Homework No. 5

Alexander Shender 328626114

Snir Hordan 205689581

Question 1.

1. Prove that when running AdaBoost, the distribution is updated such that the error of the chosen weak classifier h_t, w.r.t the updated distribution $D_i^{(t+1)}$, is exactly $\frac{1}{2}$.

   That is, prove that $\sum_i D_i^{(t+1)} \cdot \mathbf{1}_{h_t(x_i)\neq y_i} = \frac{1}{2}$.

   Hint: You can fill the missing steps in the following derivation:

   $$\sum_i D_i^{(t+1)} \cdot \mathbf{1}_{h_t(x_i)\neq y_i} = \cdots = \frac{\epsilon_t}{\epsilon_t + (1-\epsilon_t)\exp\{-2w_t\}} = \cdots = \frac{1}{2}.$$

We start by indeed from writing the expression for the updated error value with the updated data weights (distribution) for the last chosen weak classifier:

$$\mathrm{E}_{t+1} = \sum_i D_i^{t+1} \cdot 1_{h_t(x_i)\neq y_i}$$

By using:

$$D_i^{t+1} = D_i^t \cdot \frac{\exp\left(-w_t y_i h_t(x_i)\right)}{\sum_j D_j^t \exp\left(-w_t y_j h_t(x_j)\right)} = D_i^t \cdot \frac{\exp\left(-w_t y_i h_t(x_i)\right)}{Z_t}$$

Putting back:

$$\mathrm{E}_{t+1} = \sum_i D_i^{t+1} \cdot 1_{h_t(x_i)\neq y_i} = \sum_i \frac{D_i^t \exp\left(-w_t y_i h_t(x_i)\right)}{Z_t} \cdot 1_{h_t(x_i)\neq y_i}$$

Z_t is a normalization factor, so we can put it outside of the sum:

$$\mathrm{E}_{t+1} = \frac{\sum_i D_i^t \exp\left(-w_t y_i h_t(x_i)\right)}{Z_t} \cdot 1_{h_t(x_i)\neq y_i}$$

We divide into 2 cases:

$$\begin{cases} E : h_t(x_i) = y_i \\ C : h_t(x_i) \neq y_i \end{cases}$$

For each case,

$$\begin{cases} E : 1_{h_t(x_i)\neq y_i} = 0\,;\ y_i h_t(x_i) = 1 \\ C : 1_{h_t(x_i)\neq y_i} = 1\,;\ y_i h_t(x_i) = -1 \end{cases}$$

So we get:

$$\mathrm{E}_{t+1} = \frac{\sum_{i\in E} D_i^t \exp\left(-w_t y_i h_t(x_i)\right)}{Z_t} \cdot 0 + \frac{\sum_{i\in C} D_i^t \exp\left(-w_t y_i h_t(x_i)\right)}{Z_t} \cdot 1 = \frac{\sum_i D_i^t \exp(w_t)}{Z_t}$$

The numerator contains the expression for the error value, since we have isolated for case $\{E :\ h_t(x_i) = y_i\}$:

$$\mathrm{E}_t = \sum_i D_i^t$$

$$\mathrm{E}_{t+1} = \frac{\mathrm{E}_t \exp(w_t)}{Z_t}$$

Opening the denominator using same 2 cases:

$$Z_t = \sum_{i \in E} D_i^t \exp(-w_t) + \sum_{i \in C} D_i^t \exp(w_t)$$

Putting back:

$$\mathrm{E}_{t+1} = \frac{\mathrm{E}_t \exp(w_t)}{\sum_{i \in E} D_i^t \exp(-w_t) + \sum_{i \in C} D_i^t \exp(w_t)} \quad ; \ / \exp(w_t)$$

$$\mathrm{E}_{t+1} = \frac{\mathrm{E}_t}{\sum_{i \in E} D_i^t \exp(-2w_t) + \sum_{i \in C} D_i^t} = \frac{\mathrm{E}_t}{\sum_{i \in E} D_i^t \exp(-2w_t) + \mathrm{E}_t}$$

Taking $\exp(-2w_t)$ out of the sum:

$$\mathrm{E}_{t+1} = \frac{\mathrm{E}_t}{\exp(-2w_t) \sum_{i \in E} D_i^t + \mathrm{E}_t}$$

The sum over weights of the correct predictions E, is 1-sum over incorrect predictions, since they sum to 1:

$$\sum_{i \in E} D_i^t = 1 - \sum_{i \in C} D_i^t = 1 - E_t$$

So we get:

$$\mathrm{E}_{t+1} = \frac{\mathrm{E}_t}{\exp(-2w_t)\,(1 - E_t) + \mathrm{E}_t}$$

Using the expression for the weight of the weak classifier:

$$w_t = \frac{1}{2} \log\left(\frac{1}{\mathrm{E}_t} - 1\right)$$

$$\exp(-2w_t) = \exp\left(-log\left(\frac{1}{\mathrm{E}_t} - 1\right)\right) = \exp\left(\log\left(\left(\frac{1}{\mathrm{E}_t} - 1\right)^{-1}\right)\right) = \left(\frac{1}{\mathrm{E}_t} - 1\right)^{-1} = \left(\frac{1 - \mathrm{E}_t}{\mathrm{E}_t}\right)^{-1}$$
$$= \frac{\mathrm{E}_t}{1 - \mathrm{E}_t}$$

Putting everything back:

$$\mathrm{E}_{t+1} = \frac{\mathrm{E}_t}{\frac{\mathrm{E}_t}{1 - \mathrm{E}_t}(1 - E_t) + \mathrm{E}_t} = \frac{\mathrm{E}_t}{\mathrm{E}_t + \mathrm{E}_t} = \frac{1}{2}$$

Question 2

2.a.

Let $\alpha \in \mathbb{R}_{>0}$

Notice σ is positive-homoegeneous, i.e. $\sigma(\alpha x) = \max\{0, \alpha x\} = \alpha \max\{0, x\} = \alpha\sigma(x)$

Denote $\alpha\theta = \left(\alpha W^{(1)}, ..., \alpha W^{(L)}\right)$.

Let h be defined on θ as in question and $\tilde{h}$ defined on $\alpha\theta$ as in question.

Claim. $\tilde{h}_L(x) = \alpha^L h_L(x)$

Proof. Proof by induction on L.

Base : $\tilde{h}_1(x) = \sigma\left(\alpha {W^{(1)}}^T x\right) \underbrace{=}_{\alpha\, positive\, homogeneous} \alpha\sigma\left({W^{(1)}}^T x\right) = \alpha h_1(x)$

Assume for n<L, $\tilde{h}_n(x) = \alpha^n h_n(x)$

Induction step:

By definition, $\tilde{h_L}(x) = \sigma\left(\alpha {W^{(L)}}^T \tilde{h}_{L-1}(x)\right) \underbrace{=}_{induction\, hypothesis} \sigma\left(\alpha {W^{(L)}}^T \alpha^{L-1} h_{L-1}(x)\right) =$

$\underbrace{=}_{\alpha\, positive-homogeneous} \alpha^L \sigma\left({W^{(L)}}^T h_{L-1}(x)\right) = \alpha^L h_L(x)$ □

By definition, $F_{\alpha\theta}(x) = {W^{(L)}}^T \tilde{h}_{L-1}(x) = {W^{(L)}}^T \alpha^{L-1} h_{L-1}(x) = \alpha^{L-1} F_\theta(x)$

Then $C = \alpha^{L-1}$.

2.b.

$$c_{y_i} = \frac{\exp\left(\left(F_{\alpha\theta}(x)\right)_i\right)}{\sum\limits_{j=1}^{k} \exp\left(\left(F_{\alpha\theta}(x)\right)_j\right)} = \frac{\exp\left(\alpha^L \left(F_\theta(x)\right)_i\right)}{\sum\limits_{j=1}^{k} \exp\left(\alpha^L \left(F_\theta(x)\right)_j\right)} \underset{\alpha \to 0}{\to} \frac{1}{k}$$

Then the distribution is uniform.

2.c.

<u>Case 1</u> $\left(F_\theta(x)\right)_i > \left(F_\theta(x)\right)_j, \forall j \in \{m_1, ..., m_{k-s}\}$ and $\left(F_\theta(x)\right)_i = \left(F_\theta(x)\right)_j, \forall j \in \{m_{k-s+1}, ..., m_k\}$, $(\{m_1, ..., m_k\} = \{1, ..., k\})$

$$c_{y_i} = \frac{\exp\left(\left(F_{\alpha\theta}(x)\right)_i\right)}{\sum\limits_{j=1}^{k} \exp\left(\left(F_{\alpha\theta}(x)\right)_j\right)} = \frac{\exp\left(\alpha^L \left(F_\theta(x)\right)_i\right)}{\sum\limits_{j=1}^{k} \exp\left(\alpha^L \left(F_\theta(x)\right)_j\right)} = \frac{1}{\sum\limits_{j=1}^{k} \exp\left(\alpha^L \left(\left(F_\theta(x)\right)_j - \left(F_\theta(x)\right)_i\right)\right)} = ...$$

$$... = \frac{1}{s + \underbrace{\sum\limits_{j \in \{m_1, ..., m_{k-s}\}} \exp\left(\alpha^L \left(\left(F_\theta(x)\right)_j - \left(F_\theta(x)\right)_i\right)\right)}_{\underset{\alpha\to\infty}{\to} 0}} \underset{\alpha \to \infty}{\to} \frac{1}{s}$$

($s \geq 1$ because exists at least one maximum value then limit is well-defined)

<u>Case 2</u> $\exists 1 \leq j \leq k\, s.t. \quad \left(F_\theta(x)\right)_i < \left(F_\theta(x)\right)_j$

$$c_{y_i} = \frac{1}{1 + \underbrace{\sum\limits_{j \neq i} \exp\left(\alpha^L \left(\left(F_\theta(x)\right)_j - \left(F_\theta(x)\right)_i\right)\right)}_{\underset{\alpha\to\infty}{\to} \infty}} \underset{\alpha \to \infty}{\to} 0$$

Then distribution is uniform on the $s \leq k$ entries with maximum value and zero otherwise.

i.e. $\left(0, ..., \underbrace{\frac{1}{s}}_{m}, ..., 0\right)$ where $m \in argmax_{1 \leq i \leq k} \left(\left(F_{\theta}\left(x\right)\right)_{i}\right)$.

Wet section

To succeed in the classification task in the best possible way, we first had to go back into feature imputation stage.

Then we have tried more methods to succeed in the classification task, such as trying to verify the disease separation using clustering, or using SMOTE to tackle the class imbalance.

For classification, high hopes were on the MLP models, which were eventually beaten by the AdaBoost classifier. We have also tried transfer learning, which didn’t succeed to beat AdaBoost either.

In the end we succeeded to reach above 80% accuracy on each of the tasks using the AdaBoost classifier model.

Code Review

As before, our code is fully automatic, and performs the model selection automatically.

To the existing model we have added new models, which are AdaBoost, and the MLP.

The same actions on features are later done on the unseen data, and the output is given.

- The code is being provided, and you can run it by running the "automatic_classification_main.py" file.

2 output folders will be created:

1. outputs_csv – includes the datasets after the features operation on each of them. (It supports different feature operations for different tasks)
2. outputs_clf – contains the outputs – the test & unseen dataset predictions, and log file.

Features imputation

Some feature imputations techniques depend on the dataset, such as replacing missing values with the mean value. For such cases, the mean values found from the operation on the training set were found and saved. And later used on the test set and the unseen data.

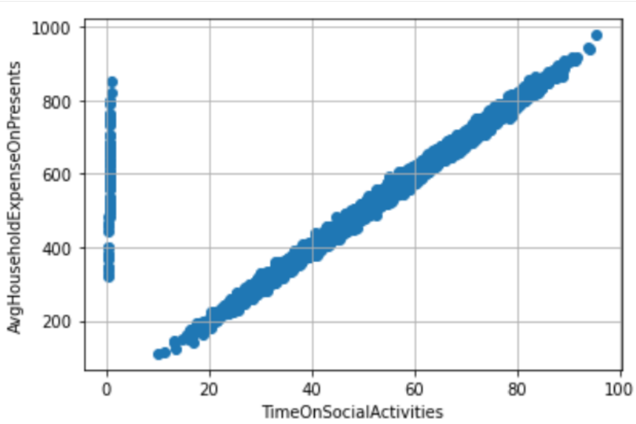
All the feature operations can be seen in the “given_features_pipe” procedure in the “data_preparation.py” file from the code.

1. Discipline score

The discipline score had outliers, but it was noticed that they are simply multiplications of existing discipline scores by 10. Those outliers were divided by 10. The other missing were replaced with a mean.

2. Social Activities

There is a big correlation between the TimeOnSocialActivities and the AvgHouseholdExpenseOnPresents graphs:

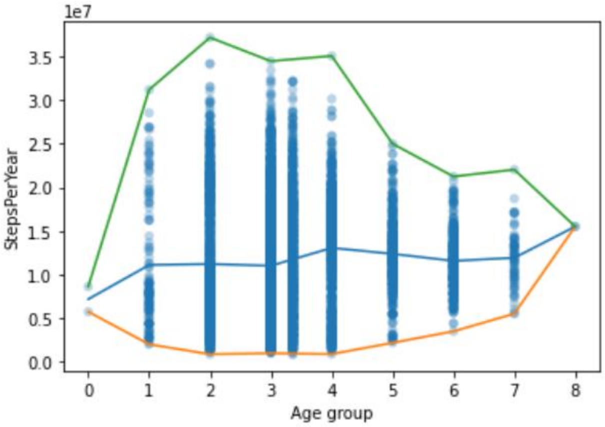


The following imputations are done:

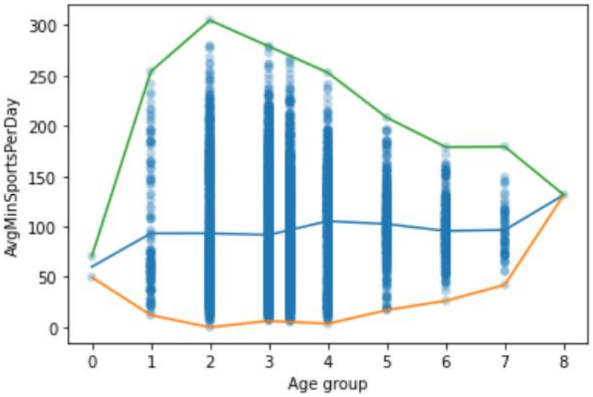
- The outliers (TimeOnSocialActivities < 5) was replaced according to AvgHouseholdExpenseOnPresents
- The non-existing values were also replaced according to AvgHouseholdExpenseOnPresents (if those existed there)
- For all the rest, replacing with the mean value of TimeOnSocialActivities

3. Age Handler

To handle missing age, we have examined the AvgMinSportsPerDay and StepsPerYear parameters, and this is the distribution (with upper and lower boundaries)



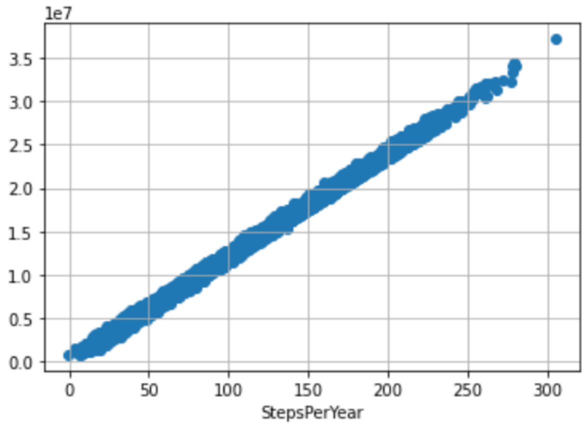
Same for AvgMinSportsPerDay:



We can see some correlation (which is logical).

4. Steps Per Year

There is a strong correlation between AvgMinSportsPerDay and StepsPerYear:



So if the value for StepsPerYear is missing, but is existing in AvgMinSportsPerDay, we calculate this value. Other values are replaced by the mean value.

5. PCR Results

Like with the case with clustering (homework no. 4) we try both the mean and the KNN imputer. For each PCR result, we use the z-value to discover outliers. Then calculate the mean without the outliers. Then we replace the outliers and the missing values by those means or the KNN.

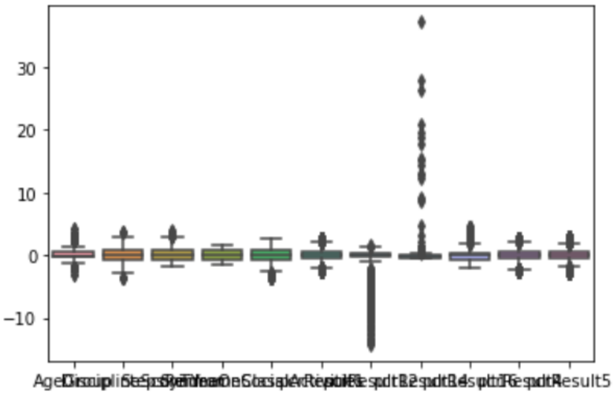
- For training set: we SAVE the mean and STD values that were calculated
- For test/unseen sets: we use the saved mean and std values for each feature to find outliers in the unseen dataset. (we use the same z-score value). We then replace the outliers and missing values with the mean.

6. Syndrome Class

No correlation was found between this feature and any other feature. So we add a new categorical value ‘5’ for ‘unknown syndrome class’.

Features Scaling

StandardScaler was used on all features in the training dataset. The boxplot distribution of the data after the scaling:



After being performed on the training dataset, the mean and std parameters from StandardScaler object were saved.

Later, those parameters were used to do MANUAL standard scaling on validation and test datasets using the equation:

$$scaled\ X = \frac{X - \mu}{\sigma}$$

Where µ is the mean and σ is the std of the feature (from the training dataset).

This way, the distribution of the test dataset does not affect the scaling, and it is same for every dataset.

Cluster Analysis

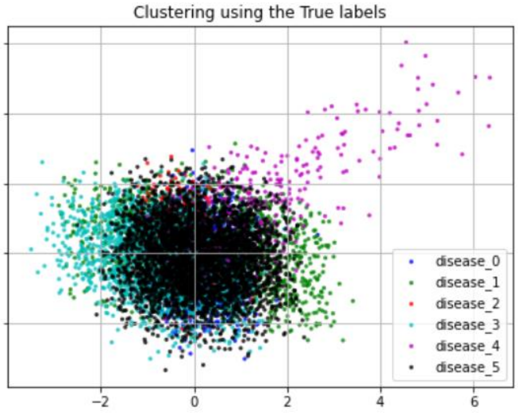
By learning to perform the cluster analysis, we tried to visualize the 6 clusters of the disease type. This would also help us to verify that the feature imputation procedures were indeed effective.

We use the most successful clustering model from the previous homework, which is GaussianMixture Model, since most of the features distributions resemble Gaussian distribution.

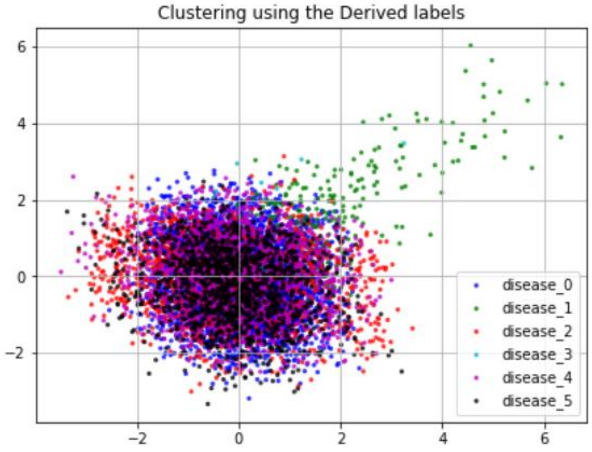
To visualize, we use the 2 PC (3d plot doesn't add much more information).

We used the whole training set of 9000 samples.

Using the TRUE samples (from the dataset):



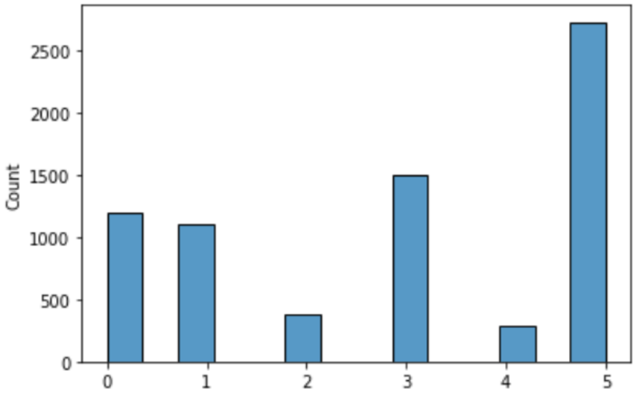
And using the Derived cluster indexes from the GMM:



We can see that GMM succeeded to discover some clusters (like the easy one which spans through big x and y values), but in general it doesn't correspond too much. We will later use this visualization to assert the precision of the classifier.

Class Imbalance

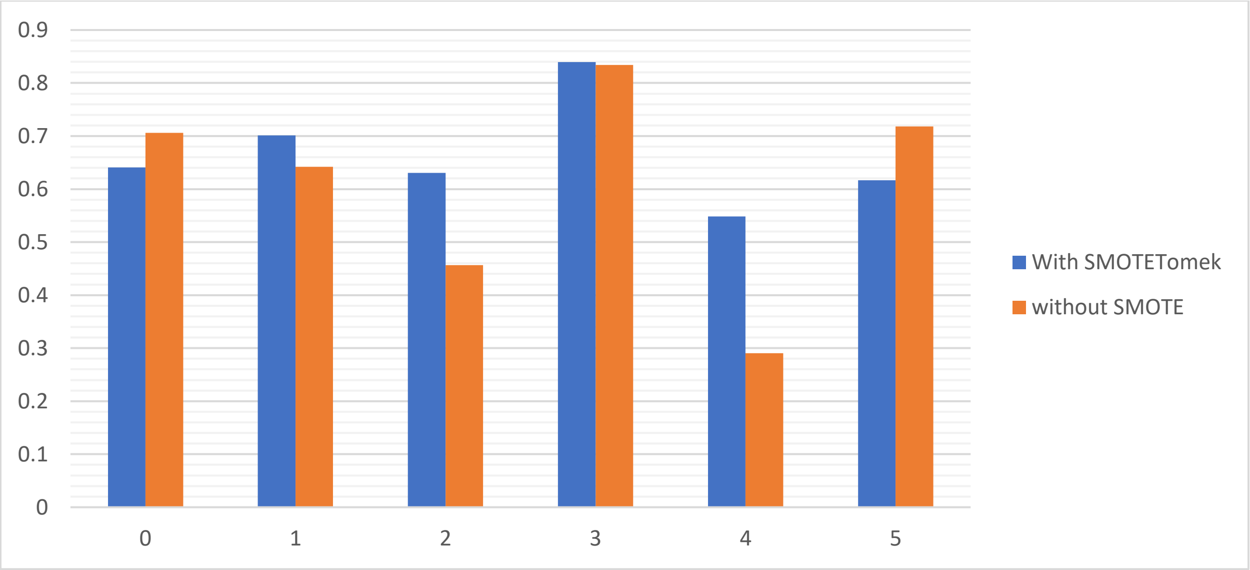
For the disease type, the distribution of the classes in original dataset is not balanced:



We handle the imbalanced data using SMOTE (Synthetic Minority Oversampling Technique), which created new samples for the minority classes, equating them with the majority class. The class imbalance handling was also added to the automatic training process.

After a research, SMOTE didn't give us the expected results. If fact it was misleading. After obtaining up to 90% on the dataset after SMOTE (using MLP classifier, with built-in validation ratio of 0.15) we could only reach up to 70% on the unseen validation set. More than that, the same MLP which was trained on the original dataset (without SMOTE) gave better validation scores. We illustrate it in this graph:

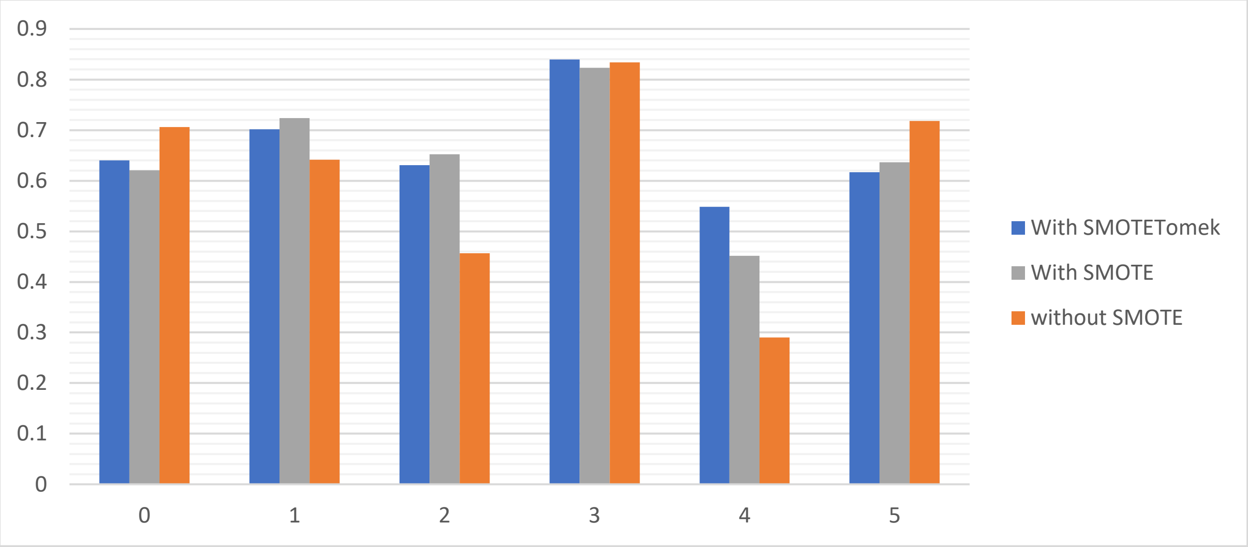
- Total validation accuracy with SMOTETomek: 609/900 (67.667 %)
- Total validation accuracy without SMOTETomek: 630/900 (70.0 %)



```
Those are the mappings of the disease indexes:
{'flue': 0, 'covid': 1, 'cmv': 2, 'cold': 3, 'measles': 4, 'notdetected': 5}
```

On the graph we can observe that SMOTE did indeed help where expected. It does perform better on diseases 1,2,3,4 – especially 2 and 4, which were heavily underrepresented. But because the method that was used was SMOTETomek, which uses Tomek Links, the majority oversamped classes get their samples removed. After using regular SMOTE, which only deals with undersampling (and bringing the amount of samples in each disease to be equal to the maximum value among diseases we receive (using same network):

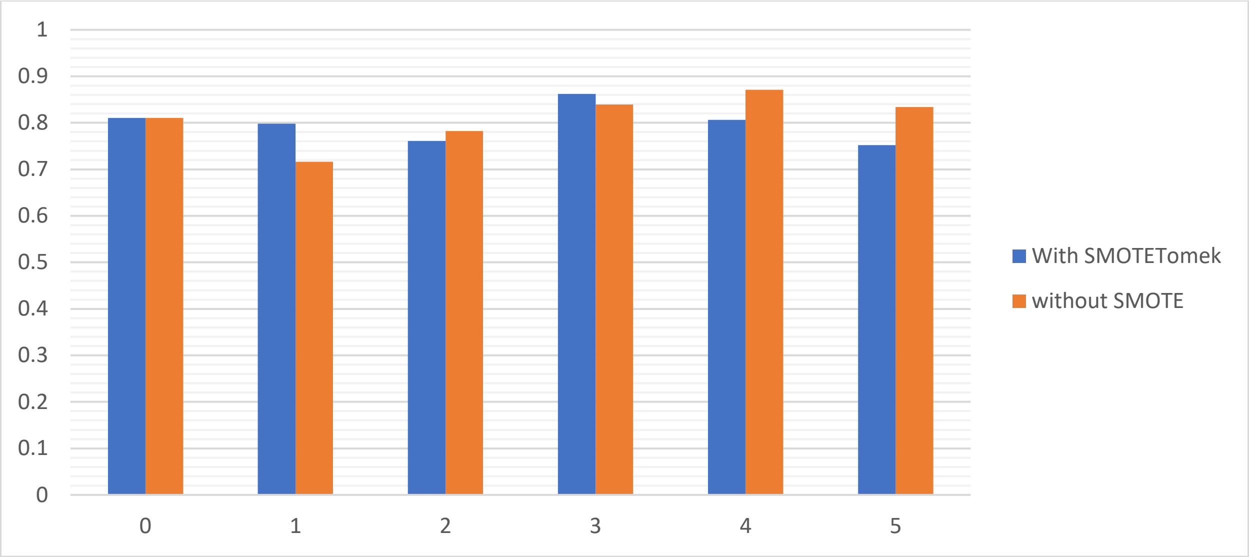
Total validation accuracy with SMOTE: 611/900 (67.889 %)



We can see that the results are quiet similar between SMOTE and SMOTETomek.

We do the same verification on the AdaBoost with the best chosen parameters:

- Total validation accuracy with SMOTETomek: 714/900 (79.333 %)
- Total validation accuracy without SMOTETomek: 731/900 (81.222 %)



We can see that the results are much more similar, and the effect of the SMOTE is not visible (we can see that previously minority classes (2 and 4) get their performance REDUCED after using SMOTE, which is surprising.

Conclusion:

We can see positive effects of SMOTE to detect minority classes for specific models like MLP. This is a significant effect for tasks where the minority undersampled case is the important target that we want to detect, like anomaly, or an illness. And should be used in those cases.

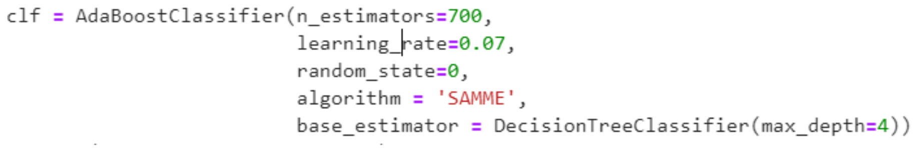
Here, we see that Adaboost performs better, and SMOTETomek doesn't improve it's classification abilities, so in order to succeed in the competition (where the accuracy is the decisive score, it was decided **NOT TO USE** SMOTE of any kind.

Classification

We have implemented new gained knowledge to tackle the classification problem with more success than previously. We first concentrated on the Disease classification, since it was the most challenging task from the HW3. Here we talk on the NEW models that were tested (and were not tested in the HW no. 3).

1. Adaboost

We have used the AdaBoost classifier with various parameters, experimenting with the algorithm (sklearn offers SAMME, SAMME.R), number of estimators, base estimator and its parameters. The most successful parameters found manually were the following:



```
clf = AdaBoostClassifier(n_estimators=700,
                         learning_rate=0.07,
                         random_state=0,
                         algorithm = 'SAMME',
                         base_estimator = DecisionTreeClassifier(max_depth=4))
```

Which gave around 86% on the validation set for the Disease classification (after SMOTETomek). Later while using the automatic training we used the CV to test around 25 variations, and the best one was similar to the one presented here (max_depth=5 was found a bit better).

Pros

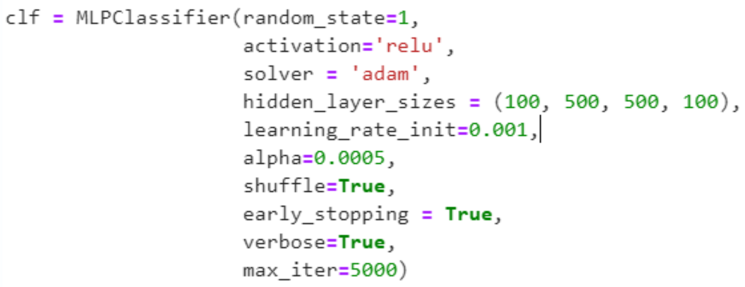
- Freedom of choosing base estimators.
- Easy to implement.

Cons

- Weak base estimators can lead to low margins overfitting
- Uniform noise can distort results

2. MLP

The sklearn library provides the MLP classifiers. We have as well examined different parameters variations; we found the following to be relatively fast converging and accurate:



```
clf = MLPClassifier(random_state=1,
                    activation='relu',
                    solver = 'adam',
                    hidden_layer_sizes = (100, 500, 500, 100),
                    learning_rate_init=0.001,
                    alpha=0.0005,
                    shuffle=True,
                    early_stopping = True,
                    verbose=True,
                    max_iter=5000)
```

In reality it would converge already after 50 epochs (iterations), with the accuracy of 89% on the test set (after SMOTE), thus proving superior to the Adaboost in this case. But as we saw, the MLP model doesn't perform well on the validation set after being trained on a dataset using SMOTE.

Pros

- Flexibility of optimizing classification and regression problems.
- Free to choose any number of layers, thereby being able to solve complex pattern recognition.
- Improves with more data.

Cons

- No clear guideline on how to choose hyperparameters (layer sizes, amount of layers, learning rate, etc.)
- Computationally expensive.
- Requires large amounts of data to converge

3. MLP + Transfer Learning

We have used the Keras library to build a new network.

In transfer learning, we used the following process:

a. First, train the binary classification on the Disease (0 = No disease, 1 = Any other disease). Use the SMOTE to fix class imbalance between classes (No disease, Disease)
b. Build a new model based on the trained model in (a.). Add new layers (only those layers will be trained)
c. Train the new model.

Using this process, we could achieve about 55% of accuracy on the Disease classification, which didn't improve our earlier tries.
Our analysis is that transfer learning did not improve results, because binary classification of disease/no disease did not generalize to differentiating between diseases, which led to low accuracy. This occurred even though the no disease class is largest, which is why we tried this method out in first place.

Transfer learning is usually used for NLP and Image Recognition and trained on much larger datasets. This is another reason for the low accuracy derived from the model.

(The code is available in transfer.py in the submitted work.)

4. Ensemble Learning

Ensemble learning is learning by a collection of estimators where we take a hard vote, regarding the best classification for each instance agreed on by most models, or a soft vote, summing over the probability assigned to each class and then picking the class with the maximum sum of probabilities assigned to it.

If the underlying collection of models is robust then ensemble learning provides a mechanism to increase certainty in the classification task.

Results

After running the automatic model selection, those are the models that were chosen by the process and their metrics. We have tested numerous models (including Ensemble model as asked in the assignment) as previously in a fully automatic manner. Big hopes were places on the Deep Networks which didn't give the expected results.

The predictions file was also created automatically in a format similar to provided template using the best model for each task.

The log containing accuracies for all the models tried can be found in "output/log.txt" along with "predicted.csv".

Those were the models in final automatic selection:

- RandomForestClassifier(max_depth=3, max_features=10, n_estimators=500) over params {}
- KNeighborsClassifier() over params {'n_neighbors': [3, 5, 10, 20, 50]}
- LinearSVC() over params {}
- GaussianNB() over params {}
- DecisionTreeClassifier() over params {'max_depth': [5, 10, 15, 20]}
- LogisticRegression(max_iter=1000) over params {}
- OneVsRestClassifier(estimator=DecisionTreeClassifier(max_depth=5)) over params {}
- VotingClassifier(estimators=[('mlp', MLPClassifier(hidden_layer_sizes=(100, 100, 100), max_iter=100000)), ('rf', RandomForestClassifier(max_depth=3, max_features=10, n_estimators=500)), ('ada', AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=5), n_estimators=100))]) over params {}
- VotingClassifier(estimators=[('mlp1', MLPClassifier(hidden_layer_sizes=(100, 150, 150, 100, 10), max_iter=150)), ('mlp2', MLPClassifier(hidden_layer_sizes=(50, 100, 200, 200, 100), max_iter=150)), ('mlp3', MLPClassifier(hidden_layer_sizes=(50, 100, 100, 200, 100), max_iter=150)), ('ada', AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=4), n_estimators=100))]) over params {}
- AdaBoostClassifier(algorithm='SAMME', base_estimator=DecisionTreeClassifier(max_depth=5), learning_rate=0.07, n_estimators=700, random_state=0) over params {}
- MLPClassifier(alpha=0.0002, early_stopping=True, hidden_layer_sizes=(100, 1000, 1000, 100), learning_rate='adaptive', learning_rate_init=0.005, max_iter=1000, n_iter_no_change=5, random_state=1, solver='sgd', validation_fraction=0.15) over params {}

Risk detection

For risk detection the chosen model was AdaBoostClassifier(algorithm='SAMME', base_estimator=DecisionTreeClassifier(max_depth=5), learning_rate=0.07, n_estimators=700, random_state=0)

- The f1 score reached on the validation: 0.853
- The f1 score reached on the test set: 0.834

Metric Selection :

F1 score is ideal because we need a balance of low false negatives and high accuracy, because don't want to tell healthy people they are at risk but also don't want to make someone falsely certain he will have no consequences rom the virus

Spreader detection

For risk detection the chosen model was DecisionTreeClassifier with depth of 5

- The accuracy reached on the validation: 0.942
- The accuracy reached on the test set: 0.955

Metric Selection :

Recall is the optimal metric to assess risk of spread, because a False Negative should be penalized harshly. A false negative can lead to massive spread so we would rather stay on the side of safety.

Disease detection

For risk detection the chosen model was AdaBoostClassifier(algorithm='SAMME', base_estimator=DecisionTreeClassifier(max_depth=5), learning_rate=0.07, n_estimators=700, random_state=0)

- The accuracy reached on the validation: 0.81
- The accuracy reached on the test set: 0.828

Metric selection:

Different diseases require varied interventions, therefore accuracy is vital.