Treat First What Kills First: Predicting Disease Severity with DDXPLUS

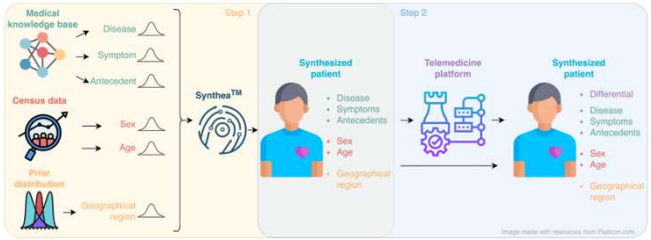
Leandra Hogrefe
hogrefe.16@osu.edu
Department of Physics, The Ohio State University

Motivation

In the medical field, the accurate and timely prediction of diseases is essential to ensure patients have the best possible chances of getting healthy. At the same time, medical professionals are needed more than ever, and many medical institutions are short-staffed increasing the workload of each doctor and nurse. With machine learning on the rise, there have been many efforts to support medical professionals by producing differential diagnoses or predicting/ruling out severe diseases.
This project makes use of the DDXPLUS data set to explore exactly this opportunity. How well can a machine learning model predict a disease's severity? The data set comes with a wide variety of evidence assigned to each virtual patient. This evidence is then used in this project as the input for models to separate the patients into different severity classes.

Dataset

The patients in the dataset are artificially generated to ensure individual anonymity. The graphic shows how the "patients" are generated. The dataset is imbalanced in pathologies as well as disease severity. The differential diagnosis in the patient tables lists possible pathologies and possibilities that the respective patient has this disease. These results are generated by "a real-world telemedicine platform". The pathology then lists the name of the actually diagnosed disease. In most cases, these two agree showing that the model used for this prediction did a good job.

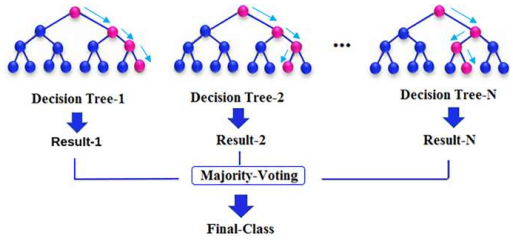


There are 223 symptoms and antecedents (evidence) in the DDXPLUS data set. Out of those 208 are binary (shows evidence yes/no) and 15 can take different values (e. g. pain on a scale from 0-10, classifying the type of pain, etc.). To use these as features, we one-hot encode the evidence. This process leads to a different shape for every patient table as soon as one piece of evidence is missing added. Too keep a consistent shape, we will only use the train file from DDXPlus because one million patients leaves us with enough to train even when splitting it into train and test sets. The binary evidence was already normalized from 0 to 1 but the numerical evidence initially went from -1 to 10. Here, I adjusted the range from -1 to 1.

Methods

A Random Forest is a collection of many Decision Trees which separate the data by splitting it based on a criterion. Each layer achieves a better separation. However, these trees adapt very closely to the data resulting in overfitting which the Random Forest regressor counteracts by combining many trees, 50 in our case. To train the different trees a method named "Bootstrapping" is first applied to randomize the data every tree sees, so that the different trees capture different structures in the data. To further increase the differences between the trees, the trees "grow" based on different features. The graphic in the "Results" section shows how the results of the trees are combined at the end by voting for a majority result.
As the name suggests, CNNs take advantage of a process called "convolution". In this process a kernel moves across the input multiplying the features by the kernel resulting in a convolved feature. The kernel can be sensitive to different structures in the data which is how the network recognizes patterns. Additionally, the model contains max pooling layers which increase translational invariance, so it saves the structure but not the exact position. At the end of the model, there are a global average pooling layer and a dropout layer both of which effectively reduce overfitting, one by averaging, one by setting as fraction of values to 0. The dropout layer can also lead to the validation accuracy being higher than for the training because some training data is dropped. The structure is shown in the in the results section.

Results



The Random Forest metrics were very similar for train and test data. I also ran a k-fold validation with 5 folds to get another metric but kept the number of folds low to keep the model simple. Each time the precision was about 92% and the fitting process took about 10min.
For the CNN whose architecture is shown below (layer in the red box was removed) the fitting took much longer with about 1h and 47min but the validation accuracy got up all the way to 0.997 after 14 epochs (the model got stopped due to early stopping I implemented in the model.

```
model_m = keras.models.Sequential()
#
# Our first layer gets the input from our samples
model_m.add(keras.layers.Conv1D(100, 10,
                    activation='relu', input_shape=(  462,1))
#
# Another convolutional layer
model_m.add(keras.layers.Conv1D(100, 10, activation='relu'))
#
# Max pooling
model_m.add(keras.layers.MaxPooling1D(3))
# Two more convolutional layers
model_m.add(keras.layers.Conv1D(80, 10, activation='relu'))
model_m.add(keras.layers.Conv1D(80, 10, activation='relu'))
#
# Global average pooling use this instead of "Flatten"
model_m.add(keras.layers.GlobalAveragePooling1D())
model_m.add(keras.layers.Dropout(0.5))
model_m.add(keras.layers.Dense(num_classes, activation='softmax'))
```

Discussion

	Pred:1	Pred:2	Pred:3	Pred:4	Pred:5
True:1	13113	568	2358	120	0
True:2	0	34723	2314	4627	0
True:3	0	249	59136	2593	0
True:4	0	0	1925	61661	10
True:5	0	0	3	3951	17770

The CNN required a lot of resources to achieve a high accuracy (see confusion matrix below, but the Random Forest got some very far off classifications as seen in the confusion matrix above. Using a simple and faster to train CNN architecture led to results that were significantly worse. The Random Forest performed not quite as good but was significantly quicker to train (10min compared to 1h 47min) and to evaluate test data. Here, we have to not that the performance metrics can be misleading: a precision of 92% does not seem that much lower than 99% but comparing the confusion matrices below show where the misidentification are and that the Random Forest more frequently is far off meaning that the impact on the patient could be much worse.

	Pred:0	Pred:1	Pred:2	Pred:3	Pred:4
True:Most	16142	12	4	1	0
True:More	4	41637	3	20	0
True:Intermediate	0	21	61933	21	3
True:Less	0	28	7	63098	463
True:Least	0	0	3	18	21703

Conclusions and Future Work

The CNN outperformed the Random Forest in identifying disease severity classes. However, it required significantly more resources to do so. In a medical context this raises the question: How long would you be able to wait for a diagnosis to make sure it is as accurate as possible? A goal for a continuing project would be to experiment with models more that can achieve a high precision/recall quickly. To increase performance, there are also some aspects in the contexts of this medical situation that could be reflected better by the data/model structure. Implementing a higher penalization for "far off" classes (True: 1, Predict: 5) could boost performance. Additionally, the current one-hot encoding method causes the fitting to fail if testing data had more or less evidence columns due to the expected input shape. In a real-world application a new symptom might show up and the model still needs to perform. A more flexible model structure or developing a pre-modeling process to deal with these features would be an interesting project to tackle this problem.

References:
[1] Tchango, Arsene F. et al. DDXPlus Dataset \url{https://github.com/mila-iqia/ddxplus}, 2022.
[2] Classifying Sequences. Carmen Canvas: AU24 PHYSICS 5680 - Big Data Analytics. module8_sequences_2024.pdf, 2024.
[3] Random Forest. Carmen Canvas: AU24 PHYSICS 5680 - Big Data Analytics. module4_random_forests_only.pdf, 2024.