# A Machine-learning Mobile App to support prognosis of Ebola Virus Diseases in an evolving environment

Supervisors: Mary-Anne Hartley, Martin Jaggi

Sina Fakheri

May 2017

## 1 Introduction

Ebola Virus Disease (EVD) comprises a constellation of potentially fatal symptoms caused by infection with a species of the Ebolavirus genus. The virus is thought to drive its pathogenicity by a potent stimulation of the immune system, which creates inflammatory collateral damage to the surrounding tissues. Inflammatory damage to blood vessels and consequent haemorrhage, has placed EVD in a group of diseases known as "haemorrhagic fevers" . For decades since its discovery near Ebola river in DRC in 1976, its impact remained localized and limited to fewer than 1000 infections. However, this changed in 2013 when it spread to West Africa creating an epidemic that infected over 28700 persons across 3 countries among which 11325 died [1]. Recently, the WHO stated that another Ebola epidemic is probable in the near future and urged for improved preparedness [2].

Due to the lack of EVD-specific symptoms, the current symptomatic triage protocol for EVD diagnosis misclassifies more than half of the cases. This puts EVD-negative patients at risk of nosocomial infection and dilutes EVD-specific resources across false-positive cases. Despite its notoriety as a fatal disease, EVD is a heterogeneous disease with outcomes ranging from asymptomatic to fatal (60% fatality rate).

Thus, we need efficient and robust methods for diagnostic and prognostic triage and this is where technology can be of great help.

As Ebola preferentially spreads in the context of poor health-care infrastructure, it is critical that triage tools are cheap and easy to implement. Current state-of-art solutions are both rare and limited. Hartley et al.[3] proposed a scoring system for triage which is based in statistical modeling. The major limitation is due to the fact that this score is static and is thus unable to adapt to

advances in clinical management, changes in patient behaviour or viral evolution. A Machine-learning based solution intrinsically takes these changes into account and it's performance generally improves as more data is available.

We propose a solution via a machine learning smartphone app, which uses symptoms to predict the risk of a person being EVD+ (diagnosis) and in the case that they are EVD+, their risk of mortality (prognosis). This will improve both resource management and the precision of EVD+ detections which in turn slows down the propagation rate of EVD. This solution is cheap, highly portable and easy-to-use. Health care assistants just have to install the app on their Android smartphones. The app then self-adapts to changes in disease presentation, such as could be caused by eventual virus mutations in time and location specificities.

# 2 Methodology

## 2.1 Clinical dataset

We used the GOAL dataset from the retrospective cohort study conducted by Hartley et al.[3] which comprises anonymized patient data collected between December 14, 2014 and November 15, 2015 at the GOAL ETC in Port Loko, Sierra Leone. Data comprised patient demographics, geographic location, clinical signs and symptoms, and laboratory results (for malaria infection and semi-quantitative Ebola viremia), as well as the final patient out- come of death or survival." It contains 575 patients. As we were interessted about prognosis, we only considered EVD+ patients (144 patients).

We also had access to another dataset, from data collected in a district hospital in Sierra Leone, Kenema Government Hospital (KGH). This dataset was evaluated for its applicability to external validation. . However, due to the large number of missing values (more than 80%) and the low number of samples in this dataset, we decided not to use it as such few number of samples can not be of real help to externally validate our model. Additionally, Doing imputation on very few available data with high levels of heterogeneity gives generally poor results.

## 2.2 Missing data and imputation

The four following features contain missing values : referral_time (i.e. the time taken for the patient to present at an Ebola treatment centre since their first symptom), evd_ct (inversely proportional to viral load in patient's body), malaria infection and quarantine.

We used MissMech library from R, by Jamshidian et al., to asses if missing values are MCAR (Missing Completely At Random), a necessary condition in order to use Multiple Imputation techniques as MICE and the hypothesis of MCAR was rejected at 0.05 significance level. Therefore we focused on Single

Imputation techniques.
Missingness impacted 26 patients (20%) of the cohort and we tested imputation by mean, KNN imputation and imputation by most frequent element were tested. Imputation by most frequent is an option proposed by Imputer class from Sklearn. For each feature containing missing values, it takes the element that appear the most frequently (for binary and categorical features) and simply takes the mean for features with continuous values.

The results on classification, for the three imputation techniques used, were very similar. This can be explained by the relatively low proportion of missing values, and the fact that the concerned features were not among the most prevalent for the classification. Additionally, Hartley et al. [3] previously reported these features to be missing completely at random. The fact that 2/3 of features with missing values were binary variables can also explain the small difference between different imputation techniques. Finally, for simplicity and computation speed, we retained the imputation by the most frequent element option. The details are reported in the Results section.

## 2.3 Pre-processing and feature selection

The dataset has two sets of features: the first one concerns information and measurements gathered at arrival, in the Ebola Treatment Center (ETC), i.e. triage, and the second is the same set of information gathered at daily intervals during the patient's admission (determined by 'days_admitted'). This approach allows us to better understand how the evolution of certain measures influence death risk and also helps us to see which subset of features are the most determinant for each prognostic model.

Furthermore, It's also a common practice to standardize data, i.e. subtract the mean and divide by the standard deviation for each dimension, before classification. After this processing, each dimension has zero mean and unit variance. Note that this is not equivalent to data whitening, which additionally de-correlates the dimensions. The aim of standardization is to render features into equivalent numerical bounds, so that they may contribute equally to the neural network output.

Here, standardization was undertaken using StandardScaler from Scikit-learn. We ran once classification algorithms with standardized data and once without: On average, standardized data decreased performance by 22% and thus, we decided not to use standardization. A possible reason for the poor performance of standardization may be that the majority of features (nearly 90%) are binary with with set numerical bounds, and thus, standardization of these features does not help the classifier. However, for feature selection, we did use standardization to be able to compare scaled features.

## 2.4 Feature engineering

A case of interest for health experts is the difference between prediction models across geographic locations and varying time periods, i.e. population selection. This allows us to highlight the nuances in health-care seeking behaviour and public health infrastructure across regions or the evolution of the virus over time (to observe eventual virus mutations). Additionally, it also helps to adjust the treatment with the specificities of each region. The GOAL dataset we used didn't have any location or date related feature. Therefore, to integrate this ability in our model, as a proof of concept, we artificially added a location categorical feature and date feature(s) to the dataset:
For the location feature, using Numpy's random.choice function, we attributed a location (from an array of most prevalent locations) to each patient. The locations could not be attributed to individual patients due to patient privacy and protection of data anonymity. Then, to be able to use this information in our prediction models, we used the DictVectorizer package from Sklearn.feature_extraction library which allows us to add as many binary features as we have distinct locations. For each patient, only one of the newly created binary features has a value of 1. A population is then selected corresponding to a specific location with which we create prediction model.
For the date features, we used Pandas Timestamp to artificially create entry-date and end-date between two fixed dates: we generated dates from March 2013 to March 2016 corresponding to biggest Ebola outbreak. We can then filter the dataset by taking the date(s) specified by entrydateFilter and (optionally) enddateFilter.

To account for different filtering possibilities, we created the 'filtering' variable. Here are possible values :

- 0 : No filtering
- 1 : Filtering only the location
- 2 : Filtering only the entry date
- 3 : Filtering the location and the entry date
- 4 : Filtering between entry date and end date
- 5 : Filtering location between entry date and end date

## 2.5 Algorithms used

For our prediction model on prognosis, we have tried five common Machine learning classification algorithms:

- Logistic Regression : a linear model which finds the best coefficient (weight) for each feature that, when combined linearly, explain data the best.

- KNN (k-nearest neighbors ) : an object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors
- SVM : classifies by using an optimal linear separating hyperplane by using maximal margin classifier.
- Random Forest : averages the prediction of many decision trees (classifies by organizing data features in form of a tree)
- Neural Networks : are organized in a series of layers, where the input vector enters at the left side of the network, which is then projected to a "hidden layer." Each unit in the hidden layer is a weighted sum of the values in the first layer. This layer then projects to an output layer, which is where the desired answer appears.
- Bagging : groups "weak" classifiers together to obtain a stronger classifier

For each classifers, we proceed with the same methodology (See Framework and Sampling section) in order to find the best model (based on MCC).

We have used Python as programming language and Scikit-learn library for the machine learning algorithms.

## 2.6 Framework and Sampling

In general, cross-validation is an excellent way to learn models with low bias and avoid over-fitting but it's drawback is that information from test samples can leak to the model and even contribute to over-fitting. In presence of large number of models the risk for over-fitting increases and the performance estimated by cross validation is no longer an effective estimate of generalization.

To avoid this, for our hyper-parameter optimization, we proceed in two steps. First we did Hold-out validation, which splits the dataset into a training and testing portions . Hold-out is also biased (as train/test splits are obtained by a random shuffle of the data and the process is run only once, we can, for example, obtain by chance a 'favorable' shuffle which gives very high classification score) if done once therefore we did 10'000 random Hold-outs, independently, given by different seeds of train_test_split function. We used 15% as test size. In the second step, we used GridSearchCV with 10-folds to confirm the results from first part by exhaustive consideration of all parameters' permutations.

## 2.7 Metric

For binary classification, many metrics exist to evaluate the performance of the classifier (Accuracy, Balanced Error Rate (BER), F-1 measure, Matthews Correlation coefficient (M.C.C), etc.).
We have chosen to work with M.C.C as main metric for this project : it takes into account true and false positives and negatives and is generally regarded as

a balanced measure which can be used even if the classes are of very different sizes, which is our case. It returns a value between 1 and +1. A coefficient of +1 represents a perfect prediction, 0 no better than random prediction and 1 indicates total disagreement between prediction and observation. It's obtained as follows:

$$M.C.C = \frac{TP * TN + FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

To give a better perspective, here is an example of confusion matrix with its corresponding M.C.C. This is what we obtained in one simulation of our Logistic Regression model on a test set of 25%. Note that this is very near the average result of our classifiers.

$$\begin{bmatrix} 12 & 2 \\ 2 & 20 \end{bmatrix} = 0.766(M.C.C)$$

### 2.8 Class imbalance

From the 144 patients, we have 86 case of death (outcome 1) against 58 non-death (outcome 0). Therefore, we have an imbalanced dataset. It means we can always predict death and we would be correct 60% of the time. To solve this issue, when available, we used class-weight parameter of the classifiers to give a higher weight to class 0. We tried various weights and found that balanced option proposed by the classifier performs the best. The 'balanced' mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as :

$$\frac{n\_samples}{n\_classes * np.bincount(y)} = \frac{144}{2 * \begin{bmatrix} 86 & 58 \end{bmatrix}}$$

Said in words, balanced mode replicates the smaller class until there is as many samples as in the larger one, but in an implicit way.

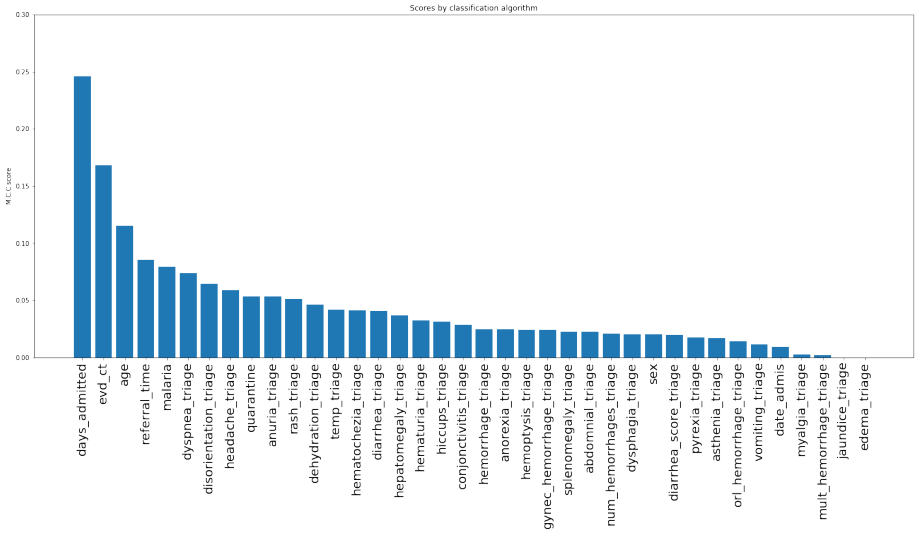
KNN and NN classifiers don't have this option available. In all other classifiers, using this option always improved the score and therefore we always used it for our classifiers.

## 3 Results

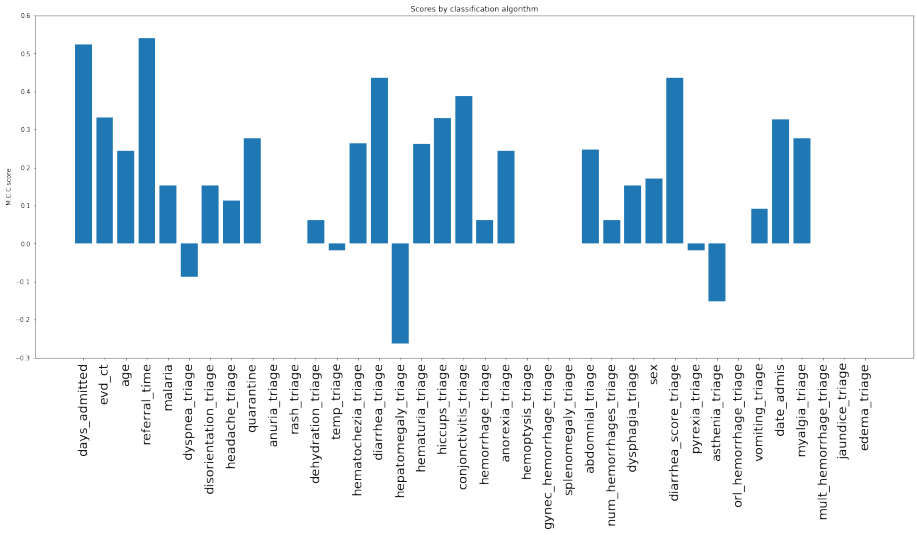
### 3.1 Feature Importance

We tried various techniques for feature selection like random forest, RFE (Recursive Feature Elimination), Logistic regression coefficients, Randomized Lasso and decision tree graphs. To illustrate an example for the feature selection, here is the plot of the coefficients of all features obtained from the Logistic Regression model. Before fitting the model, we standardized the data so the representation is more meaningful. The second plot shows the MCC score when a single feature

is used for Logistic Regression. We can observe that the the plots don't have the same trend. While some features, amongst all others, are important for classification, this is not anymore the case when they are taken individually.It would be interesting to observe the behaviour in other scenarios like classification on all feature except one specific feature, etc. But since we didn't obtained good results with these methods (See Result section), we didn't explore it further.




**(a)** Coefficients of Logistic Regression




**(b)** M.C.C of of LR classifier on a single feature

**Figure 1:** Feature importance

## 3.2 Population Selection

For the population selection part, as we generated the data randomly for the locations and dates, any plot or result will be of little help. We just implemented it as proof of concept. In the presence of real data, they can be studied from various perspectives and can help health mangers to work accordingly. For example, it would be of interest to do a feature ranking study as we have done in our project to compare the most relevent symptoms for different locations or periods of time.
In order to avoid the overfitting caused by adding the binary features (added for categorical location feature) for population selection, once the filtering/selection was done, we removed all features (binary features created for location, the categorical location feature and entry-date and end-date features) related to location or dates.

## 3.3 Standardization

As mentioned before, normalizing the data performed poorly on classification compared to non-normalized data. Here are the results:

| - | LR | KNN | SVM | RF | NN |
|---|---|---|---|---|---|
| Standardized Data | 0.619 | 0.529 | 0.635 | 0.699 | 0.527 |
| Non-normalized data | **0.758** | 0.715 | 0.711 | 0.699 | 0.750 |

## 3.4 Imputation

Once we had obtained our best models, we ran the dataset on all of them : once with missing values removed, then by mean imputation and finally by KNN imputation. Below are the results.

| - | LR(Classic) | LR(Ridge) | KNN | SVM | RF | NN |
|---|---|---|---|---|---|---|
| Mean Imputation | 0.757 | 0.766 | 0.715 | 0.709 | 0.706 | 0.750 |
| KNN Imputation | 0.757 | 0.766 | 0.712 | 0.703 | 0.707 | 0.749 |
| Most frequent Imputation | 0.762 | **0.767** | 0.715 | 0.711 | 0.707 | 0.750 |

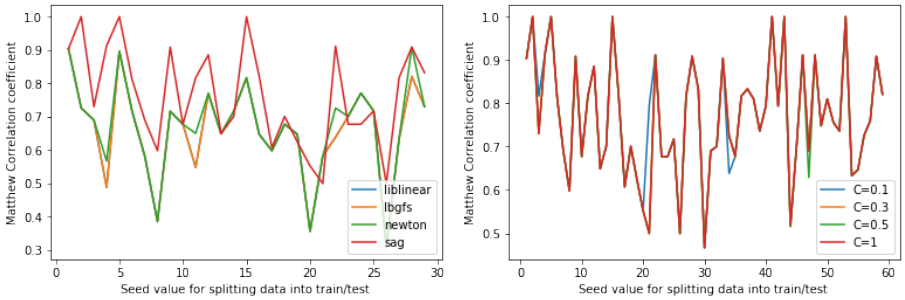
The imputation techniques we have used give the same results but the Imputation by the most frequent element gave slightly better result so that's what we chose.

## 3.5 Logistic Regression

### 3.5.1 Classical Logistic Regression

For Logistic regression we analyzed following parameters: C (inverse of regularization strength), number of iterations, type of solver and the Class-weight : to balance the two classes , Penalty : L1 or L2

At first, We ran several simulations and we observed that L2 regularization performs often slightly better than L1.Then we tried also different optimization solver algorithms (default liblinear, newton, lbfgs and Stochastic Average Gradient(sag) ). Sag largely outperformed all other solvers. The left plot below shows the performance of different solvers over different random seeds used to split the data in train and test using train_test_split from Sklearn. Note that liblinear and lbgfs give almost the same results here and that's why the blue line can not be seen on the graph.
Once the solver was set to sag, other parameters (except class-weight, see below) of the LR classifier nearly didn't impact the performance. Below at right, we can see that various regularization values all perform the same. This is not the case when other solvers are used.




**(a)** Performance of different optimization solvers for different data splitting seeds **(b)** Performance of different regularization values for different data splitting seeds

### 3.5.2 Regularized Logistic Regression

We used Ridge Classifier to be able to penalize heavy weights. We optimized the regularization term alpha in the same way as above.

Indeed, the optimized Ridge Classifer performed always slightly better than the optimized classical logistic regression (0.767 vs 0.762 M.C.C). Not only Ridge classifier had a better score but it's run a lot faster (on average, 1ms against 7ms for classical LR) and it has a single parameter (alpha) to optimize. Therefore we used it for further comparisons.

From now on, by Logistic Regression we mean the penalized version (Ridge classifier).

### 3.5.3 LR performance

For illustration purposes, here is the ROC-AUC of the optimized Logistic Regression on the test set for a random split of the data. We used 15% of data samples (N=144) for test set, which corresponds to 22 patients.

```
Confusion Matrix
 [[ 9  1]
 [ 2 10]]
Classification report
             precision    recall  f1-score   support

          0       0.82      0.90      0.86        10
          1       0.91      0.83      0.87        12

avg / total       0.87      0.86      0.86        22

Matthew Correlation coefficient 0.73029674334
```

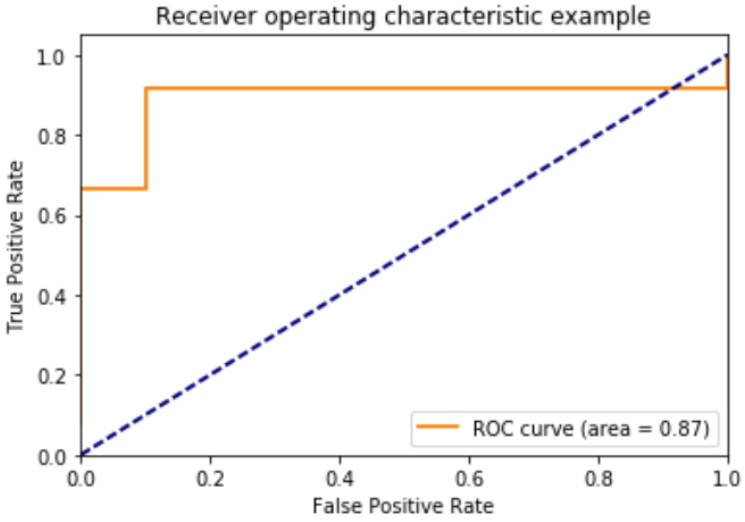



**Figure 3:** Plot of ROC-AUC

### 3.6 KNN

As the number of dimensions of our dataset is small, we thought KNN can be a good algorithm for our classification problem.For KNN, the main parameter of interest is the number of neighbours. We averaged the result of 10000 iterations with different data splitting seeds. On each iteration we computed 10 different models, each with a different number of neighbours. We can see the result below. We observe that k=8 gives the best result.

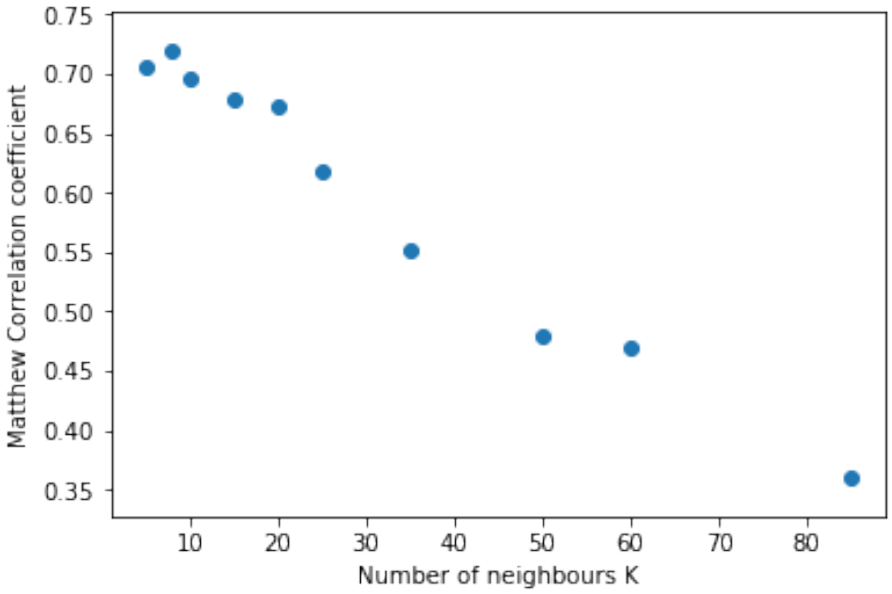



**Figure 4:** Average score of 10'000 resampling

Next we checked how weighted model (the closer the sample, the higher it's weight) performs compare to uniform (empirical average) model and we observed that the uniform model is better (0.718 vs 0.706 M.C.C). Finally, we tested different metrics (Minkowski,Manhattan,Chebyshev) and the default Minkowski performed, on average, significantly better (0.718, 0.653, 0.636 M.C.C, respectively)

### 3.7 SVM

We ran train_test_split function 10'000 times with different seeds (resampling technique) and for each data permutation, we ran multiple SVM models, comparing different values of the same parameter.
The best model is polynomial of degree 1. Polynomial kernel with degree 1 is different than linear kernel Polynomial kernel has a Kernel coefficient gamma which controls the trade-off between error due to bias and variance in the model. To avoid over-fitting we should prefer small values of gamma. Other parameters are left as default.

| - | Polynomial(degree 1) | Linear | Gaussian (RBF) |
|---|---|---|---|
| M.C.C. | 0.704 | 0.64 | 0.699 |

Below are the results, averaged over 10'000 runs, for the polynomial kernel, we clearly see that less is more.

| - | degree 1 | degree 3 | degree 5 | degree 8 |
|---|---|---|---|---|
| M.C.C. | 0.704 | 0.625 | 0.646 | 0.561 |

Next we wanted to find the gamma value giving best results. So we run the above setup (10'000 iteration with different seeds) on polynomial kernel of degree 1.

| - | 1/number of features | 1/10 | 1/90 | 1/1000 |
|---|---|---|---|---|
| M.C.C. | 0.704 | 0.694 | 0.708 | 0.696 |

We also had to optimize the penalty parameter C of the error term was left. It's the cost of misclassification. Large C makes the cost of misclassification high ('hard margin") leading to overfitting. We computed it with the best parameters so far from above. It turn out the default value of C=1 gives the best score.

| - | 0.01 | 0.1 | 1 | 10 |
|---|---|---|---|---|
| M.C.C. | 0.350 | 0.697 | 0.708 | 0.693 |

Lastly, we ran our so-far optimized model another time with various class_weights and again, as in LR, the balanced mode gave the best score.

### 3.8 Random Forest

For the random forest we looked at the following parameters: max_features (The number of features to consider when looking for the best split), max_depth (The maximum depth of the tree) and n_estimators (number of trees in the forest).

Random forest, as it's name indicates, has a random component to it. It corresponds to the features and samples that are selected for each tree. This makes evaluation this model more difficult. Now there are two randomness, one (as for other models) that comes from randomly splitting data into train/test sets (train_test_split function) and the second one for the construction of the forest. In order to avoid bias, we ran the R.F (for hyper-parameter optimization) models in two embedded loops, once varying the seed for train_test_split function and then varying the seed for the R.F models (same for all). We evaluate different models at the end by taking the average of the average for each model.

For max_features, we first tried log2 and sqrt options but as our number of feature (38 features) is already small, these options performed poorly compared to higher values. We then tried with higher values and below are the results.

| - | 10 | 20 | 25 | 28 | 32 |
|---|---|---|---|---|---|
| M.C.C. | 0.719 | 0.721 | 0.728 | 0.724 | 0.720 |

Based on this, we continued tunning other parameters with max_feature = 25. For max_depth, we ran our setup with different models corresponding to max_depth values ranging from 10 to 90 (increments of 20) and all the models gave the same score (0.727). We changed the n_estimators from 40 to 100 and again all models performed the same so we decided to ignore this parameter for the final model.

Next, for the number of the trees in the forest, we ran also the setup described above with values from 10 to 250. For random forests, generally more trees give better accuracy. However, more trees also mean more computational cost and after a certain number of trees, the improvement is negligible, as we can also see from out plot. An article from Oshiro et al. [4] pointed out that, based on their test with 29 data sets, after 128 of trees there is no significant improvement.

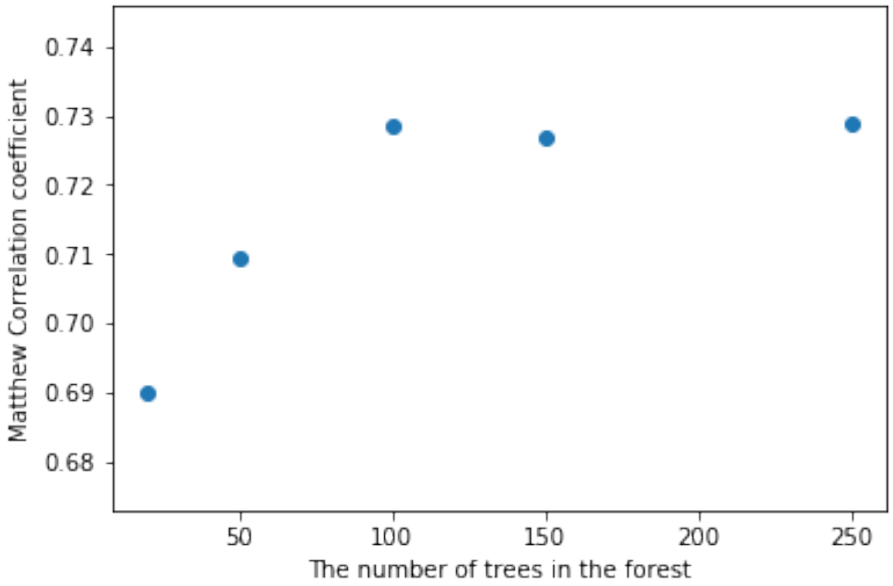



**Figure 5:** Performance of various hidden layers

Finally, we checked again for class-weight parameter and as for other classification algorithms used, the 'balanced' mode gave a better score.

### 3.9 Neural Networks (NN)

We used MLPClassifier from Scikit-learn to implement our NN. We have a binary classification so our output layer has just 1 neuron. For the input layer, our dataset has 38 features. This is the number of neurons for the input layer. For the number of nodes for hidden layer(s), having too many neurons leads to over-fitting the training data and in other hand too few neurons will cause high training error due to under-fitting.There exist some empirically-derived rules-of-thumb. One that is often used states that the optimal size of the hidden layer is usually between the size of the input and size of the output layers. Another, more precise, rule-of-thumb suggests that the size of hidden layers should be around two third of the input layer size. We ran the neural network with 0 (not plotted) to 4 hidden layers and varying the number of nodes from 1 to

40 (always keeping same number of nodes for all hidden layers). Here are the results:

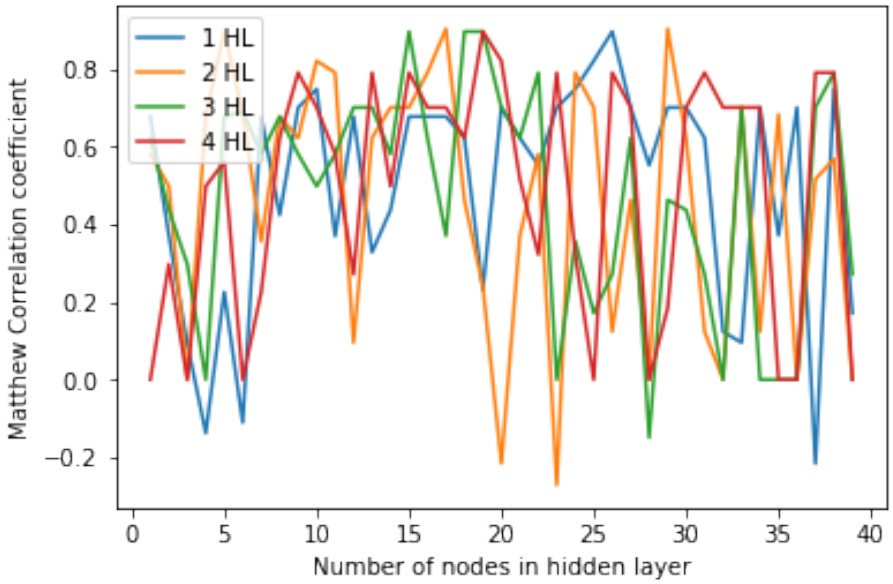



**Figure 6:** Performance of various hidden layers

We observe that the results are very unstable and there is no specific trend. Therefore it's impossible to conclude anything on the the optimal number of hidden layers and nodes. The apparent lack of stability and predictability is a drawback for this model and might be a consideration (However, as detailed in Result section, the results of the chosen model were consistent and stable through various simulations). From various runs (different initial weights), we observed that one hidden layer with around 26 nodes gives usually amongst the best scores and other configurations don't significantly improve the performance. Furthermore as this configuration just uses one hidden layer it's computationally lighter so that's what we chose for our final NN model.

We also tried different activation functions (the rectified linear unit function, the hyperbolic tan function and the logistic sigmoid function). The same setup as above was run several times and we observed that the tanh activation function usually performs better than others (only slightly better than logistic). In addition, in order to avoid over-fitting, we also checked various regularization terms (0.001,0,01,0.1 and 1) through various runs and we found that, on average, there was no significant difference among them.

### 3.10 Ensemble Learning

Finally, and after we obtained the optimized model from each classification algorithm, we decided to combine all of them to create other models. We created 10 new models in 3 categories:

1. 4 Bagging models using the prediction of our 5 optimized models (Majority Voting)
2. 4 Bagging models using the estimated probability of the outcomes for our 5 optimized models (Averaging probabilities)
3. 2 'Blending' models, applying a Logistic regression on the prediction and estimated probability of the 5 optimized models

For the two first categories, for each patient, we took the prediction (1 or 0) and the predicted probability, respectively, of our 5 models and decided 1 or 0 based on the 'weighted' majority of the models and averaged probability for 2nd category. In the simple case and without loss of generality, if 3 models predicted 1 and two predicted 0, the bagging model (category 1) would 'predict' outcome 1 for this patient. Furthermore, we tried different weightings for our bagging model (same for both bagging categories). We choose to do so because some classification algorithms (LR and NN) performed always better than other models and an equal weight doesn't take this into account.
We created four bagging models. Weightings are as follows:

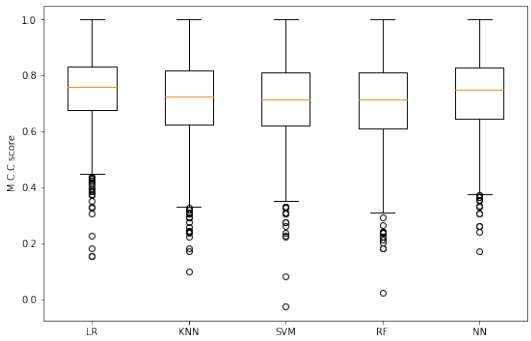
1. Equal weights (1/5 each model)
2. 2/5 each: LG and NN; 1/5 KNN
3. 2/7 each : LG and NN; 1/7 each KNN, SVM and RF
4. 3/7 LG; 2/7 NN; 1/7 each KNN and SVM

For the baggigng models, we also used VotingClassifier by Scikit-learn with the same models as above and we obtained the exact same results.

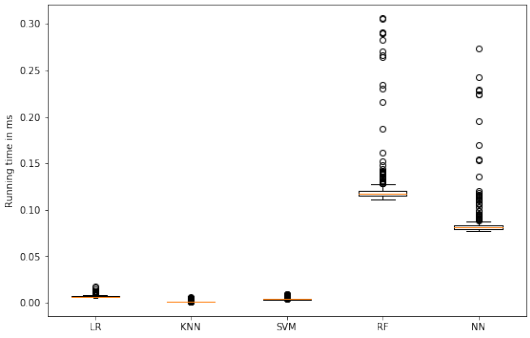
For the two blended categories, we trained the two Logistic Regressions on the test set of our 5 optimized algorithms and then evaluate the LRs on the validation set.

## 4 Discussion

Let's first box-plot the result of our classifiers.




**(a)** M.C.C score of each classification algorithm




**(b)** Running time of each algorithm

First of all, for the 5 optimized models, we observe that in terms of classification performance (as represented by MCC), there is not a huge difference among different models. However, Logistic Regression and Neural Networks classifiers consistently perform better than the three other models, with the former being usually slightly better. On the other hand, in terms of running time Random Forest and Neural Networks are computationally heavy and take a considerably longer time to finish. More generally, our dataset being small the running time is in all cases relatively small and it's not the first priority on which to choose the best model but it can become important as the dataset becomes larger. However, for the sake of humanity, we hope this will not happen.
The Random Forest model returned the lowest performance both in terms of classification performance and running time, thus, it is relatively poorly fitted to the data.

The KNN model has by far the lowest running time and its score is relatively good. Plus, it requires a single parameter, i.e number of neighbours. So it's a fast and easy-to-setup classifier which can function as a benchmark. We think the main reason KNN didn't obtain a better score is that KNN can suffer from skewed class distributions. For example, if a certain class is very frequent in the training set, it will tend to dominate the majority voting of the new example.

Overall, Logistic Regression is the winner, it's score is slightly better than NN but it's running time is significatively lower. Also it requires less parametrization and optimization than NN. Furthermore, from our optimization of the NN classifier, we obtained a model with a single hidden layer and no bias term so the NN is relatively close to a Logistic Regression model as the scores attest.

We think the main reason the classifiers perform quite similarly is that first, we have spent extensive time optimizing each of the models to best tailor them for our problem and, second and more importantly the dataset is relatively 'well-behaved' and doesn't have particularities like non-linearity or other characteristics. We mean the type of characteristics which create a bigger difference on the choice of the classification algorithm.

Now let's have a look at all models including the ensemble models. This is the average of 10'000 seeds from train_test_split:

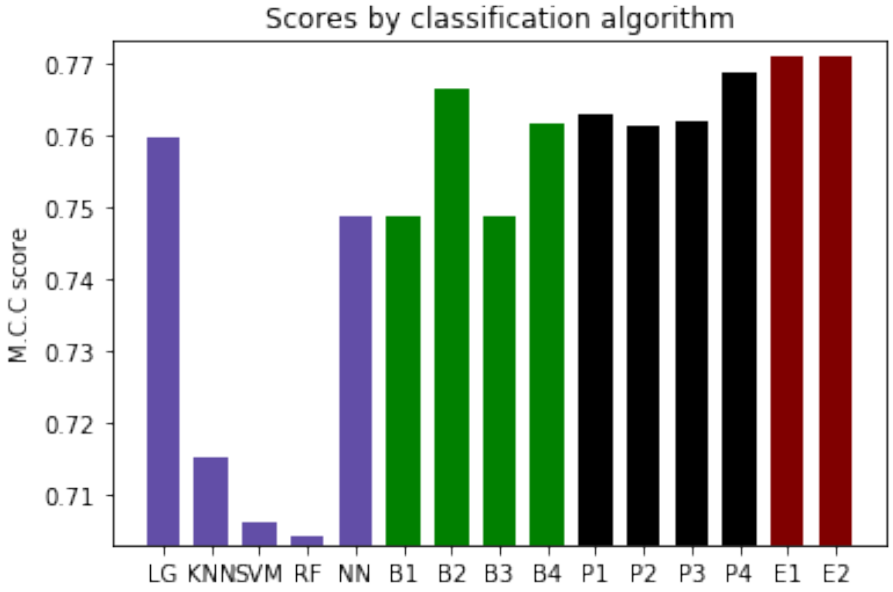



**Figure 8:** Classification algorithms and bagging models

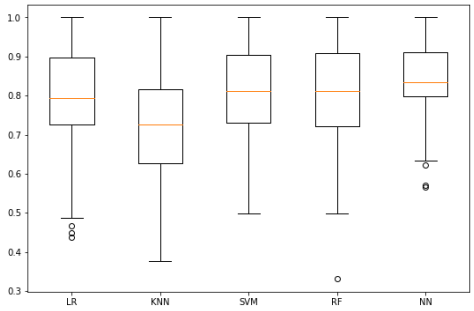
We are happy to observe that more than half of the bagging models performed better than the best classification algorithm (LR) alone. It's interesting to notice that the remaining bagging models (in particular the simple bagging, i.e same weights for all models) performed worse than LR. Therefore, it's very important to choose the appropriate weights to optimize the score. The best bagging model for category 1 was the one which only chose the best classification algorithms to begin with and weighted them optimally (Bagging model 2). Interestingly, for category 2, the best model was not the equivalent model (in weighting distribution) of category 1. Model 4 had the best performance for category 2.
A point to mention is that ensemble methods reduce the interpretability of the models. The output aside, It becomes hard to explain the result. Category 2 is of greater help because we have the probabilities and we can use this information to give a scaled score for the Android application.
One should note that although the Ensemble learning method performs best, it's computationally heavy as it needs the result of all classifiers in order to work. This might be a consideration for future.

Now below are the results on after-triage model (left) and the model with chosen features (right).

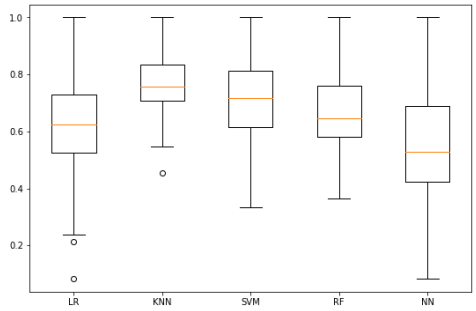
As expected, the after-triage model performs better than triage model. This is because the symptoms have now become more evident and it helps to predict

the outcome.

We can also observe that the model with chosen features is not very performant.Among all techniques used, none of them performed well comparing to the complete model.




**(a)** Performance of classification algorithm on after triage




**(b)** Classification only using selected feature give by mentionned techniques (RFE, RF, etc.)

Finally, compared to the Harvard models, we obtain a better result on our dataset. Technically, this is logical since we have trained our models on (part) our dataset and optimized it consequently. In order to compare objectively our model with the Harvard model, we should evaluate them on another independent dataset.
There are also clinical reasons that explain why our model outperforms the Harvard model: Harvard's dataset was a heterogeneous set of confounding/modifying factors that dilute the predictive capacity of individual features whereas our dataset was homogeneous and all data was coming from the same ETC. Also, Harvard's dataset was gathered with a wide temporal span, during which time the clinical response was erratic and the virus evolved to favor a less virulent strain whereas our dataset was collected right at the end of the epidemic where the referral time of patients was pretty low.

# 5 Limitations

As mentioned earlier, we have an imbalance dataset. Although we did our best to balance things out (by using the class-weight parameter of classifiers), the classification error is not yet perfectly balanced between two classes. Namely, as expected, we classify better the death cases. To illustrate this, we ran 10000 runs (with various data splitting seeds) on the Logistic Regression model each time recording the F1-Score for each class: We obtain an average F1-score of 0.851 for class 0 and 0.899 for class 1. So we more often misclassify class 0.
This is acceptable. We prefer to have false positives rather than false negatives but it's something to be aware of.

Additionally, we recommend using the population selection tool(when needed

information will be available) when using the app in a new 'environment' (different country, different period).

# 6 Conclusion

This study investigates some of the most widely used machine learning classification algorithms in order to classify the outcome (death or no) of EVD+ patients. Our final model has a high prediction precision and can aid objective clinical prioritization.

Based on our results and considering the need of a model which is understandable for the medical team and does not only operate as a black-box giving a high classification score, we would recommend the use of the Regularized Logistic regression (RLR) model for this classification task. Taken individually (without bagging models), RLR always obtain the best classification score, it performs only slightly less better than the best bagging model, it's running time is quite low, it's a simple model which is easy to setup and to understand, it's results are stable and we can obtain nice insights about the model by analyzing it via feature importance plots (as in Figure 1).

# 7 Perspectives

External validation of our model is needed in order to further improve its accuracy and relevance. Furthermore, collecting data regarding geographical location and entry date of patients in the ETC can, first, improve the model's accuracy and relevance, and second, contribute to put our concept of population selection into practice. This is of particular interest for medical team. Finally, integrating the machine learning model into the Android app and interaction of the app with a centralized server (for updating data about patients and updating the ML models in the app) remains to be done.

# 8 Acknowledgments

I would like to thank my supervisors Dr. Hartley and Dr Jaggi for their precious help and for enabling me to work on such an interesting project.

# References


[1] Outbreaks Chronology: Ebola Virus Disease
`https://www.cdc.gov/vhf/ebola/outbreaks/history/chronology.html`

[2] Opening remarks at the Ebola vaccines for Guinea and the world event
`http://www.who.int/dg/speeches/2017/ebola-vaccines-guinea/en/`

[3] Hartley M-A, Young A, Tran A, Okoni-Williams H-H, Suma M, Mancuso B, et al. *Predicting Ebola Infection: A Malaria-Sensitive Triage Score for Ebola Virus Disease* PLoS Neglected Tropical Disease. 2017.

[4] Thais Mayumi Oshiro, Pedro Santoro Perez, and Jose Augusto Baranauskas. *How Many Trees in a Random Forest?* University of Sao Paulo. 2012.