CS-E5890 - Statistical Genetics and Personalised Medicine
Machine learning in neonatal intensive care

Maximilian Proll Michele Vantini

May 16, 2018

Abstract

In this work, different classification models are explored, in order to classify simulated patient data for predict in-hospital mortality. Given a simulated data set of 700 patients, composed of both basic data and time-series data, the objective is to predict if they will die in the NICU or if they will survive it. In this project, a number of classification techniques has been assessed for classifying the basic data and the time-series data separately. In particular, for classifying time-series data, different statistics are extracted from the time-series. The obtained results are clearly suggesting that the different models are correctly learning the separation between the data points. Finally, the combination of the two groups of features allows to produce high-quality prediction by combining the results of different classifier.

1 Introduction

Nowadays, there are lot of concerns about newborn babies, particularly in the case of preborn infants. Indeed, in this situation babies receive particular treatments in the Neonatal Intensive Care Unit (NICU). The main reason is that there exists a correlation between premature births and developmental issues. Therefore, a number of techniques has been developed for predicting the risk of death for the monitored patients. Specifically, combining several parameters, such as heart rate, blood pressure and birth weight, it is possible to obtain a single value to use for prediction.

The approach in this work is a little different, indeed, all the features provided in the simulated data has been used more extensively. This means that the different classifiers have been provided directly with all the features in order to further study the correlation between the possible causes (features) and the final results (the patient died or survived).

2 Data

The data that has been used in this work consists of simulated data about 700 patients. The data are divided into two data set:

- Basic data: for each patient the features that are reported are:
  - ga - Gestational age at birth (in days)
  - bw - Birth weight (g)
- Time-series data: this data set contains for each patient the time-series made of 24 hours of measurement of:
  - ABP_S - Arterial blood pressure, systolic

- ABP_M - Arterial blood pressure, mean
- ABP_D - Arterial blood pressure, diastolic
- HR_{ECG} - Heart rate (ECG)
- SpO_2 - Oxygen saturation (pulse oximetry)

Of all the patients, 500 represent the training set, hence, for them it is also present the label: 1 if the patient survived, 2 otherwise.

In Figure 1, you can find the plot of the provided training set for the basic data.

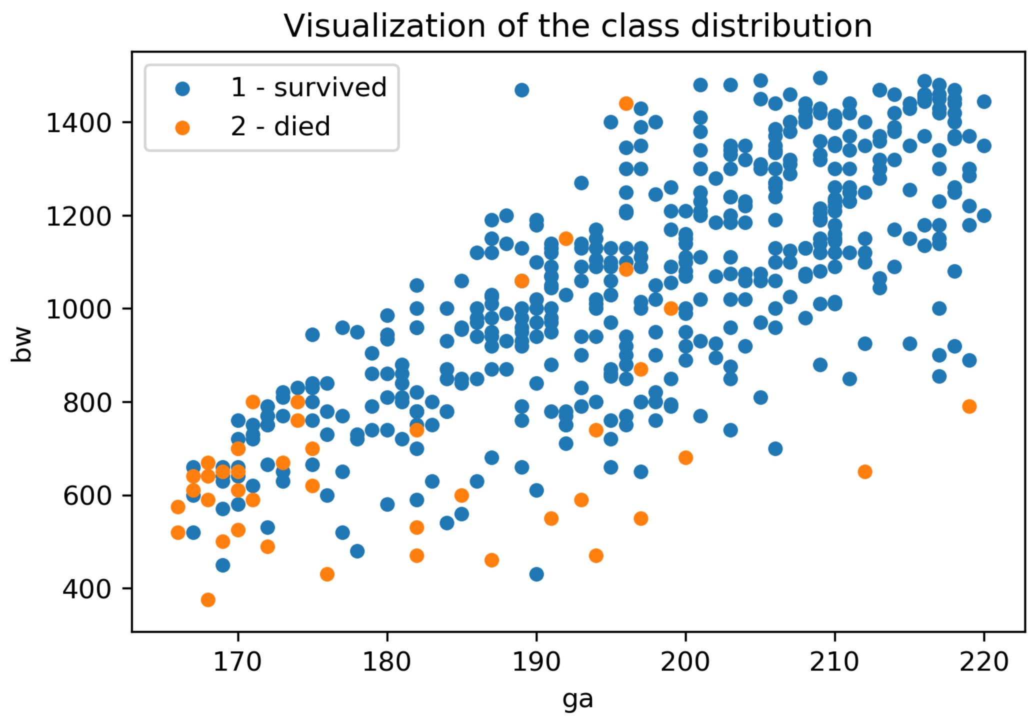

Figure 1: Distribution of the training data

3 Method

In order to build a strong machine learning algorithm that can predict accurately the in-hospital mortality we used and evaluated a wide range of existing ML methods, that are used for binary classification. The ML methods we used are:

- Logistic Regression
- Decision Trees
- Support Vector Machine (SVM)
- Gaussian Naive Bayes
- Gaussian Processes

Those methods were then applied separately to the basic data set and to the extracted statistics from the time-series data and eventually to the combined information of both data sets.

The following subsections give a short explanation of the different methods used.

Logistic Regression

Logistic regression is a regression model where the dependent variable is categorical. In our case the dependent variable is a binary variable. But logistic regression can also be applied to cases where the dependent variable has more than two outcome categories, it is then called multinomial logistic regression [1].

Logistic regression uses a predictor map $h(.)$ with $h(\mathbf{x}) \in [0, 1]$. One common choice for the predictor function h is the so-called sigmoid function $\sigma(z)$:

$$h^{(\mathbf{w},b)}(\mathbf{x}) = \sigma\left(\mathbf{w}^T\mathbf{x} + b\right) \text{ with } \sigma(z) := \frac{1}{1 + \exp(-z)}$$

In contrast to linear regression there is no closed form solution for the cost-minimising parameters, which is why gradient descent is used to find those optimal parameters.

Decision Trees

Decision trees are a very useful tool when predicting a target value as a trained decision tree is simple to understand and to interpret. Classification trees are a type of tree models where the target variable is a discrete set of values. In these tree structures, the leaves represent those class labels and branches represent conjunctions of features that lead to those class labels [2].

When constructing or learning the tree structure the algorithm generally work top-down and choose those conjunctions of features at each step that best splits the set of items. Depending on the algorithm different metrics are used to choose the 'best' conjunction of features. But generally these metrics measure the homogeneity of the target variable within the subsets.

Support Vector Machine

Support Vector Machines (SVM) construct a hyperplane in a high- dimensional space, which then is used for classification or regression [3]. The premise of SVM is to select a particular hyperplane which results in a good separation of training points. A good separation is achieved by a hyperplane that has the largest distance to the nearest training-data point of any class. In other words SVM finds those hyperplane that maximise the functional margin. After the training a SVM are used to categorise the test data according to on which side of the hyperplane the prediction lies.

Gaussian Naive Bayes

In general naive Bayes classifiers are the Bayesian adaptation of a probabilistic classifiers, they apply Bayes' theorem with strong (*naive*) independence assumptions between the features [4]. For continuous data one typically assumes a gaussian distribution of the data within each class.

First the data has to be segmented by the class, then the mean and variance in each class has to be computed. μ_k is the mean of the values in x and σ_k^2 denotes the variance of the values in x associated with class C_k. The probability of some observation value v given a class C_k follows then Normal distribution parameterised by μ_k and σ_k^2:

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

3.1 Gaussian Processes Classification

Gaussian processes(GP), other than for regression task, can be used also for classification[5]. Essentially, what we have to change from the regression case, that consist of using a Gaussian Process as prior distribution for the model, is using a "squashing function" $\phi : \mathcal{R} \rightarrow (0, 1)$ in this way:

$$p(y_n \mid x_n) = \phi(y_n \cdot f(x_n))$$

where $y_n \in \{-1, 1\}$. In this way the output of the GP is mapped to the probability of belonging to one of the two classes $\{-1, 1\}$. The common form used for the squashing function is a Gaussian CDF:

$$\phi(x) = \int_{-\inf}^{x} \mathcal{N}(z \mid 0, 1) dz$$

3.2 Modelling of basic data

In the case of the basic data, we can build classifiers on the base of the two collected features gestational age at birth *ga* and birth weight *bw*. Therefore, we applied and assessed the above reported classification methods. For assessing each individual classifier, we verified some of the most common score that can be computed during the training, namely mis-classification count, precision, recall, and accuracy. However, we spent time also checking visually that the classifier produced meaningful results. This visual check can be done by plotting the classification results on the test set and verify that graphically they resemble the training set. The objective is to select sensible classifier for this particular task. After checking both the scores and the results on the test set, the selected classifiers become part of a voting system. In this voting system it is sufficient that only one classifier labels a data point as "died", to have "died" as final classification for that data point. From a medical prospective, this is a safe choice, since it reduces the possibility of false negative.

3.3 Modelling time-series data

In order to model time-series data, we decided to extract the following statistics from each time-series [6]:

- Mean and variance
- Intercept and slope of linear regression on the data

On the base of this feature extrapolation, we applied the same classifiers presented above. Again, by checking both visually and the scores obtained by the different classifiers, one can select sensible classifiers to be part of the final voting system. In this case to graphically check the results on the test set, we mapped back the results on time-series in the (ga, bw) space of the basic data.

3.4 Combining both data frames

Ultimately we augment the basic data with the information gained from the time-series data mentioned in the section above. This should theoretically lead to a richer data set and we hope to build better classifiers by combining the information of both the basic and the time-series data.

4 Results

In this section, the results on basic data (4.1), time-series data (4.2) and the combination of the two data sets (4.3) are presented.

4.1 Basic data

The score of the tested method on the training set are reported in Table 1. Average mis-classification count, precision, recall and accuracy over the results of 5-fold cross validation on the training set are reported. In addition, in Figure 2 the results on the test set for the different classifier are presented.

Classifier	mis-classification count	Precision	Recall	Accuracy
Gaussian Processes (RBF)	8.0	0.986	0.919	0.929
SVM (RBF)	14.0	0.866	0.859	0.977
Decision tree	12.19	0.943	0.877	0.924
Logistic regression	8.0	0.995	0.919	0.923
Gaussian Naive Bayes	10.6	0.925	0.893	0.956

Table 1: Average mis-classification count, precision, recall and accuracy over the results of 5-fold cross validation on the training set.

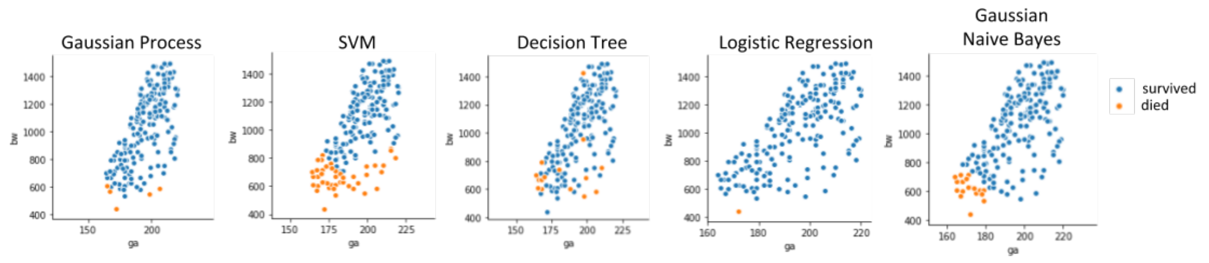

Figure 2: Basic data: Classifier results on the test set.

By combining the predictions of the different classifiers, we obtain the results reported in Figure 3. In this case, it is sufficient for a data point to be classified as "died" by only one classifier to be classified as "died" in these results. This is a conservative approach as we want to build a classifier, which reduces the false positives in its predictions.

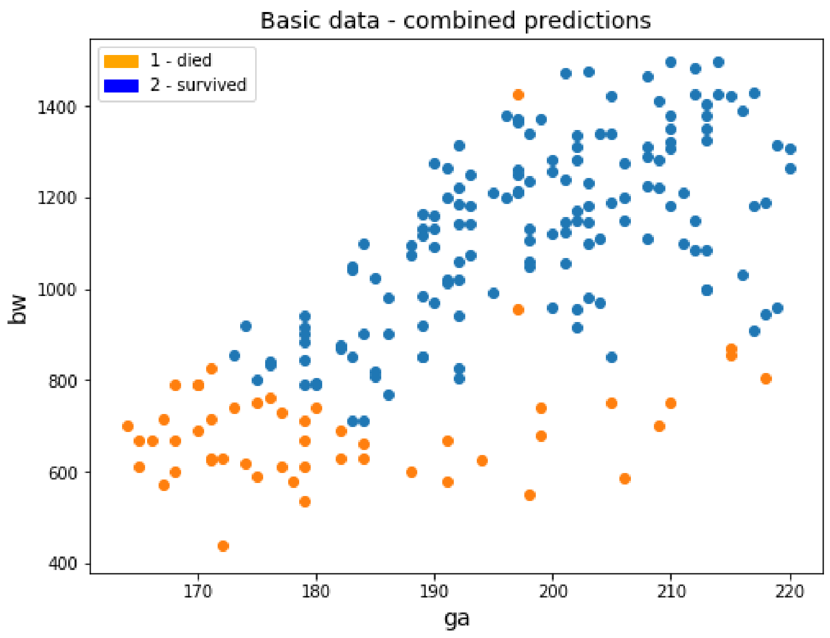

Figure 3: Classification results obtained by combining the opinions of different classifiers on the test set

4.2 Time-series data

As for the basic data, the score of the tested method on the training set are reported in Table 2. Average mis-classification count, precision, recall and accuracy over the results of 5-fold cross validation on the training set are reported. In addition, in Figure 4 the results on the test set for the different classifier are presented. It is important to notice that the results of the classification has been plotted in the feature space of the basic data in order to get a meaningful visualization. Regarding Figure 5 we did not include the Gaussian Naive Bayes classifier as it provides a very strange classification, indeed there are definitely too many deaths.

Classifier	mis-classification count	Precision	Recall	Accuracy
Gaussian Processes (RBF)	9.0	0.986	0.909	0.920
SVM (RBF)	28.0	0.727	0.719	0.959
Decision tree	15.80	0.881	0.841	0.942
Logistic regression	8.59	0.995	0.914	0.917
Gaussian Naive Bayes	45.39	0.526	0.546	0.962

Table 2: Average mis-classification count, precision, recall and accuracy over the results of 5-fold cross validation on the training set.

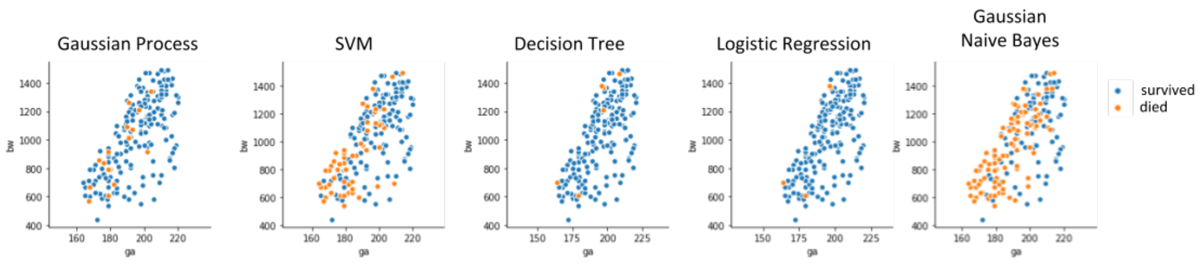

Figure 4: Time-series data: Classifier results on the test set.

By combining the predictions of the different classifiers, we obtain the results reported in Figure 5. As for the basic data, it is sufficient for a data point to be classified as "died" by only one classifier to be classified as "died" in these results.

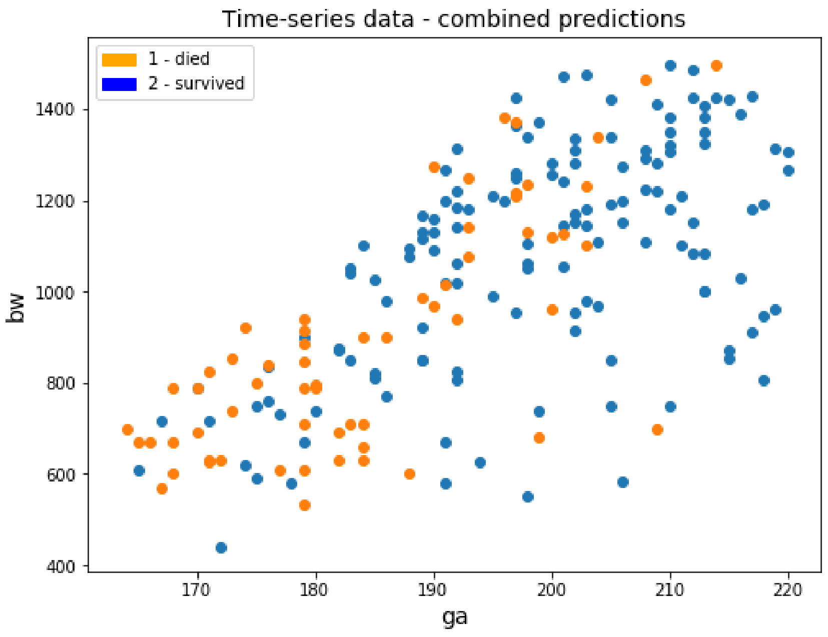

Figure 5: Classification results obtained by combining the opinions of different classifiers on the test set (Gaussian Naive Bayes excluded)

4.3 Basic data and time-series data combined

As for the basic and time-series data, the score of the tested method on the training set are reported in Table 3. Average mis-classification count, precision, recall and accuracy over the results of 5-fold cross validation on the training set are reported. In addition, in Figure 6 the results on the test set for the different classifier are presented. It is important to notice that the results of the classification has been plotted in the feature space of the basic data in order to get a meaningful visualization. Regarding Figure 7 we did not include the Gaussian Naive Bayes classifier as it provides a very strange classification, indeed there are definitely too many deaths.

Classifier	mis-classification count	Precision	Recall	Accuracy
Gaussian Processes (RBF)	8.59	0.986	0.914	0.924
SVM (RBF)	17.6	0.831	0.823	0.972
Decision tree	10.6	0.942	0.893	0.941
Logistic regression	9.19	0.980	0.908	0.923
Gaussian Naive Bayes	33.79	0.650	0.662	0.970

Table 3: Average mis-classification count, precision, recall and accuracy over the results of 5-fold cross validation on the training set.

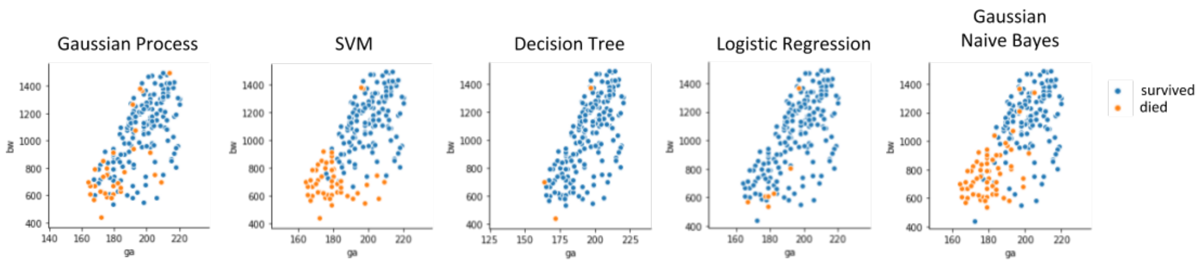

Figure 6: Combined data: Classifier results on the test set.

By combining the predictions of the different classifiers, we obtain the results reported in Figure 7. As for the basic and time-series data, it is sufficient for a data point to be classified as ”died” by only one classifier to be classified as ”died” in these results.

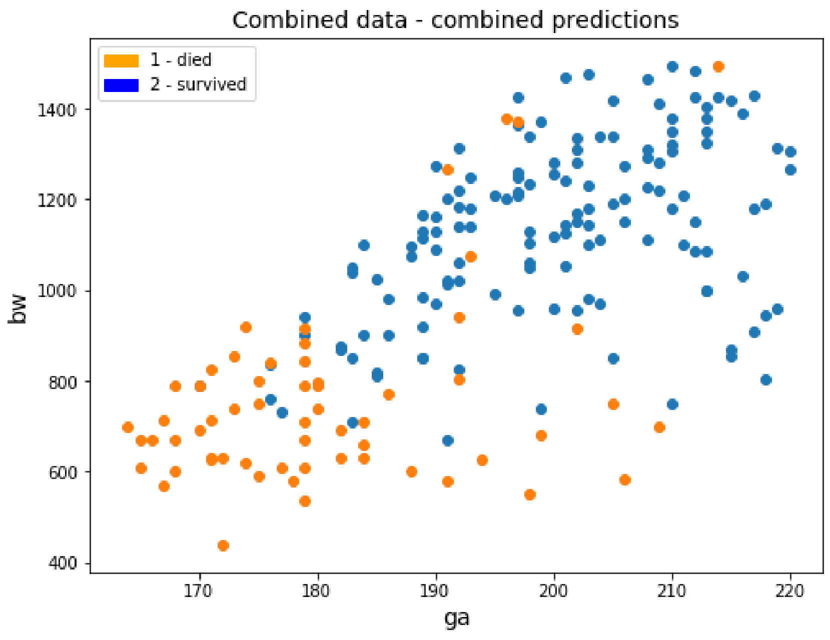

Figure 7: Classification results obtained by combining the opinions of different classifiers on the test set (Gaussian Naive Bayes excluded)

As a results, the following list summarises the patient ID’s in the test set that have been classified as not surviving the NICU: 501, 502, 520, 522, 532, 533, 536, 540, 541, 542, 543, 547, 548, 549, 551, 552, 563, 564, 566, 567, 574, 587, 590, 591, 592, 597, 599, 604, 606, 607, 608, 609, 611, 613, 614, 615, 625, 627, 635, 639, 640, 644, 648, 649, 652, 654, 659, 663, 665, 666, 669, 680, 682, 685, 686, 688, 691, 693, 696, 697

5 Discussion

In this section we discuss the results obtained on the basic data, the time-series data and finally the combined data.

Basic data

What we can see from the results in Table 1 is that on average the methods that are performing less in terms of mis-classification count (SVM, decision tree and Gaussian naive Bayes), are the ones that actually are learning more from the data. This can be clearly seen from the plots of the results on the test set reported in Figure 2. Moreover, by combining the classification from the different methods, we can clearly see that the separation that is learned is the area under more or less 800g. With this split of the data we can actually produce meaningful prediction.

Time-series data

As we can see from the results on time-series, the pattern is not so clear as in the basic data. In particular, the Gaussian Naive Bayes classifier is producing very confusing data. Therefore, our sensible choice has been to ignore it in the combination of the classification results. However, it seems that time-series data are not contributing significantly to improve the performance of the model.

Basic data and time-series data combined

In conclusion, by combining the two data set, we can clearly see that the classifiers are relying more on the basic data. This can be noticed by looking at the predictions on the test set. Again, Gaussian Naive Bayes predictions are quite confusing. Hence, we excluded this classifier from the final combined prediction. We believe that the combined results reported in Figure 7 can be considered our final classification results. They represent the combination of the good baseline provided by the basic data and the information added through time-series data.

6 Conclusion

The task of this report was to predict in-hospital mortality based on simulated data. We used various classifiers ranging from logistic regression and decision trees to SVMs and Gaussian naive Bayes and finally Gaussian processes to make a robust binary classification. Those classifiers were applied to the basic data set and the time-series data set alone and eventually to the combined data set. In order to gain one global prediction we merged the separate classifiers with the rule, that it is sufficient for a data point to be classified as "died" if only one classifier labels the data point as "died".

As discussed in Section 5 we can conclude that the basic data set alone is already giving a reasonably good amount of information for predicting in-hospital mortality. On the contrary the time-series data set does not contain enough information in order to build a robust classifier. Hence a pure reliance on the time-series data is not recommended and information covered in the basic data should be rather used.

When both data sets are combined the predictions are strongly influenced by the basic data set. Which is why we suggest to focus on gaining more information in the same style as the basic data set rather than time-series data. Other single-valued patient scores like Apgar, CRIB-II, SNAP-II or SNAPPE-II should be added for a more precise prediction of the in-hospital mortality. Values contained in those scores are generally easy to compute by hand and do not need special equipment. Examples of additional values are: skin color, pulse rate, activity, respiration effort, sex, admission temperature. There are many more values one could add to enrich the basic data and which help the mentioned classifies to give more accurate and robust predictions.

7 References

[1] D. Freedman. *Statistical Models: Theory and Practice.* Cambridge University Press, 2005.

[2] Lior Rokach and Oded Maimon. *Data Mining With Decision Trees: Theory and Applications.* World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2nd edition, 2014.

[3] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sep 1995.

[4] David J. Hand and Keming Yu. Idiot's bayes: Not so stupid after all? *International Statistical Review / Revue Internationale de Statistique*, 69(3):385–398, 2001.

[5] David J. C. MacKay. *Information Theory, Inference & Learning Algorithms.* Cambridge University Press, New York, NY, USA, 2002.

[6] Olli-Pekka Rinta-Koskia, Simo Särkkä, Jaakko Hollména, Markus Leskinenc, and Sture Anderssonc. Gaussian process classification for prediction of in-hospital mortality among preterm infants. *Neurocomputing*, 02 2018.

8 Appendix

Appendix A

The following pages show our source code for building the classifiers.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import auc
from sklearn.metrics import roc_curve
from sklearn.metrics import recall_score

from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.gaussian_process.kernels import ConstantKernel as C
from sklearn.model_selection import KFold
import matplotlib.patches as mpatches


# Importing the data
data = pd.read_csv("neonatal_basic_data.csv")
training_set = data[data['death']!=0]
training_labels = training_set['death']
training_data = training_set[['ga', 'bw']]
test_set = data[data['death']==0]
#test_labels = test_set['death']
test_data = test_set[['ga', 'bw']]


# ### Scaling the data
scaler = StandardScaler()
training_data_stand = scaler.fit_transform(training_data)
test_data_stand = scaler.fit_transform(test_data)

minmaxscaler = MinMaxScaler()
training_data_minmax = minmaxscaler.fit_transform(training_data)
test_data_minmax = minmaxscaler.fit_transform(test_data)


# ## Visualization of the class distribution
labels = training_labels.as_matrix()
classes, distribution = np.unique(labels, return_counts=True)
plt.bar(classes, distribution)
plt.xticks(classes, ["survived", "died"])
plt.show()

sns.pairplot(training_set[['ga', 'bw', 'death']]
             , hue="death"
             , vars=['ga', 'bw']
             , diag_kind="kde"
             #, kind="reg"
             , size=3
            )
plt.show()

#kernel = 1.0 * C(constant_value=1.0) + 1.0 * RBF(length_scale=1.0)
kernel = 1.0 * RBF(length_scale=1.0)
clfs = [(GaussianProcessClassifier(kernel=kernel, optimizer=None),"Gaussian Processes(RBF)")
        , (svm.SVC(class_weight='balanced', kernel='rbf'),"SVM")
        , (DecisionTreeClassifier(),"Decision tree")
        , (LogisticRegression(),"Logistic regression")
        , (GaussianNB(),"Gaussian naive Bayes")
       ]

for clf, method_name in clfs:
    misclassif_lst = []
    precision_lst = []
    accuracy_lst = []
    recall_lst = []
    f1_lst = []

    kf = KFold(n_splits=5)
    for train_index, test_index in kf.split(training_data_stand):
        X_train, X_test = training_data_stand[train_index], training_data_stand[test_index]
        y_train, y_test = training_labels[train_index], training_labels[test_index]
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        misclassif = np.sum(y_pred!=y_test)
        precision = precision_score(y_pred, y_test)
        accuracy = accuracy_score(y_pred, y_test)
        recall = recall_score(y_pred, y_test)
        f1 = f1_score(y_pred, y_test)

        misclassif_lst.append(misclassif)
```

```
            precision_lst.append(precision)
            accuracy_lst.append(accuracy)
            recall_lst.append(recall)

            f1_lst.append(f1)

        plt.plot(range(5), precision_lst, label="Precision")
        plt.plot(range(5), accuracy_lst, label="Accuracy")
        plt.plot(range(5), recall_lst, label="Recall")
        plt.plot(range(5), f1_lst, label="F1-score")
        plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
        plt.title(method_name)
        plt.show()

        print("Average on k-fold cross validation")
        print("misclassif error:", np.mean(misclassif_lst))
        print("precision:", np.mean(precision_lst))
        print("accuracy:", np.mean(accuracy_lst))
        print("recall:", np.mean(recall_lst))
        print("f1:", np.mean(f1_lst))

        clf.fit(training_data_stand, training_labels)
        p = clf.predict(test_data_stand)

        classes, distribution = np.unique(p, return_counts=True)
        plt.bar(classes, distribution)
        plt.title(method_name)
        plt.xticks(classes, ["survived", "died"])
        plt.show()

        pred = pd.DataFrame(data={'ga':test_set['ga'].as_matrix()
            , 'bw':test_set['bw'].as_matrix(), 'death':p})

        if method_name=="Gaussian Processes(RBF)":
            gp_pred = pred
        elif method_name=="SVM":
            svm_pred = pred
        elif method_name=="Decision tree":
            dt_pred = pred
        elif method_name=="Logistic regression":
            lr_pred = pred
        elif method_name=="Gaussian naive Bayes":
            gnb_pred = pred

        sns.pairplot(pred[['ga', 'bw', 'death']]
                     , hue="death"
                     , vars=['ga', 'bw']
                     #, diag_kind="kde"
                     #, kind="reg"
                     , size=3
                    )
        plt.title(method_name)
        plt.show()


# Mixing prediction
# Final prediction are:
#     - Survived: if all the models agree that the patient survived
#     - Died: if at least one of the model classify the patient as died

pred = pd.DataFrame(data={'patientid':test_set['patientid'].as_matrix()
        , 'ga':test_set['ga'].as_matrix(), 'bw':test_set['bw'].as_matrix()
    , 'death_dt':dt_pred['death'], 'death_svm':svm_pred['death']
    , 'death_lr':lr_pred['death']
    , 'death_gnb':gnb_pred['death'], 'death_gp':gp_pred['death']})

died = pred.loc[(pred['death_lr'] == 2) | (pred['death_dt'] == 2)
        | (pred['death_svm']==2) | (pred['death_gp'] == 2)
        | (pred['death_gnb']==2)]
survived = pred.loc[(pred['death_lr'] == 1) & (pred['death_dt'] == 1)
```

```
        & (pred['death_svm']==1) & (pred['death_gp'] == 1)
        & (pred['death_gnb'] == 1)]
plt.figure(figsize=(8,6))
plt.scatter(survived['ga'], survived['bw'])
plt.scatter(died['ga'], died['bw'])
plt.xlabel("ga", fontsize=14)
plt.ylabel("bw", fontsize=14)

orange_patch = mpatches.Patch(color='orange', label='1 - died')
blue_patch = mpatches.Patch(color='blue', label='2 - survived')
plt.legend(handles=[orange_patch, blue_patch])
plt.title("Basic data - combined predictions", fontsize=14)
plt.show()


# ## Time-series data classification
ts = pd.read_csv("timeseries.csv", names=['patientid','mean_ABP_S','var_ABP_S'
        ,'slope_ABP_S','intercept_ABP_S','mean_ABP_M','var_ABP_M','slope_ABP_M'
        ,'intercept_ABP_M','mean_ABP_D','var_ABP_D','slope_ABP_D','intercept_ABP_D'
        ,'mean_HR_ECG','var_HR_ECG','slope_HR_ECG','intercept_HR_ECG','mean_SpO2'
        ,'var_SpO2','slope_SpO2','intercept_SpO2'])
training_data = ts[ts['patientid'].isin(data[data['death']!=0]['patientid'])]
training_data.drop(['patientid'], axis=1)
test_data = ts[ts['patientid'].isin(data[data['death']==0]['patientid'])]
test_data.drop(['patientid'], axis=1)
ts.shape

scaler = StandardScaler()
training_data_stand = scaler.fit_transform(training_data)
test_data_stand = scaler.fit_transform(test_data)

minmaxscaler = MinMaxScaler()
training_data_minmax = minmaxscaler.fit_transform(training_data)
test_data_minmax = minmaxscaler.fit_transform(test_data)

# Evaluation of classifier via K-fold cross-validation
for clf, method_name in clfs:
    misclassif_lst = []
    precision_lst = []
    accuracy_lst = []
    recall_lst = []
    f1_lst = []

    kf = KFold(n_splits=5)
    for train_index, test_index in kf.split(training_data_stand):
        X_train, X_test = training_data_stand[train_index], training_data_stand[test_index]
        y_train, y_test = training_labels[train_index], training_labels[test_index]
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        misclassif = np.sum(y_pred!=y_test)
        precision = precision_score(y_pred, y_test)
        accuracy = accuracy_score(y_pred, y_test)
        recall = recall_score(y_pred, y_test)
        f1 = f1_score(y_pred, y_test)

        misclassif_lst.append(misclassif)
        precision_lst.append(precision)
        accuracy_lst.append(accuracy)
        recall_lst.append(recall)

        f1_lst.append(f1)

    plt.plot(range(5), precision_lst, label="Precision")
    plt.plot(range(5), accuracy_lst, label="Accuracy")
    plt.plot(range(5), recall_lst, label="Recall")
    plt.plot(range(5), f1_lst, label="F1-score")
    plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
    plt.title(method_name)
    plt.show()
    print("Average on k-fold cross validation")
    print("misclassif_error:", np.mean(misclassif_lst))
    print("precision:", np.mean(precision_lst))
    print("accuracy:", np.mean(accuracy_lst))
    print("recall:", np.mean(recall_lst))
    print("f1:", np.mean(f1_lst))

    clf.fit(training_data_stand, training_labels)
    p = clf.predict(test_data_stand)

    classes, distribution = np.unique(p, return_counts=True)
    plt.bar(classes, distribution)
    plt.title(method_name)
    plt.xticks(classes, ["survived", "died"])
    plt.show()

    pred = pd.DataFrame(data={'ga':test_set['ga'].as_matrix()
        , 'bw':test_set['bw'].as_matrix(), 'death':p})

    if method_name=="Gaussian Processes(RBF)":
        gp_pred = pred
    elif method_name=="SVM":
        svm_pred = pred
    elif method_name=="Decision tree":
        dt_pred = pred
    elif method_name=="Logistic regression":
        lr_pred = pred
    #elif method_name=="Gaussian naive Bayes":
    #    gnb_pred = pred

    sns.pairplot(pred[['ga', 'bw', 'death']]
                 , hue="death"
                 , vars=['ga', 'bw']
                 , diag_kind="kde"
                 #, kind="reg"
                 , size=3
                )
    plt.title(method_name)
    plt.show()

pred = pd.DataFrame(data={'patientid':test_set['patientid'].as_matrix()
        , 'ga':test_set['ga'].as_matrix(), 'bw':test_set['bw'].as_matrix()
    , 'death_dt':dt_pred['death'], 'death_svm':svm_pred['death']
    , 'death_lr':lr_pred['death'], 'death_gp':gp_pred['death']})

died = pred.loc[(pred['death_dt'] == 2) | (pred['death_svm']==2)
        | (pred['death_gp'] == 2) | (pred['death_lr']==2)]
survived = pred.loc[(pred['death_dt'] == 1) & (pred['death_svm']==1)
        & (pred['death_gp'] == 1) & (pred['death_lr'] == 1)]
plt.figure(figsize=(8,6))
plt.scatter(survived['ga'], survived['bw'])
plt.scatter(died['ga'], died['bw'])
plt.xlabel("ga", fontsize=14)
plt.ylabel("bw", fontsize=14)

orange_patch = mpatches.Patch(color='orange', label='1 - died')
blue_patch = mpatches.Patch(color='blue', label='2 - survived')
plt.legend(handles=[orange_patch, blue_patch])
plt.title("Time-series data - combined predictions", fontsize=14)
plt.show()


# Combining time-series with basic data
# ts, data
fulldata = data.merge(ts, on='patientid')
training_set = fulldata[fulldata['death']!=0]
training_labels = training_set['death']
training_data = training_set.drop(['patientid', 'death'], axis=1)
test_set = fulldata[fulldata['death']==0]
test_data = test_set.drop(['patientid', 'death'], axis=1)
```

```
scaler = StandardScaler()
training_data_stand = scaler.fit_transform(training_data)
test_data_stand = scaler.fit_transform(test_data)

minmaxscaler = MinMaxScaler()
training_data_minmax = minmaxscaler.fit_transform(training_data)
test_data_minmax = minmaxscaler.fit_transform(test_data)

for clf, method_name in clfs:
    misclassif_lst = []
    precision_lst = []
    accuracy_lst = []
    recall_lst = []
    f1_lst = []

    kf = KFold(n_splits=5)
    for train_index, test_index in kf.split(training_data_stand):
        X_train, X_test = training_data_stand[train_index], training_data_stand[test_index]
        y_train, y_test = training_labels[train_index], training_labels[test_index]
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        misclassif = np.sum(y_pred!=y_test)
        precision = precision_score(y_pred, y_test)
        accuracy = accuracy_score(y_pred, y_test)
        recall = recall_score(y_pred, y_test)
        f1 = f1_score(y_pred, y_test)

        misclassif_lst.append(misclassif)
        precision_lst.append(precision)
        accuracy_lst.append(accuracy)
        recall_lst.append(recall)

        f1_lst.append(f1)

    plt.plot(range(5), precision_lst, label="Precision")
    plt.plot(range(5), accuracy_lst, label="Accuracy")
    plt.plot(range(5), recall_lst, label="Recall")
    plt.plot(range(5), f1_lst, label="F1-score")
    plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
    plt.title(method_name)
    plt.show()

    print("Average on k-fold cross validation")
    print("misclassif error:", np.mean(misclassif_lst))
    print("precision:", np.mean(precision_lst))
    print("accuracy:", np.mean(accuracy_lst))
    print("recall:", np.mean(recall_lst))
    print("f1:", np.mean(f1_lst))

    clf.fit(training_data_stand, training_labels)
    p = clf.predict(test_data_stand)

    classes, distribution = np.unique(p, return_counts=True)
    plt.bar(classes, distribution)
    plt.title(method_name)
    plt.xticks(classes, ["survived", "died"])
    plt.show()

    pred = pd.DataFrame(data={'ga':test_set['ga'].as_matrix()
        , 'bw':test_set['bw'].as_matrix(), 'death':p})

    if method_name=="Gaussian Processes(RBF)":
        gp_pred = pred
    elif method_name=="SVM":
        svm_pred = pred
    elif method_name=="Decision tree":
        dt_pred = pred
    elif method_name=="Logistic regression":
        lr_pred = pred
    #elif method_name=="Gaussian naive Bayes":
    #     gnb_pred = pred

    sns.pairplot(pred[['ga', 'bw', 'death']]
                 , hue="death"
                 , vars=['ga', 'bw']
                 , diag_kind="kde"
                 #, kind="reg"
                 , size=3
                )
    plt.title(method_name)
    plt.show()

pred = pd.DataFrame(data={'patientid':test_set['patientid'].as_matrix()
    , 'ga':test_set['ga'].as_matrix(), 'bw':test_set['bw'].as_matrix()
    , 'death_dt':dt_pred['death'], 'death_svm':svm_pred['death']
    , 'death_lr':lr_pred['death'], 'death_gp':gp_pred['death']})

died = pred.loc[(pred['death_dt'] == 2) | (pred['death_svm']==2)
        | (pred['death_gp'] == 2) | (pred['death_lr']==2)]
survived = pred.loc[(pred['death_dt'] == 1) & (pred['death_svm']==1)
        & (pred['death_gp'] == 1) & (pred['death_lr'] == 1)]
plt.figure(figsize=(8,6))
plt.scatter(survived['ga'], survived['bw'])
plt.scatter(died['ga'], died['bw'])
plt.xlabel("ga", fontsize=14)
plt.ylabel("bw", fontsize=14)

orange_patch = mpatches.Patch(color='orange', label='1 - died')
blue_patch = mpatches.Patch(color='blue', label='2 - survived')
plt.legend(handles=[orange_patch, blue_patch])
plt.title("Combined data - combined predictions", fontsize=14)
plt.show()

died["patientid"].as_matrix()
```

Appendix B

The following is the code for the feature extration from the time-series data.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import linear_model

# Importing the data
data = pd.read_csv("neonatal_24h.csv", names=['ID', 'time', 'ABP_S', 'ABP_M', 'ABP_D','HR_ECG', 'SpO2']

IDs = np.unique(data['ID'])

data_per_ID = []
for ID in IDs:
    data_per_ID.append(data.loc[data['ID']==ID].sort_values('time'))


# Visualisation of one ID
features = ['ABP_S', 'ABP_M', 'ABP_D','HR_ECG', 'SpO2']
col = ['red', 'blue', 'green', 'black' , 'orange']

fig = plt.figure(figsize=(1.61 * 6, 6))
ax = fig.add_subplot(1, 1, 1)
ID = 100

for y, c in zip(features, col):
    data_per_ID[ID].plot(x='time', y=y, marker='.', kind='scatter', ax=ax, label=y, color=c)
plt.title('Visualisation for ID = ' + str(IDs[ID]) )
plt.ylabel('');
```

```
# extract additional features like mean and variance and also slope and intercept of a linear regressio
mean = []
var = []
slope = []
intercept = []

for i in range(len(IDs)):
    m = []
    v = []
    s = []
    inter = []
    for j, feature in enumerate(features):
        # extract mean and variance
        m.append(np.nanmean(data_per_ID[i], axis=0)[j])
        v.append(np.nanvar(data_per_ID[i], axis=0)[j])

        # create linear regression object
        regr = linear_model.LinearRegression(fit_intercept=True)

        # select time as x and feature as y
        df = data_per_ID[0][['time', feature]].dropna()
        x = np.array(df['time']).reshape(-1, 1)
        y = df[feature]

        if len(df) > 1:
            # train the model using the training sets
            regr.fit(x,y)
            s.append(regr.coef_[0])
            inter.append(regr.intercept_)
        else:
            s.append(np.nan)
            inter.append(np.nan)

    mean.append(m)
    var.append(v)
    slope.append(s)
    intercept.append(inter)

mean = np.array(mean)
var = np.array(var)
slope = np.array(slope)
intercept = np.array(intercept)

# save arrays for future analysis
np.savetxt('mean.csv', mean, delimiter=',')
np.savetxt('var.csv', var, delimiter=',')
np.savetxt('slope.csv', slope, delimiter=',')
np.savetxt('intercept.csv', intercept, delimiter=',')
np.savetxt('IDs.csv', IDs, delimiter=',', fmt='%i')

timeseries = []
for i in range(len(IDs)):
    ts = []
    ts.append(IDs[i])
    for j in range(mean.shape[1]):
        ts.append(mean[i][j])
        ts.append(var[i][j])
        ts.append(slope[i][j])
        ts.append(intercept[i][j])
    timeseries.append(ts)
timeseries = np.array(timeseries)

np.savetxt('timeseries.csv', timeseries, delimiter=',')
# order of the file:

# ID1    mean ABP_S          var ABP_S          slope ABP_S          intercept ABP_S
#             mean ABP_M          var ABP_M          slope ABP_M
intercept ABP_M
#             mean ABP_D          var ABP_D          slope ABP_D
intercept ABP_D
```

```
#                    mean HR_ECG                  var HR_ECG                  slope HR_ECG
intercept HR_ECG
#                    mean SpO2                    var SpO2                    slope SpO2
intercept SpO2

# in total ID + (mean, var, slope, intercept) * ('ABP_S', 'ABP_M', 'ABP_D','HR_ECG', 'SpO2') = 1 + 20 c

print(timeseries.shape)
```