the results on testing data. Since there are no many efficient ways to really look for the optimal layer sizes other than some exhaustive grid search, the approach was as follows. A couple of tuples of sizes were generated and the one which yielded best results was chosen. This was done with the knowledge that the resulting sizes might not be optimal, but it is enough that they are sufficiently *good*. For the overview of tested values and resulting *gmean* values, the reader is reffered to Table II. The *gmean* value presented in said table was computed as an average of 3 runs.

According to the data in Table II, the hyperparameter was selected as `hidden_layer_sizes = (3,32)`.

The final designed MLP model definition can be found in the following listing.

Layer size	$(3,32)$	$(10,50)$	$(30,5)$	$(1,5)$	$(10,25)$
gmean	0.661	0.502	0.531	0.641	0.5571

TABLE II: Comparison of individual layer sizes and their performances.

Model	Logistic Regression	SVM	Random Forest	MLP
TPR	0.574	0.574	0.759	0.630
FPR	0.203	0.117	0.401	0.265
TNR	0.796	0.883	0.599	0.735
FNR	0.574	0.426	0.241	0.370

TABLE III: TPR and FPR values for individual models.

Listing 6: MLP model.

```
mlp_clf = MLPClassifier(
        solver = 'adam', max_iter = 5000,
        hidden_layer_sizes = (3,32),
        alpha = 0.0001)
clf = make_pipeline(StandardScaler(),
                    SMOTE(), mlp_clf)
```

V. Validation of Classificators

In this section the performance of the configured classifiers is examined and compared. The accuracy of predictions was measured in the way that all of the classifiers were trained and tested on the same subsets of data. Numerous metrics were used to assess the performance of models e.g. the ROC curve, AUC, *gmean* etc.

A. Confusion Matrices, TPR, FPR, TNR, FNR

In this subsection, the confusion matrices of individual classifiers are presented and described. The confusion matrix evaluates the accuracy of the classifier by visualising how many of the total samples were classified correctly and incorrectly. In this experiment, the testing dataset had 1923 healthy cases and 54 cases with confirmed acidosis. The confusion matrices are in Figs. 8-11.

When comparing the number of correctly classified datapoints, the most succesful model showed to be SVM with 1698 true positives (TP) and 31 true negatives (TN). On the other hand, should only TN be considered, the Random Forest classifier would be the best with 41 TN predictions. Furthermore, the positivity and negativity ratios (TPR, FPR, TNR, FNR) are summarized in Table III. The ratios are given by

$$TPR = \frac{TP}{TP+FN}, \qquad FPR = \frac{FP}{FP+TN},$$
$$TNR = \frac{TN}{TN+FP}, \qquad FNR = \frac{FN}{FN+TP}.$$

B. ROC curves and AUC

The receiver operating characteristic, see Fig. 12 is created by plotting TPR against FPR at various decision thresholds. This in turn shows the diagnostic ability of the classifier. The better the classifier, the closer to the top left hand corner (0

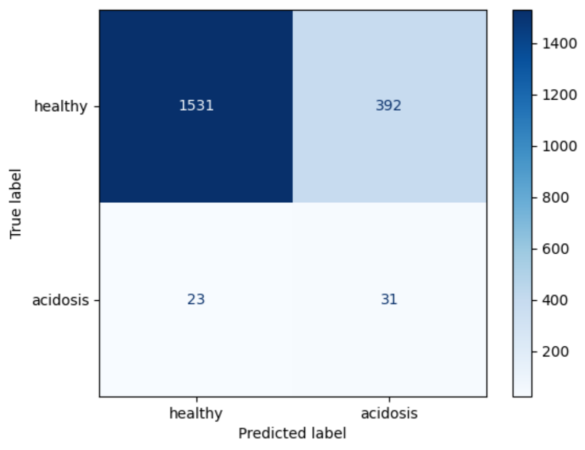

Fig. 8: Confusion matrix for Logistic Regression Model.

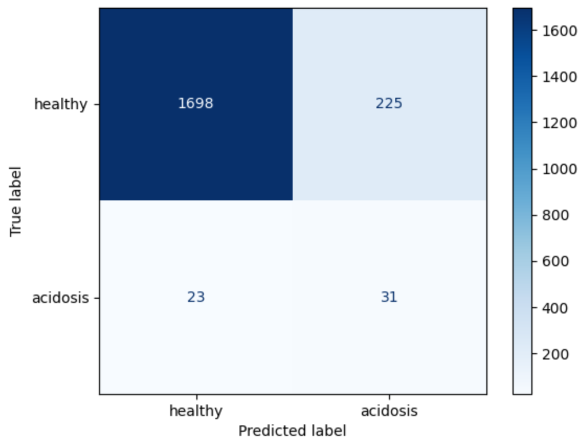

Fig. 9: Confusion matrix for SVM Model.

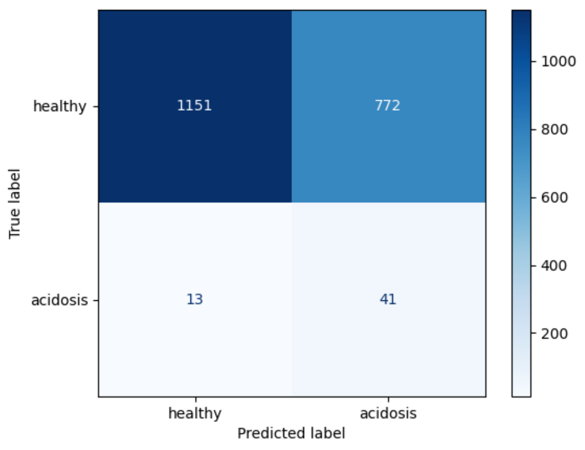

Fig. 10: Confusion matrix for Random Forest Model.

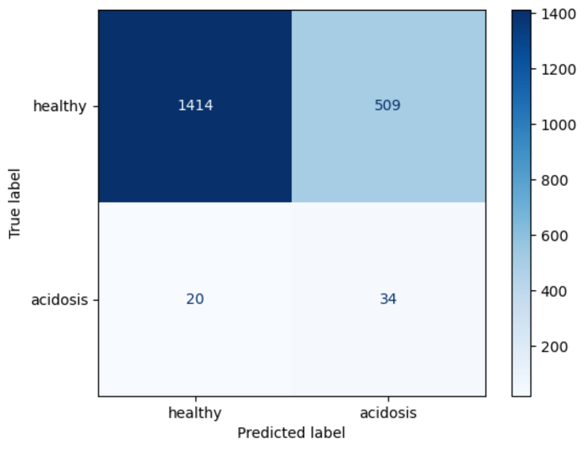

Fig. 11: Confusion matrix for MLP Model.

FPR, 1 TPR) its ROC curve. Plotted ROC curves show again, that the performance of models is quite similar with all of them crossing each other and no one being superior overall. Despite that one can still conclude that Logistic Regression performs the best (has highest TPR) for lower values of the threshold an MLP for higher values.

Another metric how the quality of a model can be measured using the ROC curve is to compute the area under the curve (AUC). This gives another simple scalar *greater–is–better* comparison method. The computed AUC values can also be found in the legend of Fig. 12. According to the AUC values MLP performs the best with AUC = 0.77, followed by SVM and Logistic Regression with AUC=0.75, finally Random Forest Model achieved AUC = 0.73.

C. Choosing the Best Classifier

From the designed classifiers described in section IV, it was necessary to choose one that performs the best overall for it to be used as the representant in the quality scoring evaluation upon submission of this semestral work. The experiment based upon which the choice between the classifiers was made was rather a simple one. 50 times was the available dataset split into a training and testing subsets and these were used for training and validation of the models. The one with highest average $gmean$ score on the testing data was picked. For the comparison of results, see Fig. 13. Horizontal lines of corresponding colour denote the average achieved $gmean$.

Numerical values of achieved $gmean$ are in Table IV.

Based on the measured performances, the chosen representant was SVM model from Subsection IV-B.1.

VI. Discussion

In this section the main results of Section V, together with the overall performance of the classifiers are summarized and

Model	Logistic Regression	SVM	Random Forest	MLP
$gmean$	0.661	0.699	0.634	0.659

TABLE IV: Average $gmean$ values.

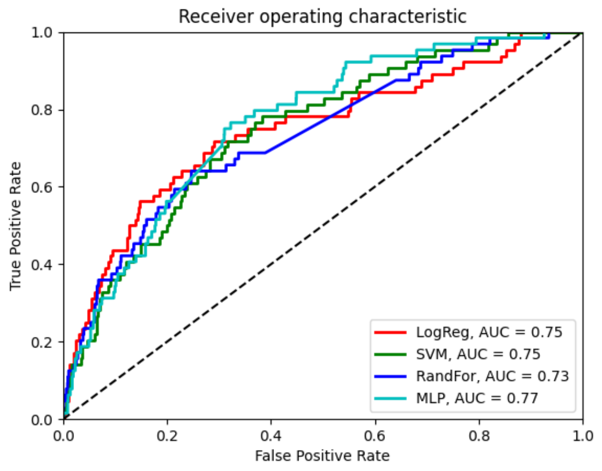

Fig. 12: ROC curves for individual models.

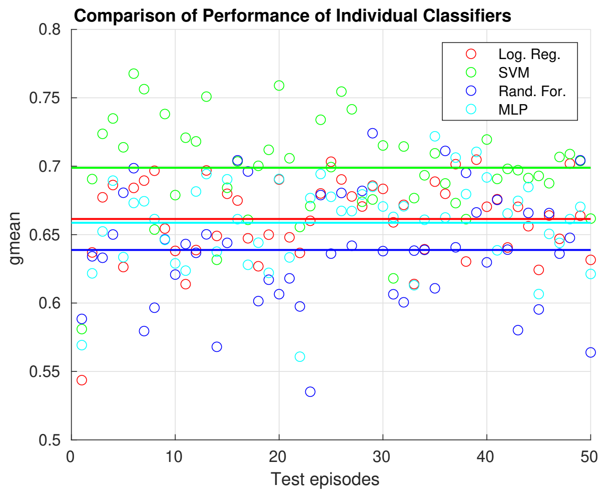

Fig. 13: Comparison of achieved $gmean$ scores.

discussed. Some key observations from Section V are restated and more elaborated.

After consulting the results of performed experiments, it is difficult to select one model which showcased the best performance in all of the tested aspects. However certain differencies between them can still be found. For instance, while the Random Forest classifier had the highest TPR = 0.76, the most successful in detecting negative cases was SVM with TNR = 0.88, see Table III.

Based on the graph of ROC curves in Fig. 12 it can be concluded that Logistic Regression performs the best out of all classifiers when considering lower decision thersholds. MLP on the other hand peaks at higher threshold achieving the highest value of TPR. Furthermore, the MLP model displays the highest AUC value (AUC = 0.77), meaning it classifies the most cases as *positive* out of all classifiers. This alone does not necessarily have to be a correct behaviour but it decreases the number of false negative predictions. In this case the amount of fetuses suffering from acidosis, but classified as healthy, which is arguably a more severe mistake than the other way around.

When it comes to pure $gmean$ the decision for the best classifier suggest itself straight away. It is clear to see, that SVM model achieves the highest average value of $gmean_{SVM} = 0.699$ with a non negligable gap to the second one in Logistic Regression with $gmean_{LR} = 0.661$. Results for all classifiers are in Table IV.

As already stated, it is difficult to select one model that stands out. The choice for the best model would probably be motivated by what would be the most important parameter for given application. If the correctly classified positive cases matter the most, the choice would probably point to Random Forest, since it showcases the highest TPR = 0.76. However if the application demanded the lowest number of false positives, SVM would be the model–to–go because of its FPR = 0.12. For this application, i.e. testing the models based on $gmean$ score, the SVM model is classified as the best one, due to the average $gmean$ it exhibits, see Fig. 13 and Table IV.

VII. Conclusion

In the presented work, four different binary classification models were designed and tested. First, the available dataset was detailed. Then each of the classifiers, namely Logistic Regression, Support Vector Machine, Random Forest and Multi–layer Perceptron were introduced. This was done by first briefly describing the classification method in general and then diving deeper into the particular model and its hyperparameters. In Section V the model's performances were tested and compared using numerous metrics, such as confusion matrices, ROC curves, AUC, TPR, FPR etc. Furthermore, the best classifier in terms of $gmean$ score was chosen. Finally, in Section VI the results were discussed and reviewed once more.

The performance of the models was somewhat comparable, however not all models performed the same in all tested criteria. While the Random Forest was best at correctly classifying acidosis cases with TPR = 0.76, Support Vector Machine was most succesful in detecting healthy fetuses with TNR = 0.88. Due to the factuality that the chosen classifier will be tested solely using $gmean$ score, the best classifier in terms of this metric was SVM with average $gmean$ = 0.699 from 50 runs.

References

[1] Spilka J.: B(E)3M33UI *Semester project 1:Fetal acidosis detection*, 2021, available online at `https://cw.fel.cvut.cz/wiki/_media/courses/ui/tasks/fetal_acidosis_detection.pdf`
[2] Chawla N.V, Bowyer K. W., Hall L. O., Kegelmeyer W. P.: SMOTE*: synthetic minority over-sampling technique,* Journal of artificial intelligence research, 321-357, 2002.
[3] Posik P.: *Neural Networks*, B(E)3M33UI lecture.