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 healthcare 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) andin 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 our app on their Android smartphones. Our solution is highly dynamic and self-adapts to 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.

We also had access to another dataset, from data collected in a district hospital in Sierra Leone, Kenema Government Hospital (KGH). The initial idea was to use this dataset for our external validation. Finally, because of the great 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 evd_ct (inversely proportional to viral load in patient's body) feature contains some missing values. This concerns the EVD- cases. For prognosis, all these cases were removed from dataset, bringing down our sample size from 575 to 144 patients.

Additionally, the three following features contain missing values : malaria infection,referral_time (i.e. the time taken for the patient to present at an Ebola treatment centre since their first symptom) and quarantine.
The number of patients with missing values was relatively small (n= 26). Nevertheless, we once ran our models with the missing values removed and another time imputed. We found that imputing results gives a model that performs

slightly better on the validation set. We think the reason is that by imputing missing values, we have more data (26 patients correspond to 20% of our dataset) to train our model on so we obtain a better model. Moreover, imputation by mean and KNN imputation were tested and the results on classification were very similar. This can be explained by the low proportion of missing values, and the fact that the concerned features were not among the most prevalent for the classification. 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), and the second one is the same set of information gathered after some days (determined by 'days_admitted') spent in ETC. 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.

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 helps to adjust the treatment with the specificities of each region. To integrate this ability to our model, we artificially added a location categorical feature and an entry date feature to the dataset:
For the location feature, using Numpy's random.choice function, we attributed a location (from an array of 5 locations) to each patient. Then, to be able to use this information in our prediction models, we used 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 entry-date feature, we used Pandas Timestamp to artificially create dates between two given 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 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 algorithms: Logistic Regression, KNN, SVM, Random Forest and Neural Networks. For each of them, we have used grid-search using 10-fold cross-validations to fine-tune the hyper-parameters and obtain the best model (based on Matthew Correlation Coefficient).

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

2.6 Metric

For binary classification, many metrics exist to evaluate the performance of the classifier (Accuracy, Balanced Error Rate (BER), F-1 measure, Matthew 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'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. Note that this is very near the average result of our classifiers.

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

2.7 Sampling

In general, cross-validation is an excellent way to learn models with low bias and avoid over-fitting but it's draw back 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 precede in two steps.

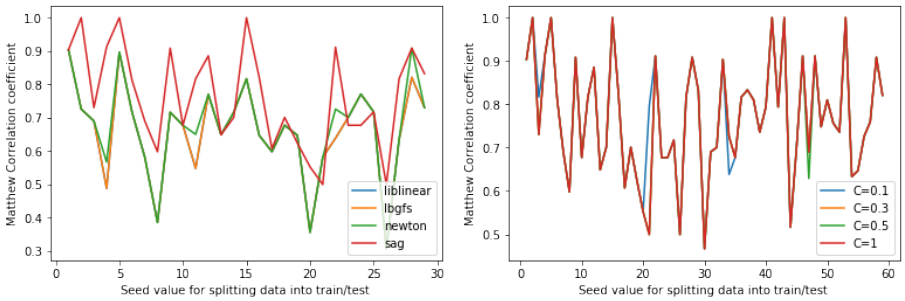
First we did Hold-out validation. Hold-out is also biased if done once therefore we did 5000 Hold-outs, independently, given by different seeds of train_test_split function. We used 15% as test size. In the second step, we used GridSearchCV with 10-folds to confirm the results from first part.

3 Results

3.1 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 : Lasso (L1) or Ridge (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

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, we used class-weight parameter of the classifier to give a higher weight to class 0. We tried various weights and the 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 (n_samples / (n_classes * np.bincount(y))). 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.

3.2 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 5000 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.

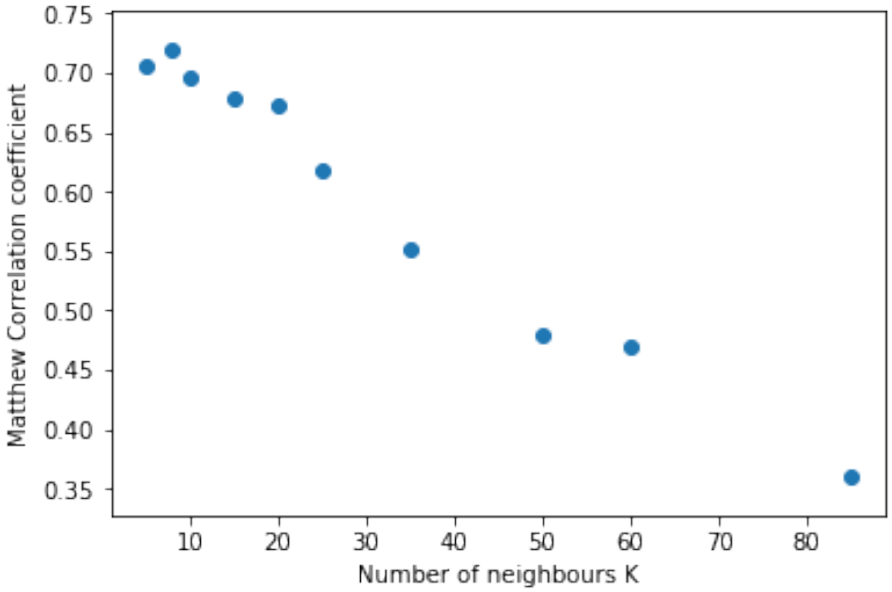


Figure 2: Average score of 5000 resampling

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)

3.3 SVM

We ran train_test_split function 5000 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 5000 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 (5000 iteration with diffrent 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.

3.4 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 foerst, we ran also the setup described above wit 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.

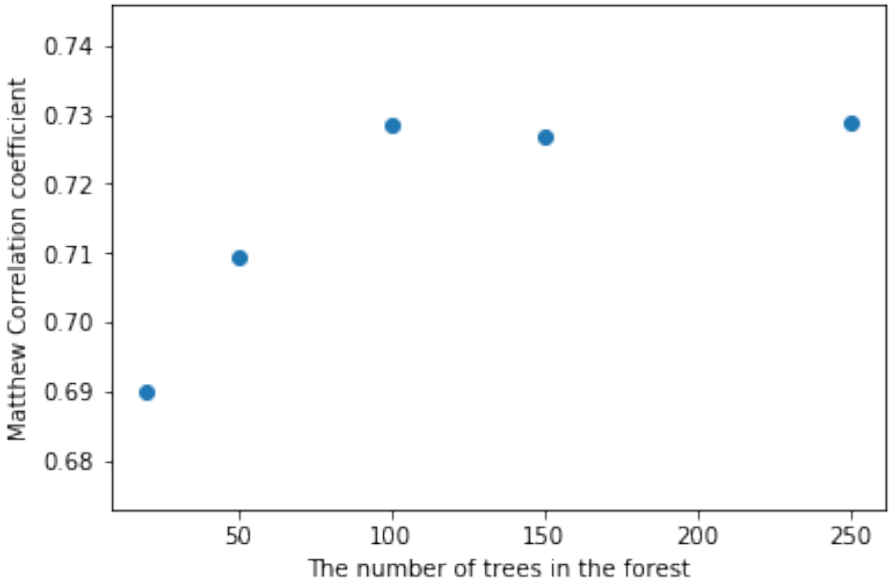


Figure 3: Performance of various hidden layers

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

3.5 Neural Networks (NN)

We used Scikit MLPClassifier 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, always keeping same number of nodes for each hidden layer, and varying from 1 to 40 nodes. Here are the results:

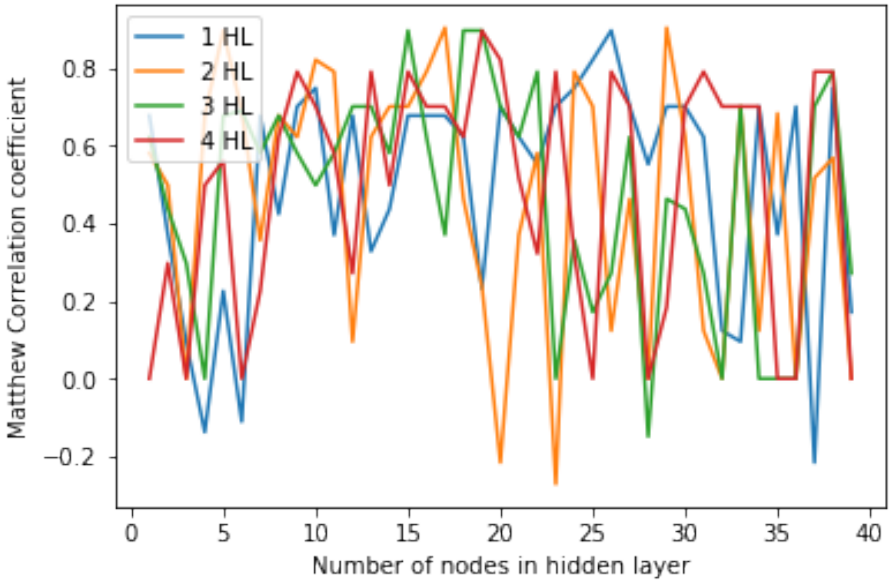


Figure 4: Performance of various hidden layers

From various runs (different initial weights), we observed that one hidden layer with around 26 nodes gives the usually the best score and other configurations don't significantly improve the performance. Another reason to use 1 hidden layer is that it's computationally lighter.
It's also a common practice to standardize data before the Call to NN classifier. We did so using StandardScaler from scikit-learn by running on the training set and then applying on the test set. We ran the same configuration as above for the Standardized data: On avergae, the performance got worse by 22% so we decided not to use standardization. We think the main reason is that the ma-

jority of features being binary standardization can not help the Neural Network. We also tried different activation functions (the rectified linear unit function, the hyperbolic tan function and the logistic sigmoid function). We ran several time the same setup as above 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.

3.6 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	KNN	SVM	RF	NN
Missing values Removed	0.737	0.687	0.708	0.692	0.750
Mean Imputation	0.757	0.715	0.709	0.698	0.750
KNN Imputation	0.757	0.712	0.703	0.699	0.749
Most frequent Imputation	**0.758**	0.715	0.711	0.699	0.750

We see that imputation gives better results than removing missing values but the imputation techniques we have used give the same result but the Imputation by the most frequent element gave slightly better result so that's what we chose.

3.7 Ensemble Selection

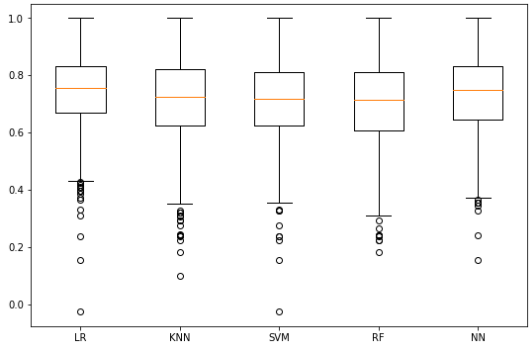
Finally, and after we obtained the optimized model from each classification algorithm, we decided to combine all of them to create other models. We used bagging model with majority voting. In other words, For each patient, we took the prediction (1 or 0) for our 5 models and decided 1 or 0 based on the 'weighted' majority of the models. In the simple case and without loss of generality, if 3 models predicted 1 and two predicted 0, the bagging model would 'predict' outcome 1 for this patient. Furthermore, we tried different weightings for our bagging model. 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

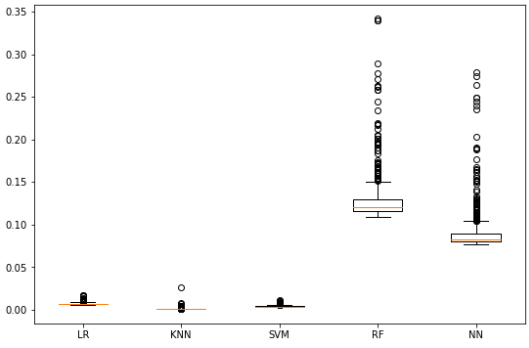
4 Discussion

(Comment: This section is not yet completed)

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, we observe that in term of accuracy (M.C.C), there is not a huge difference among different models. However, Logistic Regression and Neural Networks classifier consistently perform better than the three other models, with the former being usually slightly better. On the other hand, in term of execution time Random Forest and Neural Networks are computation heavy and take 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 preoccupation to choose the best model but it can become as the dataset becomes larger. However, for the sake of humanity, we hope this will not happen.
The Random Forest model having the worse score both in term of accuracy and running time, it's probably not a good model for this dataset.

The KNN model has by far the lowest running time and it's 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 help as 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 char-

acteristics. 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 bagging models. This is the average of 10'000 seeds from train_test_split:

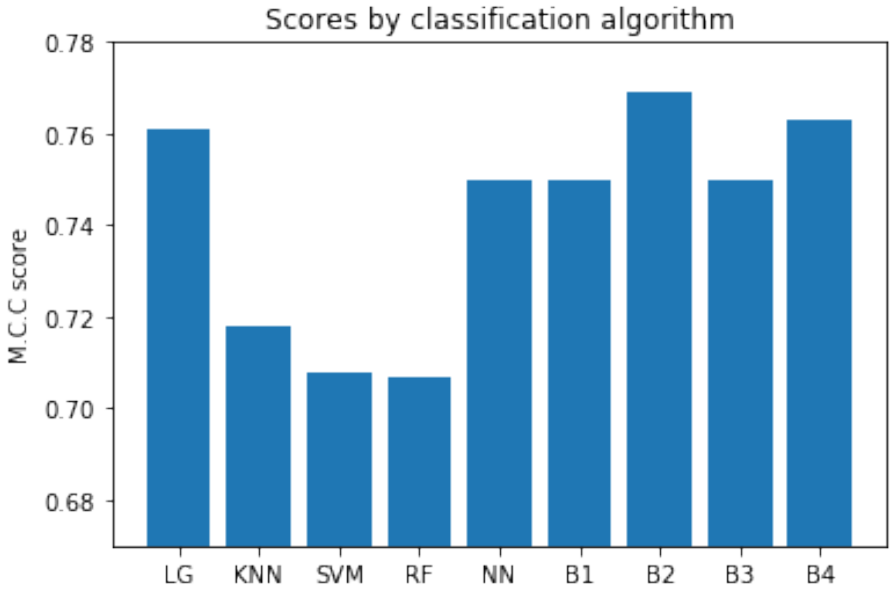


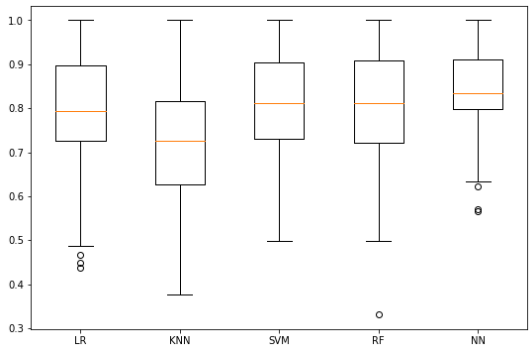
Figure 6: Classification algorithms and bagging models

We are happy to observe that two of the bagging models performed better than the best classification algorithm (LR) alone. It's interesting to notice that the two other bagging models (in particular the simple bagging, i.e bagging model 1) performed worse than LR. Therefore, it's very important to choose the right weights to optimize the score. The best bagging model (and the overall best) was the one who only choose the best classification algorithms to begin with and weighted them in a good way (Bagging model 2).

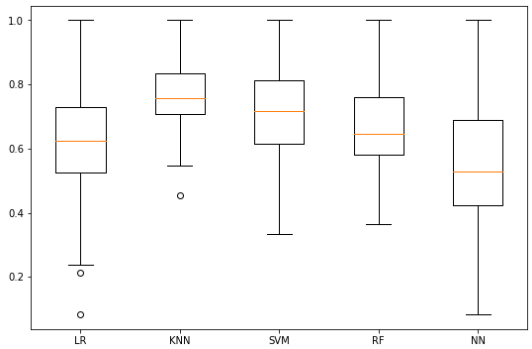
Now Below are the results on after-triage model (left) and the model with chosen features (right).

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. We tried various techniques for feature selection like random forest, RFE (Recursive Feature Elimination), Randomized Lasso and decision tree graphs non 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.

For illustration purposes, here is the ROC-AUC of the optimized Logistic Regression for a random split of the data :

```
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
```

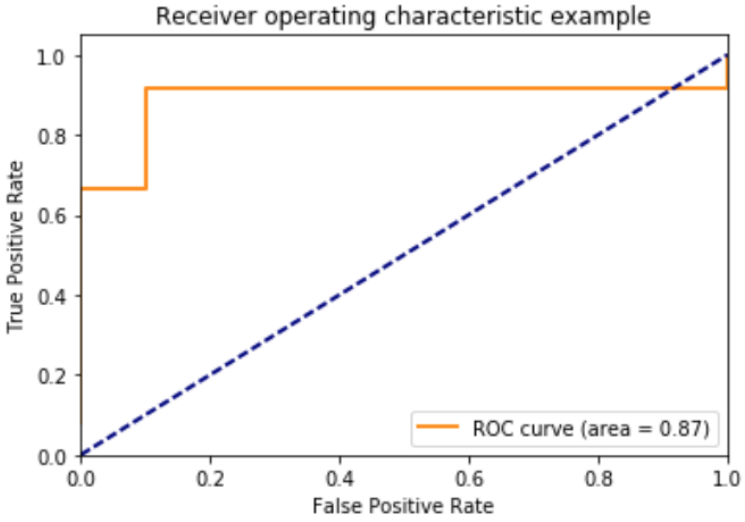


Figure 8: Plot of ROC-AUC

(to add: something about a fictiv model with the population selection mode)

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.