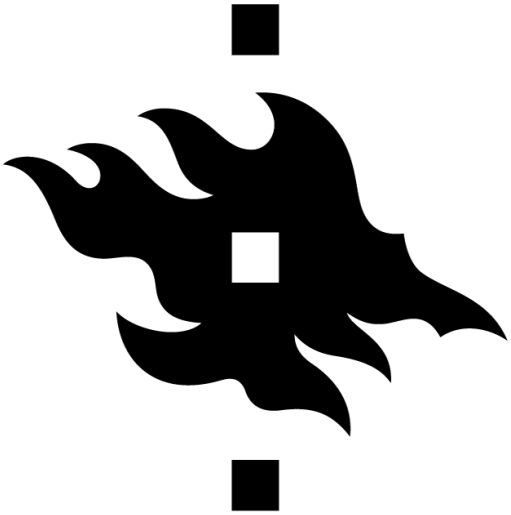

New Particle Formation

Introduction to Machine Learning project

University of Helsinki

Faculty of Science

Group 108

Enrico Buratto

Eoin Doyle

Fran Jurinec

Academic Year 2021-2022

Contents

1 Introduction

The goal of this project is to apply one or more classifiers on a New Particle Formation (NPF)[1] test dataset in order to predict the event types for days listed in the set. To put it shortly, we can say that an NPF is the phenomenon of a big particle forming from other small particles under some conditions. The dataset is composed of different day measurements of particles concentration in the air around Hyytiälä Forestry Field Station; furthermore, for each sample of the set a label indicates if that day the NPF phenomenon happened or not.

This task can be divided into two sub-tasks, which are respectively to build a binary classifier, that inferes an *event/nonevent* prediction, and a multi-class classifier that predicts four classes: *Ia, Ib, II, nonevent*.

This document, which also reflects our approach to the problem, is structured as follows: first of all, we describe the data exploration we performed in order to find recurring patterns and hints that could help us find a suitable model and, in general, a path to follow. After that, we discuss the different models we have taken into consideration and we used, and the feature selection we performed on the data. Finally, we present the results we were able to achieve and we discuss them.

The entire project has been developed in Python with Jupyter Notebooks; the most important libraries we used are Numpy and Pandas for data management, Scikit-learn for Machine Learning models and Matplotlib and Seaborn for useful graphs and plots.

2 Data

2.1 Preprocessing

The dataset was initially composed by 458 rows and 104 columns: each row corresponds to a day in which measurements for several atmospheric properties are performed, while the columns are divided as follows:

- One column contains the dates of the measurements;
- One column contains an unique id;
- One column contains a boolean condition on the measurements (`partlybad`);
- 50 columns contain mean values for several measurements, mostly the average concentration of different specific molecules at different altitudes;
- 50 columns contain the standard deviation of the associated averages;
- One column contains the class, *i.e. Ia, Ib, II, nonevent*.

After this really preliminary data analysis, we performed some simple data preprocessing in order to have a coherent and usable dataset. First of all, we dropped the following columns:

- `id`: since the dataset has been loaded in a Pandas Dataframe, this field is useless;
- `date`: as discussed below, we decided to not use any time-related variable for our predictions;

- `partlybad`: the values in this columns are all the same (*i.e.* `False`), hence this column was useless.

We then separated the training set into two different Dataframes, namely one for the features and one for the target variable. Since we started with the binary classification, initially the target variable Dataframe was composed only by *event*/*nonevent* values; later in the project, we also considered the four-class classification.

2.2 Exploratory Data Analysis

After having cleaned the data we started analysing the dataset. Some first useful analysis has consisted in plotting a *pairplot* of the entire dataset, in order to find out how the data was distributed among classes; an example of pairplot for the binary classification scenario is Figure 1.

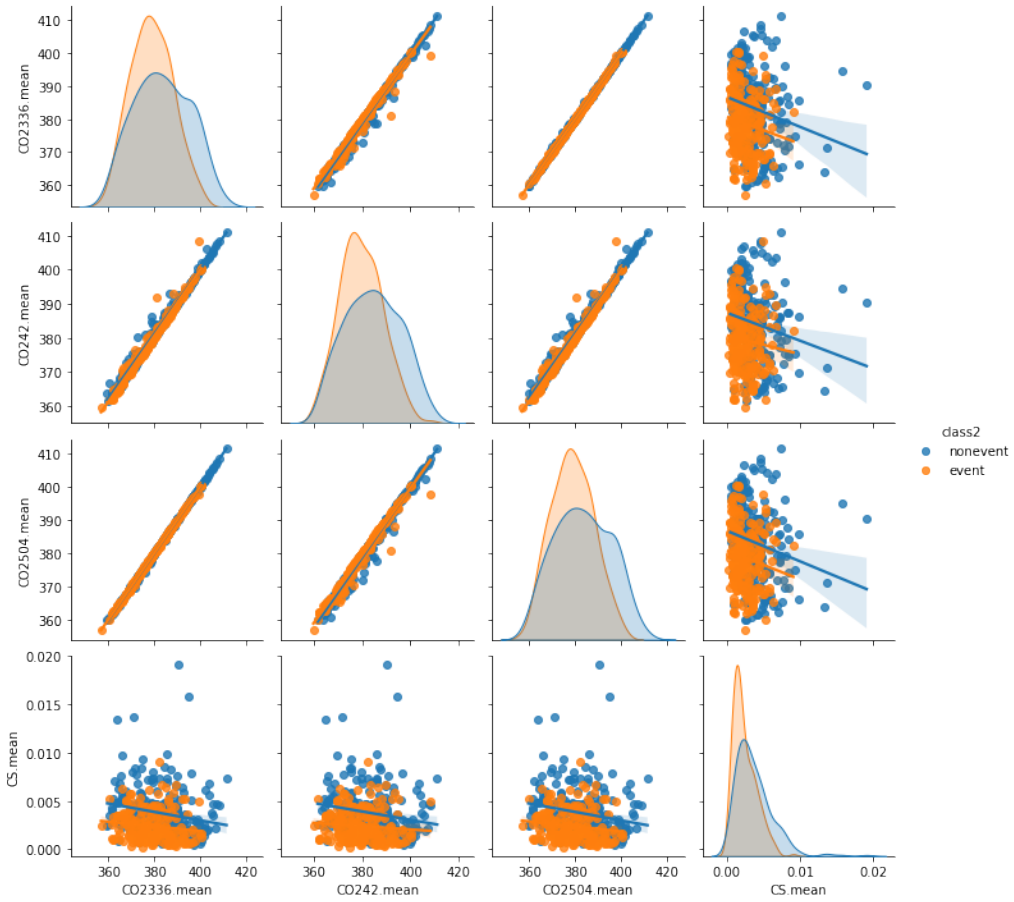

Figure 1: Pairplot of some mean features, binary classification.

Another useful insight on the data came from the analysis of correlation between features; to be more specific, we analyzed the correlations between the mean features, and we plotted them into the heatmap in Figure 2. From this heatmap it appears clear that some variables are really correlated: the lighter boxes, in fact, shows that same molecules at different altitudes are strongly similar between them. This consideration could have been useful in later feature selection.

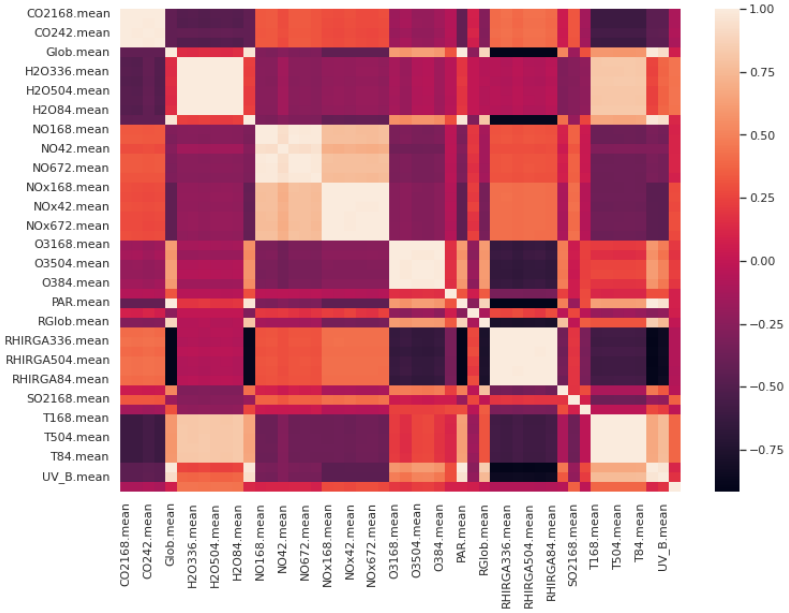

Figure 2: Heatmap of mean features.

Another technique we adopted was to apply unsupervised learning to the data. In particular, we applied KMeans clustering with a different number of clusters, from 1 to 20; plotting the total loss in function of the number of clusters (Figure 3) allowed us to see even more that only a small partition of features is really relevant for the classification task.

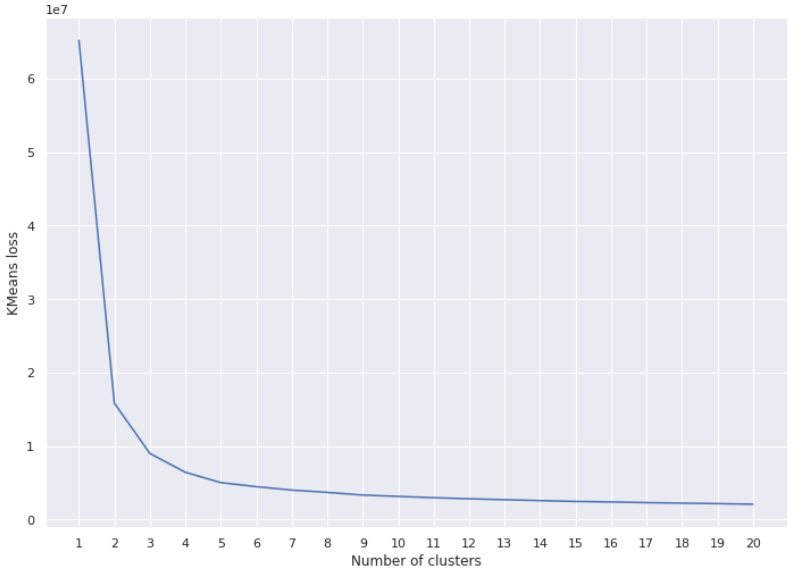

Figure 3: Total KMeans loss in function of number of clusters.

This hypothesis is confirmed also by the percentage of variance explained obtained applying Principal Component Analysis (PCA) on the dataset; as the reader may notice from Figure 4, in fact, most part of the variance on not normalized data is explained by the first three features.

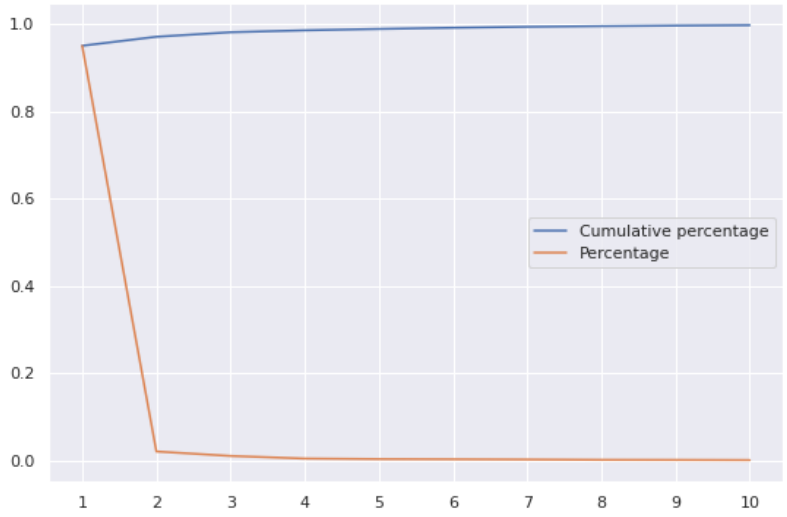

Figure 4: Percentage and cumulative percentage of variance explained on unnormalized data.

In order to see if the cumulative percentage was too high on the first few features because of data being too noisy, we also scaled the features to zero mean and unit variance; the result we got is reported in Figure 5.

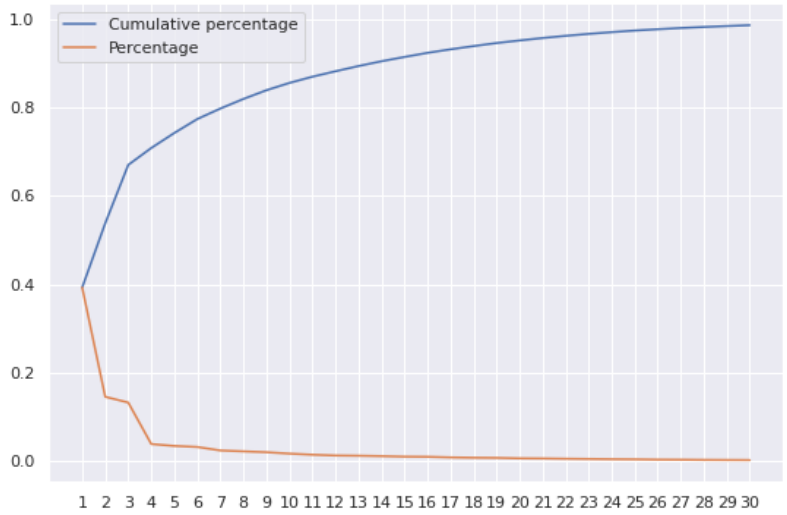

Figure 5: Percentage and cumulative percentage of variance explained on normalized data.

As the reader can see, the number of features it needs to achieve a cumulative percentage near to 100% is now 30 on normalized data: this means that some variable had a great variance that busted the results. However, even with normalized data, it is pretty clear that only a restricted number of features are really meaningful for the prediction task.

2.3 Considerations

Some considerations could be made on data before proceeding with model and feature selection. First of all, it is evident that the number of samples is very small in relation to the number of features: a hundred is, in fact, a huge number of variables when the data samples are less than five hundred. Even if we count only the means, without the standard deviations, we get an amount of fifty features, that are still a lot. This led to the idea of performing some sort of feature selection an dimensionality reduction on the data.

Another discussion we had was whether to keep the date or not as a feature for the classification task: this variable, in fact, could have been taken into consideration after some transformation, if for example there was seasonal variation in the frequency of events, or an increased probability of events after several nonevent days. However, after trying the first models we noticed that the performance measurements basically did not change. In addition to this, keeping the date would have transformed the approach from a relatively simple classification task to time-series analysis; since this is not in-scope of the project, we finally decided to not use this feature.

With these considerations in mind, we proceeded in analysing different classification models.

3 Model selection

3.1 Performance metrics

In order to measure the model performance we used different metrics on which we based all the work that follows; these are **accuracy** and **perplexity**.

Accuracy This metric is defined as the number of correct predictions divided by the total number of predictions, *i.e.*

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

We measured accuracy both for the binary classification and the for the 4-class classification tasks.

Perplexity This is a metric that defines how much "perplexed" to see a certain value is a model. It is defined as a weighted geometric average of the inverses of the probabilities, *i.e.*

$$Perplexity = e^{-\frac{1}{N} \cdot \sum_{i=1}^{N} l(\hat{y}_i, y_i)}$$

where $l(\hat{y}_i, y_i)$ is the logarithmic loss of the predicted value and the real value. We measured perplexity both for the binary classification and the for the 4-class classification tasks; in the last case, we calculated it on the 4-class model but only for the binary classification.

We performed **cross-validation** and we calculated the accuracy and the perplexity on *k-fold* cross-validation with different k, typically 5 or 10. In order to do this we used Sklearn's `GridSearchCV`[3], that enabled us to build a pipeline that could also sort the different model performances in order of accuracy or perplexity; the small excerpt of code in Listing 1 shows an example of pipeline with different hyperparameters in a Random Forest classifier.

```
p = GridSearchCV(
estimator=Pipeline([
    ("scale", StandardScaler()),
    ("model", RandomForestClassifier(
        max_depth=10,
        max_features=5,
        min_samples_leaf=3,
        min_samples_split=8,
        n_estimators=170,
        random_state=42))
    ]),
param_grid={"model__n_estimators":[n for n in range(99,400,20)]},
scoring={
        "perplexity":make_scorer(perplexity, needs_proba=True),
        "accuracy":make_scorer(accuracy_score)
        },
refit="perplexity",
cv=5
)
```

Listing 1: GridSearchCV example

3.2 Baseline

Coming to the model selection, we decided to try a bunch of different selected classification models to find out what to expect from the classification task and to fix a benchmark that we could use to measure the more in-depth model analysis progress. In order to do that, we decided to adopt an Automated Machine Learning process using the TPOT library[2]. This library is a Python AutoML tool that optimizes machine learning pipelines; it can use a configuration file containing all the pipelines it should try, and then it tries them for several generations and population sizes.

We did not used this library in a "total blackbox" fashion; in fact, we properly configured the library with models we have already seen during our careers. The excerpt of code in Listing 2 shows the TPOT configuration we used.

It is important to remind that this approach could not be considered as definitive, since it is *data agnostic*: we used it only to simplify the procedure of trying different models with different hyperparameters in the beginning of the project.

```
tpot_config = {
    'sklearn.linear_model.LogisticRegression': {
        'penalty': ['l1', 'l2', 'elasticnet', 'none'],
        'C': [1e-2, 1e-1, 1, 1e2]
    },
    'sklearn.neighbors.KNeighborsClassifier': {
        'n_neighbors': [1, 2, 3, 4, 5, 10, 20],
        'leaf_size': [20, 30, 50, 60, 100],
        'p': [1, 2],
        'num_jobs': [16]
    },
    'sklearn.svm.SVC': {
        'C': [0.5, 1, 2, 5, 10],
        'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
        'degree': [1, 2, 3, 4, 5],
        'gamma': ['scale', 'auto']
```

```
    },
    'sklearn.naive_bayes.GaussianNB': {
        'var_smoothing': [1e-10, 1e-9, 1e-8]
    },
    'sklearn.tree.DecisionTreeClassifier': {
        'criterion': ['gini', 'entropy']
    },
    'sklearn.ensemble.RandomForestClassifier': {
        'n_estimators': [50, 100, 200, 500],
        'criterion': ['gini', 'entropy']
    }
}
```

Listing 2: TPOT configuration

3.3 Chosen models

The TPOT execution showed us that Random Forest could be a possible model for the binary classification task; the cross-validation accuracies we got were, in fact, higher than the other models. We then decided to use this model for further investigation, along with a simple Logistic Regression that we kept for comparison; a brief summary of these two models follows.

Logistic regression This is one of the simplest models among the classifier algorithms class. It uses a logistic function, *i.e.* a sigmoid that models probabilities, to model a binary dependent variable. Usually this model is used for binary classification, but it could be extended to a multiple classes scenario assigning a probability to each class so that the total probability (the sum of class probabilities) is equal to one. We used this model because its results are really easy to interpret; however, although it works good with binary classification, we were not able to achieve the same accuracy scores on the 4-class classification task.

Random Forest This is one of the most powerful classification algorithms. It is an ensemble method that rely on the construction of different decision trees during the training phase; for classification tasks, the predicted class is the one selected by the majority of trees. Although this is a really powerful algorithm, its results are usually difficult to interpret; that is why we decided to use also Logistic Regression along with it.

3.4 Hyperparameter tuning

After fixing which models to use, we tried to do some hyperparameter tuning on the models in order to see if we were able to achieve higher accuracy and/or lower perplexity. We proceeded both with the TPOT library, that helped us trying different parameters on the same model, and manually changing them.
For Random Forest, we tuned the following parameters:

- `max_depth`: maximum depth of the trees;
- `max_features`: the number of features to consider when looking for the best split;
- `min_samples_leaf`: the minimum number of samples required to be at a leaf node;
- `min_samples_split`: the minimum number of samples required to split an internal node;

- `n_estimators`: the numbers of trees in the forest.

Since Logistic Regression was not our designated model, we tweaked only two parameters:

- `max_iter`: maximum number of iterations of the algorithm, in order to let it converge in every scenario;
- `C`: inverse of regularization strength, *i.e.* a value for which smaller values mean stronger regularization.

4 Feature selection

As already stated in this document, the training dataset has a small amount of samples and a relatively large set of features; consequently, some dimensionality reduction could be useful to improve the performance of the classification model. We tried different methods to increase the number of samples and/or decrease the number of features; these techniques are described in this section.

4.1 Artificial samples

A first technique we tried was to halve the number of features, removing the standard deviation columns. Along with this, we tried to expand the number of samples in the training set combining mean and standard deviation for each variable into a normal distribution, creating then a large amount of artificial samples. The three functions at Listing 3 does that.

```
def pad_data_column(col,n):
    v=np.column_stack([col]*n)
    v=v.flatten()
    return v

def generate_Gaussian_sample(m,s,n):
    samples=[]
    for i in range(n):
        samples.append(np.random.normal(loc=m,scale=s))
    return np.array(samples).flatten(order='F')

def pad_npf(df,n):
    padded=pd.DataFrame()
    for name in df.columns:
        if name == "class4":
            padded["class4"]=pad_data_column(df["class4"],n)
        if name[-5:] == ".mean":
            padded[name] = generate_Gaussian_sample(
                df[name],
                df[name[:-5]+".std"],
                n)
    return padded
```

Listing 3: Artificial samples building

Unfortunately, we didn't get the desired results: both accuracy and perplexity scores were, in fact, more or less the same; this is probably due to the fact that both Logistic Regression and Random Forest classifiers ignored the standard deviations or, at least, use them in the right way, building a loss function very similar to the one built with the artificial data.

4.2 Feature Cherrypicking

Another technique that was attempted was plotting every feature separated by class and manually observing which features show the highest distinction between classes. This may not be a very ”professional” method however it yielded fairly good results. Additionally, features which were nearly identical were reduced by only picking one from the near-identical group.

In the end, the chosen cherry-picked features were the following: ’UV_A’, ’T84’, ’RPAR’, ’RHIRGA42’, ’RGlob’, ’PAR’, ’PTG’, ’O3672’, ’NOx504’, ’NET’, ’H2O42’ and ’CO2504’.

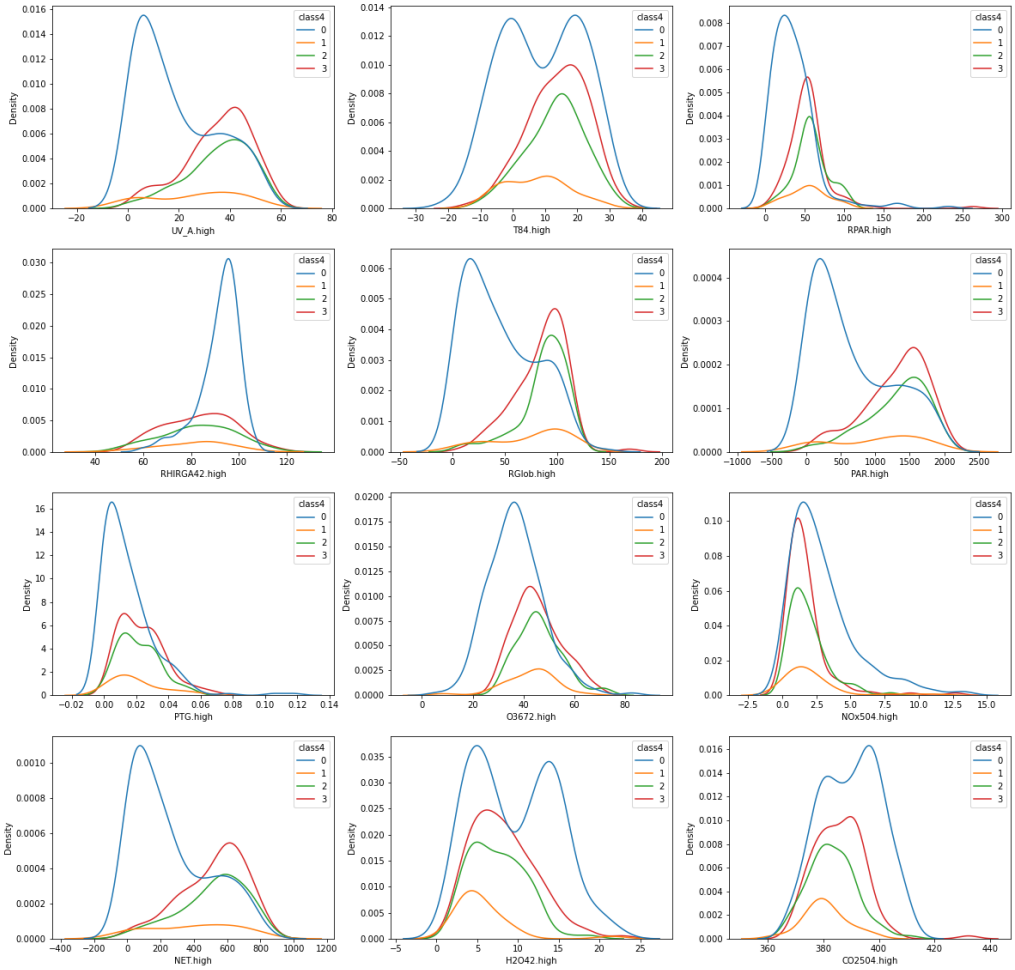

Figure 6: Cherry picked features (mean + 2sd).

By only training our final model only on these features, the classification accuracy and perplexity on both class4 and event vs. non-event classification tasks were impacted minimally. Resulting perplexity was within 0.1 difference and accuracy within 0.01 difference from using all features.

4.3 Automated PCA

We also tried to apply Principal Component Analysis in an automated fashion: Sklearn’s `PCA` class and related functions, in fact, automatically select the best n variables, *i.e.* the

features that, in order, explain more percentage of variance.
We first applied PCA on our pipeline trying different numbers of selected features on unnormalized data: we started halving them and then down until 5. Secondly, we normalized the data scaling it to zero mean and unit variance, using Sklearn's standard scaler, and we proceeded in the same way. Unfortunately, these techniques didn't improve significantly our performance measures on the classifiers; more details about these results can be found in Section 5.

5 Results

5.1 Challenge

We participated in the mid-project challenge using the Random Forest classifier, which we considered was the best at the moment. The accuracy score we reported has been calculated using a *10-fold* cross-validation; the predictions were made using the model in Listing 4, that we got after several trial and error.

```
RandomForestClassifier(
    max_features=12,
    min_samples_leaf=3,
    min_samples_split=12,
    n_estimators=170
    )
```

Listing 4: Random Forest hyperparameters

This model creates a Random Forest with 170 different trees; Figure 7 is an example of them. Note: yellow stands for `nonevent` class, green for `Ia`, blue for `Ib` and pink for `II`.

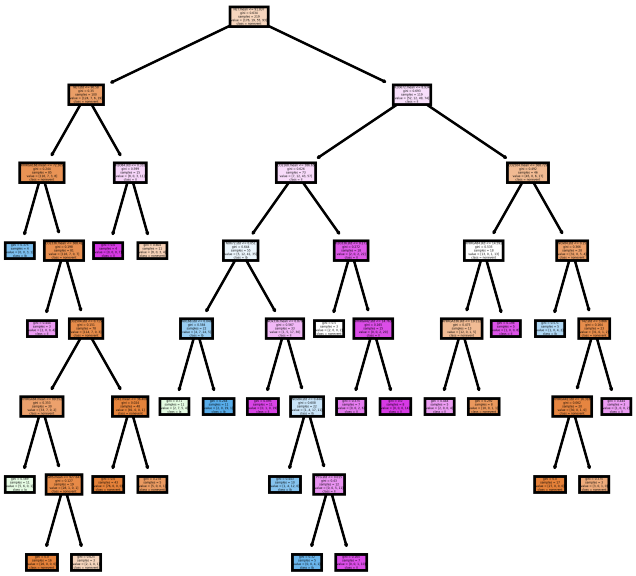

Figure 7: First decision tree in the forest.

The final results we got for the challenge with this model are:

- **Binary classification accuracy**: 0.852, that we reported decreased by 0.05 to have a little slack;
- **4-class classification accuracy**: 0.65;
- **Perplexity on `nonevent`**: 1.41.

5.2 Final results

As said in Section 4, we didn't get the desired results using PCA: the performance scores were, in fact, only slightly higher than the results we got on raw data. These results are also shown in Figure 8: these graphs represent the 4-class classification accuracy on function with the number of components in PCA. The image should be read in horizontal order: in the first row, the first graph refers to logistic regression without normalization and the second with normalization; the second row is the same but for Random Forest.

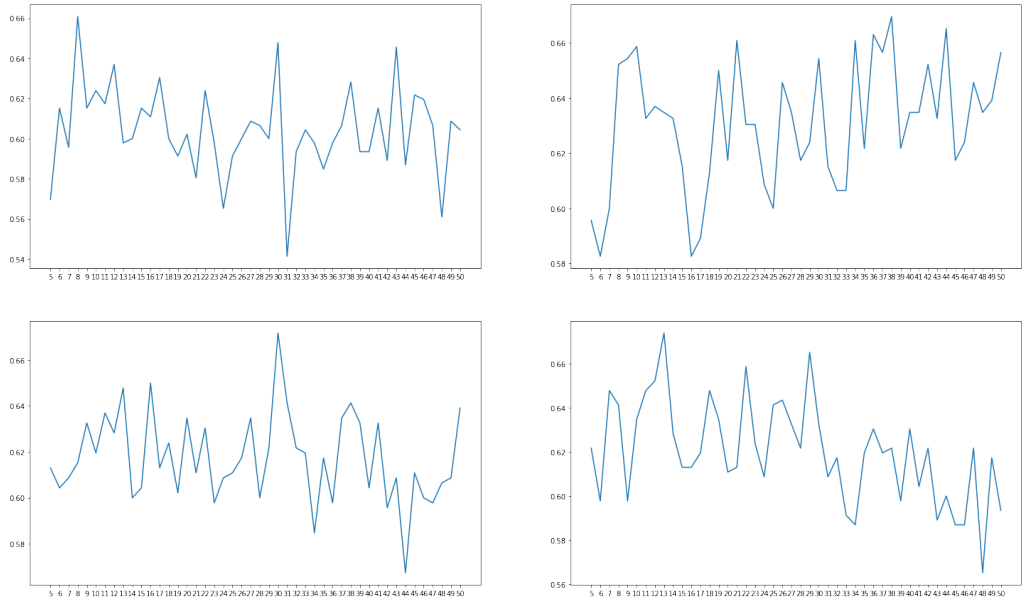

Figure 8: Accuracy vs PCA components

As the reader may notice from the previous figure and from Table 1, we actually got some improvements in accuracy. However, the maximum accuracy gain we got was less than 2%, and this is not enough to say that the models are actually better; in fact, this little fluctuation is most likely due to the random parts of the algorithms, and thus cannot be trusted.

	Without normalization	**With normalization**
Logistic Regression	components=8, acc=0.66	components=38, acc=0.65
Random Forest	components=30, acc=0.67	component=13, acc=0.67

Table 1: Performance results with PCA

6 Conclusion

From the investigation of the data as well as the results we conclude that the source data is not sufficient to accurately predict a full classification of events into all four categories. Even the limited classification of events vs. non-events only reaches 85%. This is still not an optimal classification accuracy and would not be suitable for systems in production which rely on accurate event type information. Observing these accuracy results and the research taken across various classification models, we conclude that better data is necessary for accurate predictions.

We additionally concluded that the Random Forest model was a well-suited classification model for this data as it produced consistently good results irrespective of data normalization or limited feature selection. If anything, it showed the best results when trained on the complete feature set, indication that it would likely scale well for larger numbers of new features, and would pick up on any class-indicative features without the need for pre-processing the data for either training or classification.

7 Discussion

With the data as it is, we are satisfied with the results we achieved. The reported Random Forest model can be trained on raw measurement data with no normalization without any degradation in performance. Our previous research did reveal that other models such as logistic regression performed better on a normalized set of limited features, however we have outperformed those results using un-normalized data with full features. In hindsight there may have been a better result possible through the use of normalized data and limited features through the use of some other models with better tuned parameters.

7.1 Self-grading

7.1.1 Deliverables Grading

The overall grade we would like to give to our deliverables is a 4/5.

We believe that we have presented an in-depth analysis of the data as well as a solid final model based on what we learned from our research. We analysed a wide range of classification models and feature selection processes and documented the decisions and conclusions we reached from our analysis and trials.

The short pitch presentation presented a general overview of the project. Perhaps it did not cover as many details as it could, mostly due to the limited duration time, however we do believe some presentation elements could have been improved such as presenting more concrete data as opposed to narrative about our processes. The visuals of the presentation could also use some more work if we wanted to make it presentable for the general audience.

The final challenge results achieved performed at a significantly above-average level. Therefore we believe that no points should be deduced in this area. We are satisfied with the overall result and the final results was reached through consistent group teamwork.

Finally, the final report presents a fairly in-depth overview of our process, thinking and conclusions. It covers the final model and how we reached it as well as general conclusions about the data and the problem at hand. We believe that some parts could have been presented better such as the evaluation of various other models which we have tested internally. A portion of our model testing results were never systematically recorded, so

they never made it into the report. We should have been more consistent with recording the various trials and errors we had along the way as they could have provided additional insight into our decisions.

7.1.2 Group Grading

Regarding the **group as a whole**, we think that our group deserves a full **5/5**: in fact, we collaborated in keeping the environment clean and performing, and in our opinion we worked good together. We were able to maintain our goals on meetings: in fact, we met regularly on Google Meet in order to update the others on the work we did, to divide the subsequent tasks and to reason together on the problems. These meetings lasted a reasonable amount of time, enough to do what we had to do and not so much to get bored or tired; during these appointments every component of the group brought his personal experiences on previous courses and experiences that could be useful for our work, and we discussed about them taking into account everyone's ideas. Moreover, we can say that our group also helped us in understanding and expand individual knowledge.

References

[1] Kerminen et al., "Atmospheric new particle formation and growth : review of field observations" , Environmental Research Letters, vol.13, no.10, 103003. https://doi.org/10.1088/1748-9326/aadf3c.

[2] Weixuan Fu, Randy Olson, Nathan, Grishma Jena, PGijsbers, Tom Augspurger, Joe Romano, PRONOjit Saha, Sahil Shah, Sebastian Raschka, sohnam, DanKoretsky, kadarakos, Jaimecclin, bartdp1, Geoffrey Bradway, Jose Ortiz, Jorijn Jacko Smit, Jan-Hendrik Menke, Randy Carnevale. (2020). EpistasisLab/tpot: v0.11.5 (v0.11.5). Zenodo. https://doi.org/10.5281/zenodo.3872281.

[3] GridSerchCV, https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV