# Haensel-AMS Challenge

## I. Problem Description

The Challenge data set is a 40 MB csv file containing over 66k instances/rows and 295 features/columns. The target variable is the last column (the 296th) and it is categorical with values/classes A, B, C, D, E. The challenge to analyze the data and to build some initial ML model(s) which predicts the classes. It is required that one of the used ML techniques is an ANN.

## II. Steps Taken

In this section, all the steps taken in approaching the problem are summarized, including the logic behind applying them and the pros and cons of applying chosen techniques. A summary of the obtained models' results and possible next steps is provided in Section III.

1. **Exploratory Data Analysis (EDA):** After examining the data briefly, the following points can me made:
   - The target classes (A, B, C, D, E) are highly unbalanced. Class C is the most dominant with 71% of the instances, followed by Class D with 14%, Class B with 10%, Class E with 4% and Class A with 1%. This means that building a ML model that can learn the minority classes, especially E and A, is expected to be challenging.
   - The data has no missing values in the form of NaNs.
   - There are eleven constant columns. These will be dropped in the data cleaning step.
   - The vast majority (288/295) of the attributes columns are binary. This can be because the columns represent dummy variables of categorical attributes or because they represent pixels of a black and white image.
   - There are four real-number/float attributes. Their distribution for every target class was checked and all four columns have outliers
   - There are 3 non-binary integer attributes (columns 4, 23 and 36). Their distributions have extreme outliers, especially column 36. Columns 4 and 23 are mostly constant at 0 and -1 respectively. Column 36 is mostly binary (80%) and 98.5% of its values lie between 0 and 20. The remaining 1.5% contains very extreme outliers.
   - There is are moderate to strong correlations between the float columns.

2. **Data Cleaning:** based on the above observations drawn from the EDA, the following steps were taken:
   - The 11 constant-valued columns were dropped from the data.
   - Columns 4 and 23 were dropped for their low variability.
   - Rows corresponding to values above 20 in column 36 are dropped. These comprise about 1.5% of the data. This action was taken to avoid affecting the scaling/standardization of this column negatively due to their initial very high variance in latter steps.
   - No rows were dropped to exclude outliers from the four float columns. This action was avoided since the nature of these columns/attributes is unknown and the outliers are not many orders of magnitude greater than their mean as is the case with column 36.

3. **Building Classifiers**
   - **Validation Metric:** Due to the imbalance in the target classes in the data, Accuracy will not be a good model validation metric because it will only reflect the underlying class distribution (predominantly Class C). Instead, the Confusion Matrix, and the average Precision, Recall and F1-score are used since these metrics together show how the model is performing in terms of correctly learning each class.

- **Strategy:** Ideally, the data should be split into training and testing from the onset. The performance of each classifier should then be estimated using a 10-fold cross-validation procedure. This will give an estimate about the average F1-score that can be obtained by the model on the test data. Then after performing this for all tested classifiers, each should be applied to the test data and the metrics should be examined and compared. However, since we have a relatively large dataset, the training of some of the classifiers is expensive in terms of CPU-time and memory. Hence, for each classifier, we train once using the training data and evaluate performance roughly using the test data.

**3.1. Random Forest (RF) Classifier:** RF was chosen to get a base model since the technique has the following advantages:

- RF can work with a mix of binary and float attributes (without the need to standardize attributes)
- RF can work with the relatively large number of attributes without applying dimensionality reduction first.
- RF is robust against correlated attributes, which are present in this data, at for the float attributes.

Class weights can be set in an RF model (as a dictionary or the option 'balanced'), which may enhance performance in case of unbalanced data.

**3.2. K-Nearest Neighbor (KNN) Classifier:** KNN was applied to the data after standardizing (z-score) all the attributes. The tunable parameter K was set arbitrarily to 10. KNN has the advantage of being a simple ML technique with low computational complexity. Hence it can be used to train relatively large and high dimensional datasets that grows progressively over time. However, its downside is that it is expensive in terms of memory since the trained model is represented by the entire dataset. Also, efficiency of this classifier deteriorates in problems with very high dimensions.

**3.3. Principal Component Analysis (PCA) & Support Vectors Machine (SVM):** The next classifier applied is SVM. Due to the relatively high number of attributes (and also records) PCA will be used to reduce the dimensions of the data before applying SVM. After fitting PCA using the train data, we trained the SVM model with 174 principal components (instead of 282 attributes, 40% compression). The main advantage of using SVM is:

- SVM has a generally good performance in classifications problems where the classes are not linearly separable due to its usage of the "kernel trick" technique. The problem at hand clearly falls in this category.
- Once the model has been trained, i.e. the hyperplanes dividing the classes (aka support vectors) have been identified, the training data need not be kept as in KNN.

On the other hand, the high computational complexity of SVM makes its training very time consuming for large high-dimensional datasets. Also, SVM's parameters C and gamma usually require tuning, which is can be done using a GridSearch. However, for large high-dimensional datasets such as the one at hand, this step can be computationally very expensive.

**3.4. Feed-Forward Artificial Neural Network (FF-ANN): The** last classifier applied to the data is a FF-ANN. To build the model in Python, Keras library is used, with a Tensorflow backend. A FF one-hidden layer ANN was built with the following architecture:

- $N_i = 282$, where $N_i$ is the number of perceptrons in the input layer, which is equal to the number of attributes in the data.
- $N_o = 5$, where $N_o$ is the number of perceptrons in the output layer, which is in turn equal to the number of classes in the data. The output/response column is hence one-hot encoded.
- $N_h = 80$, where $N_h$ is the number of perceptrons in the hidden layer. was set according to the rough rule of thumb $N_h = N_t/2(N_i + N_o)$, where $N_t$ is the number of records in the training data.
- RELU rectifier activation function for perceptrons in the input and hidden layers.
- A SotfMax activation function for the perceptrons in the output layer to match the one-hot output encoding.

- The number of perceptrons in the hidden layer was set according to the rough rule of thumb $N_h = N_t/2(N_i + N_o)$, where $N_t$ is the number of records in the training data, $N_i$ is the number of records in the training data.
- The Adam optimizer is used for weights updates due to its reported superior performance to other stochastic gradient descent optimizers.
- A Cross-Entropy function is used as the error/loss measure.

Input columns/attributes where scaled such that the non-binary columns values have a range between 0 and 1, which is important for correct weights training of the ANN.
The main advantage of a using a Deep Learning (DL) technique such as a ANN is that it has the potential to simulate most functions between a set of attributes and their outputs, provided that it is appropriately structured and there is enough training data for the problem. The main disadvantage is that it has a larger number of tunable parameters affecting its performance than most ML techniques and is generally more difficult to train.
Two approaches were attempted here: training and testing the ANN with the entire dataset and training and testing after applying Random Under-sampling to the dataset first to balance the classes.

## III. Results Discussion and Possible Next Steps

The table below summarizes the performance score of the applied ML and DL models. The average F1-score for all models is very close except for the SVM with class_weights set to "balanced" and ANN coupled with Random Under-sampling. FF-ANN has the highest F1-score but with only 1% enhancement over RF. All models, except the RUS+FF_ANN, overclassified Class C on expense of the other classes, producing very low individual F1-scores for the under-represented classes, especially class A and E. Weighted_SVM performed slightly better in that aspect, but on the expense of Class C: F1-score of the other four classes increased slightly but that of Class C decreased. Since Class C is the majority class by a large margin, the model has a lower average F1-score than the rest of the models, excluding RUS+FF_ANN.

**Performance Summary of the different Classifiers**

| | **RF** | **Weighted_RF** | **KNN** | **SVM** | **Weighted_SVM** | **FF_ANN** | **RUS+FF_ANN** |
|---|---|---|---|---|---|---|---|
| **Av. Precision** | 0.62 | 0.61 | 0.59 | 0.62 | 0.68 | 0.61 | 0.30 |
| **Av. Recall** | 0.71 | 0.71 | 0.71 | 0.71 | 0.49 | 0.67 | 0.30 |
| **Av. F1** | 0.63 | 0.62 | 0.61 | 0.60 | 0.55 | 0.64 | 0.30 |

All models can be further improved slightly through parameter tuning: K can be tuned using the Elbow Method in KNN, C and gamma for RBF in SVM using GridSearch and cross-validation for tuning the number of epochs and batch size in ANN. However, it is not expected that the performance will improve significantly. This because the underlying problem of severe class imbalance. Random Under-sampling did not work well for this dataset because, given the large number of attributes, there are just not enough records to train the models appropriately for the under-represented classes, especially classes A and E.
Possible next steps can be:

- Gathering further training records for the under-represented classes, especially class A and E.
- Attempting a Hybrid Sampling technique on the data, where Class C is under-sampled and the other classes are oversampled. Oversampling can theoretically be applied directly on the given dataset, but due to the extreme class imbalance, will generate a huge dataset that can be difficult to work with. After first randomly under-sampling Class C, an Over-sampling technique from imblearn package can be used, such as SMOTEENN, which will generate new samples for the minority classes using a heuristic and also clean/remove noisy records