IML – Hackathon

Background:

We choose the second task, because we found it more meaningful and clearer. First, we investigated the features provided for each patient, their meaning and interactions. We decided what is more or less relevant, and how we should combine features, complete missing values, and preprocess the data.

For example, each patient had 6 features describing the date and type of the surgical procedures they had. Since not all patients had 3 surgeries, we needed to find a way to combine these features into more meaningful features that include and express the number and type of procedures as a single feature. We decided to convert the surgery dates to average interval, that is to combine them to a feature that contains the average number of days between procedures that the patient had. We also combined the surgeries types into a single numerical value, which expresses the severity of the procedures the patient had (the more advanced procedures and higher number of procedures, the higher the value).

We created different strategies to handle missing values – Completing values using some default value, using the mean of all other samples, or using clustering and completing values from the mean of the matching cluster. We tested our models with each of the strategies, and found that completing different values with 0 achieved the best results.

Part I:

At the start of this section, we decided to create the base line model using a KNN classifier, which uses a OneVsMany strategy to expand it for multilabel predictions. This model trains on each label separately, and predicts each label separately. It then combined the resulting prediction into a binary vector, of which each coordinate represents a different metastases location in the body. That way we are able to predict multiple locations for each patient. The base line model performed relatively poorly, especially relative to the macro average F1 score. Some locations in the body, only had 2-4 entries in the data that had metastases in those locations. Therefore, these labels were harder to predict (less data to train on). In order to handle these dataset constraints, we choose to use AdaBoost algorithm with a Decision Tree base model for out final classifier. We choose AdaBoost, in order to allow the model to further learn on the more infrequent labels on which it makes more mistakes. The final model uses cross validation to learn the AdaBoost hyper-parameters.

The final score we managed to get on out test set (we split the given train data into our own train and test), is 0.99 on the micro F1 average and 0.81 on the macro F1 average.

This model was better at predicting the labels which were more common – locations which appeared in relatively many samples. This can be expected, as the common labels had more data for the model to capture and improve its predictions.

The model is worse at prediction the labels of less common labels – some labels in the data only appeared in a few samples, which didn't give the model a lot of data to train on. The poor performance of the model in these samples was visible in the difference between the

micro and macro F1 average scores – the micro score gives more weight to the more common labels and macro gives equal weight to each label. The high difference between the scores we managed to achieve expresses that our model is useful in the general case (when measuring success on all of the data, regardless of label type), but is less successful on the less common labels (which lowers the score when equal weight is given to each label).

Part II:

At the start of this section, we decided to create a linear regression model as out base line. We created one model without regularization, and one model with L_2 regularization (with a high α value). We created 2 model, each aiming at a different extreme of the underfit – overfit scale, in order to get better intuition about the data, and about the performance of our final model. Both models achieved poor performance, even the model aiming to overfit the training data, which made us conclude that the task is too complicated for a linear regression model (we would expect the overfit model will achieve lower loss while testing it on the data it trained on). In the next step, we tried to enhance out model in order to better catch the complexity of the data. We wanted to combine different models, in the hope that each model could catch some other aspect of the data, and their combination would be able to achieve lower overall loss. We decided to add other strategies to our model, KNN and decision trees, and combine them into a single ensemble (with the previous linear models). We used the clustering of KNN and the partition of the decision tree, to group samples together in a meaningful way, and using the average of each group predict the value of the new test samples. We implemented principles we learned in class, e.g. clustering, in order to find a creative and efficient way to predict the continuous value of the tumor size using the clusters found in the KNN and decision tree models. We noticed an improvement in the models performance after these changes, but we weren't satisfied with the result as a final model. We saw that some of the models performed better and some performed worse, and in order to further improve their ensemble, we decided to add a weight to each model according to its MSE loss on the train data. We started by calculating the weight of each model, by running it through a 5-fold cross validation function. If a model achieved a MSE loss of x, then we defined its weight to be $w = \frac{1}{x}$. Then, we trained each model on the entire train dataset and combined the resulting models and their weights into a weighted ensemble. This final model achieved a MSE loss of 0.81 on the test set (we split the given train data into the train set we trained our models on, and into a test set on which we later tested their performance before submission).