# BIO322 – Classification Project

## Prediction of odour based on molecular property

Written by :

- Changling Li 282440
- Alexander Popescu 286562

## Introduction

The goal of this project is to find the best machine learning method that predicts the smell, in particular, to predict whether an odour is perceived as more sour or sweet, based on some molecular properties of the substance in question. We will first explore our raw data and visualize it to give us an idea on which feature selections or engineering we could apply to our data before building our models. Then, we will apply some linear and non-linear methods. Finally, we will present our best results with both methods.

## Data exploration

For the data exploration, one R scripts was created called “exploration.R” . Different methods and plots were used to determine the nature of the data, which to some extent helped with the model selection.
The training data received for this classification task is highly dimensional but does not have many training samples (data measures about 700 x 4000 with expect for 2 mostly numerical columns). So it is clear that on the first look p > n. However, by inspecting the data closer, it can be noticed that there are many 0 columns. Since 0 columns but also constant columns do not contribute in any way to the model and in some cases can even cause problems with NA’s , they were simply removed from the data to reduce the dimensionality. In an attempt to figure out whether the dimensionality could be further reduced by removing correlating predictors, a heatmap of correlation (refer to plots)  was created. This visualized that there are quite a lot of predictors that correlate with each other and could be theoretically removed (one of the predictors would be removed).
By plotting the percentages of a sour and sweet sample in a barplot, it can be observed that the data is slightly unbalanced. There are more sour samples than there are sweet ones. To see if the data might form any clusters that correspond to sweet/sour characteristic, PCA, tSNE and K_means were performed. The plots (refer to plots directory to see the plots) showed that sweet and sour samples pretty much overlap and that no clear cluster could be detected.  By looking at the K_means plot, it could be speculated that there are multiple clusters (each being either sweet or sour) that overlap with each other. Concerning outliers removal, it does not seem very dramatic. Still, it could be a possibility to explore for model improvement.
Additionally, by using the Shapiro test it has also been shown that in general the predictors are not normally distributed, possibly indicating high skewness and therefore the use of log function as possible feature engineering.

## Results of Linear Method

### Logistic regression (src/Linear_method.R)

First, we built a simple logistic regression model trained on our raw data to have an AUC of reference (so we will know for our next models if we improved or not). We did 10-fold cross-validation where we calculated the mean of the 10 AUCs (that arise from the 10 folds). Here, we obtained an AUC of about 0.52. Then, from our results in the exploration part, the feature selections we applied was the removing of the constant predictors that do not contribute in any way to the model to improve the performance of our model. To achieve this, we removed the predictors were the variance was less than 1e-5. For further performance improvements, we also removed the predictors where the pair-wise absolute correlation cutoff => 0. 99. We chose this value to not lose too much information that could be useful to train our data, thus reducing the AUC. As expected, it did lower the AUC, but by 4 thousandth which is negligible.
As said previously in the exploration part, the predictors are not normally distributed. So we decided to check the skewness of every predictor and apply log function on the values on those who have high skewness ( > 1). This feature engineering improved the results but we still had to tune this parameter to find the skewness threshold that will improve our results. By doing 10-fold CV on each iteration on the skewness threshold from 0.70 to 1.00 with step 0.2, we found skewness > 0.7. We did the tuning in the file Linear_method_skewness.R.
We then later tried to add another feature engineering which was simply scaling the data, but it didn’t change the results. We also didn’t try to apply One-hot encoding because the categorical predictors had only 2 categories. Then, one needs to apply a subset selection to find the relevant predictors. But since we still had 1826 predictors, it may be computationally expensive. We decided to apply another alternative that is less computationally expensive:
Lasso regularization. In that case, we needed to find the best lambda value that reduces the MSE. To tune this hyperparameter, we applied a 10-fold CV. The best lambda we found was 0.01870272.

Finally, here is our best linear model: by removing the constant and correlated predictors, apply log function on the predictors that have a skewness > 0.7 and apply Lasso regularization, we obtained our best model with **AUC = 0.589**.

## Results of Non-Linear Methods

### Random Forests (src/random_forests.R)

For each one of different random forests models, we did 10-fold CV to assess the AUC. First, we removed the constant predictors that have variance < 1e-5. Then, we also removed the correlated predictors with cutoff = 0.99 as we saw in the exploration part that there was a lot of correlation between predictors. Hopefully, it did improve our results. Finally, we also scaled our data as it is known that machine learning algorithms perform better when numerical input variables are scaled to a standard range. Again, it also improved our model.
For further improvements of our random forests, we tried to tune the hyper-parameters mtry, maxnodes and ntree to improve our model and thus our results. We found mtry = 1, maxnodes = 7 and ntree = 125. You can find the code used for tuning these hyper-parameters in the file RF_tuning.R. Finally, our best model is built by applying the mentioned feature selections and engineerings and these hyperparameters with AUC = 0.62. It is with random_forests_best.R that we produced the predictions first_try_RF.csv submitted on Kaggle. Unfortunately, we couldn't reproduce the same results due to bad seed setting.

### Neural net (src/NN3.R)

This method was considered because of the high accuracy it can achieve if enough data is available. We chose 3 dense layers to allow for more subtle classification because the performed PCA indicated very difficult data (no clusters). The model was completed with 2 dropout layers, L2 regularization and a normalisation layer at the end tail of the Network. In reference to the exact number of nodes and the regularization parameters, we used the Bayes optimization algorithm (with CV), which can be found under "src/Bayes_optimisationNN". For the nodes, we chose a range of 5 to 50 per layer, since a more flexible method (even with higher regularization) did not perform better and the algorithm would work much faster this way. The ranges for the other Hyperparameters were also more or less chosen by trial and error. Note that the algorithm takes a long time (1 day for 100 iter.)
We tried many possibilities (too many for this report to mention all) to improve our NN and even tried using other activation and loss function (swish and focal loss) to adjust for the difficulty and unbalanced data respectively. However, due to the little data available (NN requires lots of data to work efficiently), the AUC would not be greater than **0.62** (estimated with CV). Nevertheless, one thing that could be improved ist he features engineering regarding the adjustment for skewness in our data. One approach was to logarithmize the data, which unfortunately did not work in NN because of a bug. Concerning the unbalanced data, class weights and the focal loss function were used. The latter did not significantly improve the AUC significantly or not at all, as one hand the weights and biases needed to be initialised differently and on the other hand, it is meant for data that is more highly unbalanced.
Unfortunately, there is no reliable Kaggle submission that was submitted in time due to a bug in data scaling and seed setting. The final NN file is under src/NN3. There the model that was found with the Bayesian optimisation algorithm would be implemented to be used on the test data set.

### Support vector machine (src/SVM.R)

This method was considered because it works very well with high dimensional data and does not require as much data as NN does. In a nutshell, data points are "transformed" to higher-dimensional space, using different kernel functions, and are then categorized using hyperplanes. Here we used again the Bayes optimisation algorithm (with defaulted range for the hyperparameters) on an RBF-SVM model (SVM that uses radial based kernel function). After 70 iterations of the algorithm, we got a model that had an AUC of **0.65**. Unfortunately, there is no reliable Kaggle submission (that was submitted in time at least), since there was a scaling bug in our code. The final SVM file is under src/SVM where the optimized model would be implemented

## Conclusion

Regarding AUC, SVM would be the best model followed very closely by Neuralnet and Random forest. After tuning SVM, an AUC of 0.65 (with CV) was achieved. Unfortunately, due to a very well hidden bug, concerning the scaling of the test data we were not able to submit reliable results with SVM and NN to Kaggle in time. For the future, it would be interesting to play around with different kernel functions of SVM. Concerning the NN, there would be much more room for improvement if more data was available. The low AUC reflects mostly the lack of data available but also the difficulty of the data to some extent. Even so there are still ideas for improvement in the room. For example, one could use a more sophisticated tuning algorithm like a genetic algorithm, which would work quite well considering that there are so many Hyperparameters for NN. This could also be combined with multithreaded cloud computing, as genetic algorithms are known to be quite costly. Both of those ideas were partially tested but failed because the libraries (CloudML and GA) are new and «buggy». For the linear and random forest model or all the models in general, we could have played more around with feature engineering. For example, we tried to logarithmize our data, because a lot of columns of the data seemed to be skewed. However, there are other transformations such as Yeo-Johanson transformation, that could be used. There is even a library (bestNormalise) that compares different transformation with respect to their fit. Also for the next machine learning, it would be nice to better benchmark the different models. It would improve the reproducibilty of our project.