HW 5

Submitted by:

- Itay Waisman - 311177562
- Vlad Keel - 312866775

Wet Part - Explanations

Data Preprocessing

We used a similar schema to what was done in assignments 2 and 3, except the feature selection which was replaced by a static list given to us this time.

This time, we splitted the dataset into 2 parts: train (80% of the dataset) and validation (20% of the dataset).

Looking at each feature's histogram (using `df.hist()`) we saw that most features where normally distributed, some with slight skewness. As expected, the features which were categorical or discrete, looked that way in the histogram.
Another thing we decided to check was missing values for each feature. We used `df.info()` and we saw that most features had ~20% missing features, which we should impute later on.
The last visualization we made was `df.describe()` to see whether we needed to treat outliers, and see the normal characteristics of each features to scale them well. We saw that most features had outliers, and clipping them to 0.25-0.85 percentiles helped later on.

The preprocessing pipeline included:

- **Data Type handling** - we transformed the `SyndromeClass` feature to a onehot embedding, as it is a categorical feature. We debated whether `AgeGroup` should be treated the same way, as it is discrete, but looking at the feature's histogram made us decide that it is almost continuosly normal distributed, thus embedding might make that characteristic be lost.
- **Data scaling** - we used `RobustScaler` from sklearn, which allowed us to scale the normally distributed features
- **Imputation** - we imputed the missing values, which were frequent amongst all features, using 2 methods:
  - `IterativeImputer` from sklearn, which implements a multivariate imputer, and supports numerical features
  - `KNNImputer` from sklearn, which allowed us to impute categorical features such as `SyndromeClass`

The last preperation task we did was to seperate the label column into 3 seperate columns, so we can fit a model for each of them seperately.

Model Comparison

We were requested to compare different models for each of the 3 different target labels, afterwhich we needed to automatically select the best model to use.

Score function

We debated whether we should choose to use the recall and precision scores again as in assignment 3, but we chose to use the accuracy measure this time. This was done with the thought in mind that our model will probably be graded using the accuracy. Although, still, in real life we would probably choose to use recall for 'AtRisk' and 'Spreader' because false negatives have very bad consequences, and percision for 'Virus' to accurately predict the virus.

Models Fitting schema

We used cross validation with `GridSearchCV` from sklearn, in order to select the best parameters for each model and maximize it's prediction ability. The cross validation was done using the accuracy score function as described above as this was our target scoring function.

Models and parameters

The models we decided to compare:

- Support Vector Machine - using the `SVC` from sklearn. The parameters we grid-searched on where: `{ 'kernel': ['rbf'], 'C': [0.1, 1, 10, 100, 1000] }`
- K-Nearest-Neighbours - using the `KNeighborsClassifier` from sklearn. The parameters we grid-searched on where: `{ 'n_neighbors': np.linspace(2, 10, 9, dtype=int) }`
- Logistic Regression - using the `LogisticRegression` from sklearn. The parameters we grid-searched on where: `{ 'max_iter': [1000], 'C':[0.1, 1, 5, 7, 10] }`
- Random Forest Classifier - using the `RandomForestClassifier` from sklearn.ensemble. The parameters we grid-searched on where: `{ 'max_depth': np.floor(np.linspace(5, 100, 20)).astype(int), 'min_samples_split': [2, 4, 8, 16], 'min_samples_leaf': [1, 2, 4, 8, 16] }`
- AdaBoost - using the `AdaBoostClassifier` from sklearn.ensemble. The parameters we grid-searched on where: `{ 'learning_rate': [0.5], 'n_estimators': np.floor(np.linspace(1, 20, 10)).astype(int) }`
- Gradient Boosting - using the `GradientBoostingClassifier` from sklearn.ensemble. The parameters we grid-searched on where: `{ "min_samples_split": [2], "min_samples_leaf": [1], "max_depth":[3], 'n_estimators': np.floor(np.linspace(10, 100, 40)).astype(int) }`

For the ensemble methods, we tried several combinations of parameters in order to deal with overfitting. We saw that a large number of estimators where chosen each time which made us worry that we where overfitting. After reading a few articles and blog posts, we decided to try to limit the decision trees by manipulating the `min_samples_split` and `min_samples_leaf` parameters.

We tried to compare a few more models, such as a multi layer perceptron, but as we didn't manage to get any intersting results we ommitted them from the report.

Model selection

The model selection schema is quite simple - predict on the validation subset, and choose the model which maximizes the accuracy of that prediction.
The final models and parameters chosen for each prediction task were:

```
{
    'Virus': (GradientBoostingClassifier(n_estimators=83),
{'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 2,
'n_estimators': 83}, 0.8183333333333334),
    'Spreader': (GradientBoostingClassifier(n_estimators=86),
{'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 2,
'n_estimators': 86}, 0.8655555555555555),
    'AtRisk': (RandomForestClassifier(max_depth=16), {'max_depth':
16}, 0.8133333333333334)
}
```

To clarify:

- Virus - best modeled by `GradientBoostingClassifier` with accuracy of 81.833%
- Spreader - best modeled by `GradientBoostingClassifier` with accuracy of 86.555%
- AtRisk - best modeled by `RandomForestClassifier` with accuracy of 81.333%%

It is intersting to see that the best models where always ensemble models, and by quite a margin (svc and logistic regression both got less than 76% accuracy, on all prediction tasks).

prediction

Prediction was done after choosing the best model automatically on the test set we received. We first transformed the test set using the data preperation pipeline, and then used each best model to predict the 3 labels.

Dry Part

Question 1

We would like to prove that $\epsilon_t = \sum_{i=1}^{N} D_i^{t+1} \cdot 1_{h_t(x_i) \neq y_i} = \frac{1}{2}$ for h_t the chosen weak classifier.

Recall our definitions from the lecture:

$$D_i^{(t+1)} = \frac{D_i^{(t)} \cdot exp(-w_t y_i h_t(x_i))}{\sum_{j=1}^{N} D_j^{(t)} \cdot exp(-w_t y_i h_t(x_j))}$$

$$w_t = \frac{1}{2} log(\frac{1}{\epsilon_t} - 1)$$

We will later on use the following statement:

$$exp(-2w_t) = exp(-2 \cdot \frac{1}{2} log(\frac{1}{\epsilon_t} - 1)) = \frac{\epsilon_t}{1 - \epsilon_t}$$

For ease of use, we shall denote the set of indices of the examples which where classfied wrongly as A, and A^C for the examples which where classified correctly.
Let's massage the target mathematical definition for the error:

$$\epsilon_t = \sum_{i=1}^{N} D_i^{t+1} \cdot 1_{h_t(x_i) \neq y_i} =$$
$$\sum_{i \in A} D_i^{t+1} \cdot 1 + \sum_{i \in A^C} D_i^{t+1} \cdot 0 =$$

We can cancel out the correct samples as they contribute 0 to the error. Then we can plug in our expression for D_i^{t+1}:

$$\sum_{i \in A} D_i^{t+1} =$$
$$\sum_{i \in A} \frac{D_i^{(t)} \cdot exp(-w_t y_i h_t(x_i))}{\sum_{j=1}^{N} D_j^{(t)} \cdot exp(-w_t y_i h_t(x_j))} =$$

Notice how $y_i h_t(x_i) = -1$ for every wrong sample, simplifyig our expression:

$$\sum_{i \in A} \frac{D_i^{(t)} \cdot exp(-w_t \cdot (-1))}{\sum_{j=1}^{N} D_j^{(t)} \cdot exp(-w_t y_i h_t(x_j))} =$$
$$\sum_{i \in A} \frac{D_i^{(t)} \cdot exp(w_t)}{\sum_{j=1}^{N} D_j^{(t)} \cdot exp(-w_t y_i h_t(x_j))} =$$

We can now seperate the sum in the denominator to two parts - the wrongly classfied samples and the correct ones, and treat the expressions similarly to before:

$$\frac{\sum_{i \in A} D_i^{(t)} \cdot exp(w_t)}{\sum_{j \in A} D_j^{(t)} \cdot exp(-w_t y_i h_t(x_j)) + \sum_{j \in A^C} D_j^{(t)} \cdot exp(-w_t y_i h_t(x_j))} =$$
$$\frac{\sum_{i \in A} D_i^{(t)} \cdot exp(w_t)}{\sum_{j \in A} D_j^{(t)} \cdot exp(w_t) + \sum_{j \in A^C} D_j^{(t)} \cdot exp(-w_t)} =$$
$$\frac{\sum_{i \in A} D_i^{(t)}}{\sum_{j \in A} D_j^{(t)} + \sum_{j \in A^C} D_j^{(t)} \cdot exp(-2w_t)} =$$

Finally, we managed to get to the given expression we needed, now we just need to plug in our calculation of $exp(-2w_t)$:

$$\frac{\epsilon_t}{\epsilon_t + (1-\epsilon_t)\cdot exp(-2w_t)} =$$
$$\frac{\epsilon_t}{\epsilon_t + (1-\epsilon_t)\cdot \frac{\epsilon_t}{1-\epsilon_t}} =$$
$$\frac{\epsilon_t}{\epsilon_t + \epsilon_t} =$$
$$\frac{1}{2}$$

Question 2

2.1

We shall show that $F_{\alpha\Theta}(x) = c \cdot F_{\Theta}(x)$ where $c = \alpha^L$ for all $L > 1$. We proove this using induction over L.

Base: For $L = 2$ (A nueral network must be with at least 2 layers).

$$F_{\alpha\Theta}(x) = \alpha {W^{(2)}}^T \cdot h^{(1)}_{\alpha\Theta}(x) =$$
$$\alpha W^{(2)} \cdot \sigma(\alpha {W^{(1)}}^T x) =$$
$$\alpha^2 W^{(2)} \cdot \sigma({W^{(1)}}^T x) =$$
$$\alpha^2 W^{(2)} \cdot h^{(1)}_{\Theta}(x) =$$
$$\alpha^2 F_{\Theta}(x)$$

Step: We assume that $F_{\alpha\Theta}(x) = \alpha^L \cdot F_{\Theta}(x)$ for all $L \leq k$ for some $k \in \mathbb{N}$. Therfore for $L = k+1$ we can show that:

$$F_{\alpha\Theta}(x) = \alpha {W^{(k+1)}}^T \cdot h^{(k)}_{\alpha\Theta}(x) =$$
$$\alpha {W^{(k+1)}}^T \cdot \sigma(\alpha {W^{(k)}}^T \cdot h^{(k-1)}_{\alpha\Theta}(x)) =$$
$$\alpha {W^{(k+1)}}^T \cdot \sigma(\alpha^k {W^{(k)}}^T \cdot h^{(k-1)}_{\Theta}(x)) =$$
$$\alpha^{k+1} {W^{(k+1)}}^T \cdot \sigma({W^{(k)}}^T \cdot h^{(k-1)}_{\Theta}(x)) =$$
$$\alpha^{k+1} {W^{(k+1)}}^T \cdot h^{(k)}_{\Theta}(x) =$$
$$\alpha^{k+1} F_{\Theta}(x)$$

2.2

Let's look on the softmax expression for one target label:

$$C_{y_i} = \frac{exp(\alpha^L \cdot (F_{\Theta}(x))_i)}{\sum_{j=1}^{K} exp(\alpha^L \cdot (F_{\Theta}(x))_j)}$$

We can try taking the logarithm to simplfy the term:

$$log(C_{y_i}) = \alpha^L \cdot (F_{\Theta}(x))_i) - log(\sum_{j=1}^{K} exp(\alpha^L \cdot (F_{\Theta}(x))_j))$$

Now taking the limit becomes simpler, as many terms either become 0, or $exp(0) = 1$. And we get:

$$Lim_{\alpha \to 0} log(C_{y_i}) = -log(K)$$
$$Lim_{\alpha \to 0} C_{y_i} = \frac{1}{K}$$

This means that the distribution induced by the softmax function is actually a discrete uniform distribution with K labels.

2.3

Let's now consider again the softmax expression for one category i:

$$C_{y_i} = \frac{exp(\alpha^L \cdot (F_\Theta(x))_i)}{\sum_{j=1}^{K} exp(\alpha^L \cdot (F_\Theta(x))_j)}$$

We now have 2 options, either i is the index of the maximal score of $F_\Theta(x)$, or it's not.
If it is not the maximal score, then the limit will converge as follows:

$$\begin{aligned} 0 < C_{y_i} &= \frac{exp(\alpha^L \cdot (F_\Theta(x))_i)}{\sum_{j=1}^{K} exp(\alpha^L \cdot (F_\Theta(x))_j)} \\ &< \frac{exp(\alpha^L \cdot (F_\Theta(x))_i)}{exp(\alpha^L \cdot (F_\Theta(x))_{max})} \\ &= exp(\alpha^L \cdot ((F_\Theta(x))_i) - (F_\Theta(x))_{max})) \end{aligned}$$

But $((F_\Theta(x))_i) - (F_\Theta(x))_{max})$ is defintly negative, which means the whole expression converges to 0 as $\alpha \to \infty$.

Because of this, the only possible value the C_{y_i} when $i = max$ can get is 1.

Basically, this makes the probability distribution a "spiky" one around the maximal score, which kind of makes the whole model "more decisive" of what label to give sample x.
In contrast, we saw when $\alpha \to 0$ the model is not decisive at all.