Term project report

Toni Rämö

Project information

- **Team name** : Group 110
- **Team members** : Toni Rämö
- Coding language: Python 3
- Used libraries:
  - Machine learning, preprocessing, pipelines: Scikit-learn (sklearn)
  - Data processing: Pandas, Numpy
  - Visualization: Matplotlib, Seaborn, stasmodels (Q-Q plot)
  - Hyperparameter tuning: Optuna
- Platforms:
  - Analysis: Jupyter Notebook (on VSCode)
  - Report: R Notebook
- Version control: Git
- Presentation (15.12.2023): slides available *on Slack*

Data analysis

Data analysis consists of the following stages.

Initial observations

Our first priority is to familiarize ourselves with the data: read the descriptions of features from the instructions but also look at the data. We consider following: How many features do we have? What are their data types? Are there unnecessary features? What transformations are necessary? What features should be included in X and which should be part of target y?

We start by reading the training data with `pandas.to_csv` to `df_train`. `pandas.DataFrame.shape` tells us that there are 27 columns (features) in the data and 27147 rows. Initial column types are accessed with `pandas.DataFrame.dtypes`. Types `int64`, `float64` and `object` imply integer, decimal or categorical values, respectively. `MW` and `pSat_Pa` are decimal, `parentspecies` categorical and all others contain integers.

We can get summary statistics of numerical features with `pandas.DataFrame.describe`. We omit the result from the report since further processing of the data is still needed.

So far, we know about the columns that:

- `Id` is clearly unnecessary feature for training as it's just a unique identifier of each molecule. This can be omitted from the training data.
- y should be base 10 logarithm of `pSat_Pa`.
- The remaining features can be included in X.

- `parenspecies` has ambiguous and missing values ('None')

For the advanced version (ADV), we also need to include *topographical fingerprints.* These are stored in `npy` files, which can be read with `numpy.load` method. Reading the training fingerprints, results in 2D array of shape `(27147, 8192)`. Each row of the array contains 8192 binary values. These cannot be used as such but instead e.g. if rows are treated as vectors, their lengths could potentially form a new, useful feature.

Next, based on the initial observations, we continue to perform initial pre-processing.

Initial pre-processing

Let us extract X and y based on observations above. We compute y by applying `numpy.log10` to column `pSat_Pa`. For initial version of X we choose all other columns but `Id` and `pSat_Pa`.

Next, let us consider `parentspecies`. Since it has missing values, we need to either impute them or drop the rows with missing values. In total, there are 206 rows with missing values corresponding to less than 1% of the total number of rows. We have an option to drop such rows or impute missing values. For simplicity, we start by dropping the rows with missing values (replace `None` with `numpy.nan` and perform `pandas.dropna(axis=0)`). After this, we can address categorical values of `parentspecies` by performing one-hot encoding. We use `pandas.DataFrame.get_dummies` with parameter `drop_first=True` to ignore the first resulting, redundant feature. This encodes `parentspecies` to `parentspecies_apin`, `parentspacies_decane` `parentspecies_toluene`. However, since `parentspecies` contains ambiguous values like `apin_decane_toluene` we end up with up to four additional columns such as `parentspecies_apin_decane_toluene`. To avoid this, we can encode ambiguous values to the columns mentioned above by splitting 1 between the relevant columns. For instance,

parenspecies_apin_decane_toluene
1

becomes

parenspecies_apin	parenspecies_decane	parenspecies_toluene
1/3	1/3	1/3

This is achieved with the following code. We will refer to this step as `handle_parentspecies`.

```
def handle_parenspecies(X):
  combinations = [['apin', 'decane', 'toluene'],
                  ['apin', 'decane'],
                  ['apin', 'toluene'],
                  ['decane', 'toluene']]
  for combination in combinations:
    column_to_omit = 'parentspecies_' + '_'.join(combination)
    for species in combination:
      column_to_update = f'parentspecies_{element}'
      # Ensure all needed columns exist
      if column_to_update not in X.columns:
        X[column_to_update] = 0
      mask = X[column_to_omit] == 1
      X.loc[mask, column_to_update] = 1/len(combination)
    X.drop(column_to_omit, axis=1, inplace=True)
```

Unfortunately, while writing this report, I noticed an error on the line 13 of the code above that resulted in a wrong mask. The code above is a fixed version. I did not have time to retest all of the earlier tried pipelines (which naturally I would have done in an actual research project) but based on the cross-validation and the late Kaggle submission of the final solution (late submission are computed but not accepted to leaderboard), introduced fix did not have an affect to the score. However, it is still possible that some better solutions were missed due to this mistake.

For the ADV, let us add feature `norm_top` which represents L^2-norm of topological fingerprints. It can be computed with `np.linalg.norm` with arguments `axis=1` (for individual rows) and `ord=2` (order 2). We will omit this feature for the main competition.

Exploring the data

Now that we have performed initial pre-processing of the data, let us observe the summary statistics.

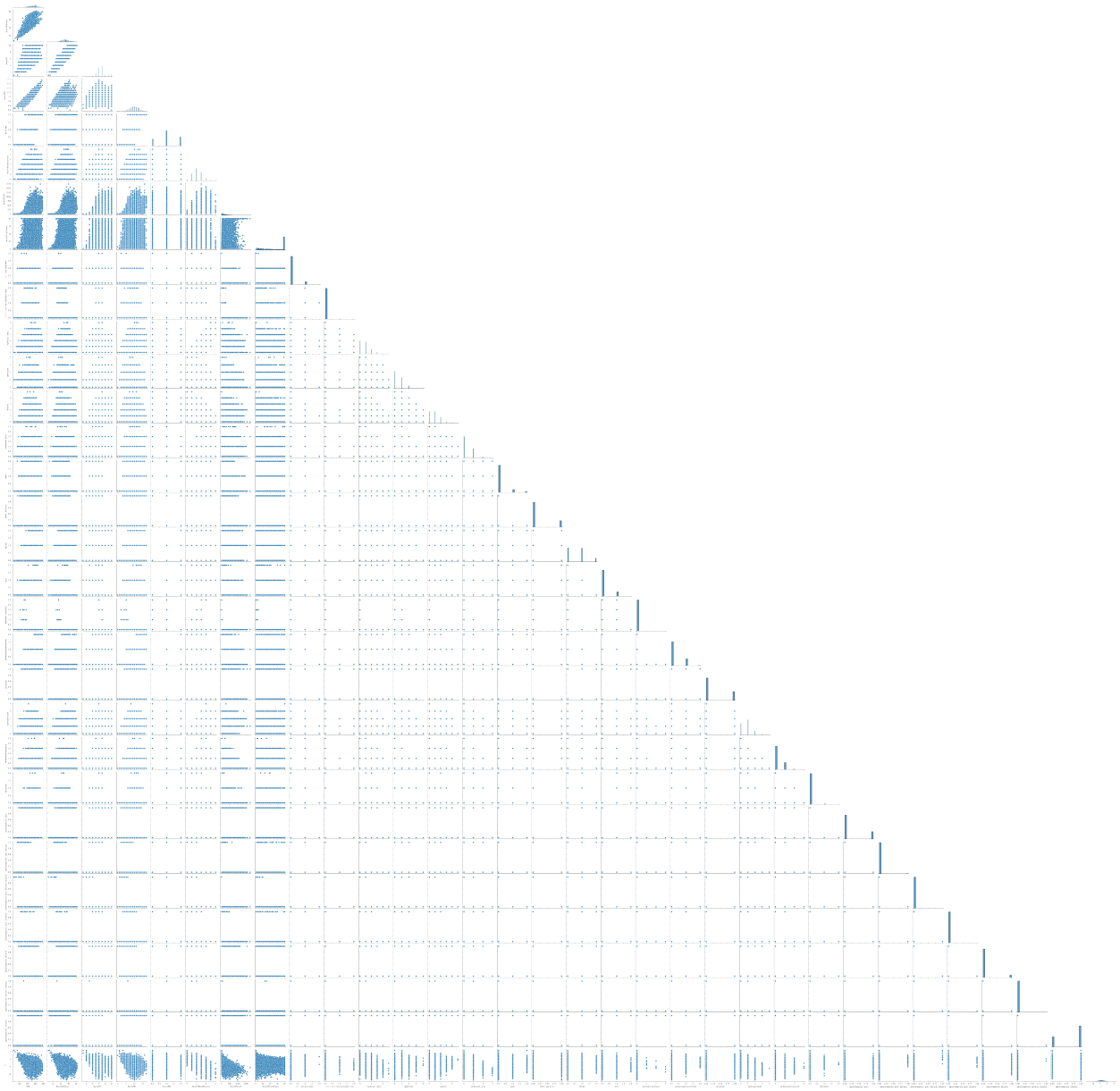
X	mean	std	min	25%	50%	75%	max
MW	264.56	49.89	30.01	232.98	266.99	299.01	386.04
NumOfAtoms	26.26	5.25	4.00	23.00	26.00	30.00	41.00
NumOfC	6.86	1.46	1.00	6.00	7.00	7.00	10.00
NumOfO	9.93	2.50	0.00	8.00	10.00	12.00	17.00
NumOfN	1.06	0.71	0.00	1.00	1.00	2.00	2.00
NumHBondDonors	2.20	1.02	0.00	2.00	2.00	3.00	6.00
NumOfConf	230.36	203.00	1.00	73.00	174.00	332.00	1743.00
NumOfConfUsed	25.80	14.66	1.00	11.00	30.00	40.00	40.00
C.C..non.aromatic.	0.09	0.29	0.00	0.00	0.00	0.00	2.00
C.C.C.O.in.non.aromatic.ring	0.01	0.13	0.00	0.00	0.00	0.00	2.00
hydroxyl..alkyl.	0.82	0.87	0.00	0.00	1.00	1.00	5.00
aldehyde	0.54	0.68	0.00	0.00	0.00	1.00	4.00
ketone	0.93	0.90	0.00	0.00	1.00	1.00	5.00
carboxylic.acid	0.34	0.53	0.00	0.00	0.00	1.00	3.00
ester	0.16	0.44	0.00	0.00	0.00	0.00	2.00
ether..alicyclic.	0.21	0.40	0.00	0.00	0.00	0.00	1.00
nitrate	0.67	0.67	0.00	0.00	1.00	1.00	2.00
nitro	0.15	0.36	0.00	0.00	0.00	0.00	2.00
aromatic.hydroxyl	0.00	0.05	0.00	0.00	0.00	0.00	3.00
carbonylperoxynitrate	0.24	0.45	0.00	0.00	0.00	0.00	2.00
peroxide	0.28	0.45	0.00	0.00	0.00	1.00	1.00
hydroperoxide	0.77	0.70	0.00	0.00	1.00	1.00	4.00
carbonylperoxyacid	0.26	0.47	0.00	0.00	0.00	0.00	3.00
nitroester	0.01	0.11	0.00	0.00	0.00	0.00	2.00
norm_top	58.00	15.95	2.45	46.89	59.66	70.18	89.78
parentspecies_decane	0.08	0.28	0.00	0.00	0.00	0.00	1.00
parentspecies_toluene	0.68	0.47	0.00	0.00	1.00	1.00	1.00
parentspecies_apin	0.00	0.03	0.00	0.00	0.00	0.00	0.50

y	mean	std	min	25%	50%	75%	max
log10(pSat_Pa)	-3.849223	2.179720	-13.789350	-5.241530	-3.805215	-2.394863	5.864807

We can see that the scale of the values vary between the features. For this reason, standardizing should be introduced in pre-processing for certain learning or additional pre-processing methods.

At the time of performing the analysis, I did not pay enough attention to the value of y. By comparing the extreme values and percentiles, one could deduce that there may be outliers in y.

Let us study pairwise relationships of the features. We do this visually with `seaborn.pairplot`.



From the resulting grid plot, we can see pairwise relationships of each feature including also target y. Additionally, the plot shows histograms of each feature. We can make some interesting observations:

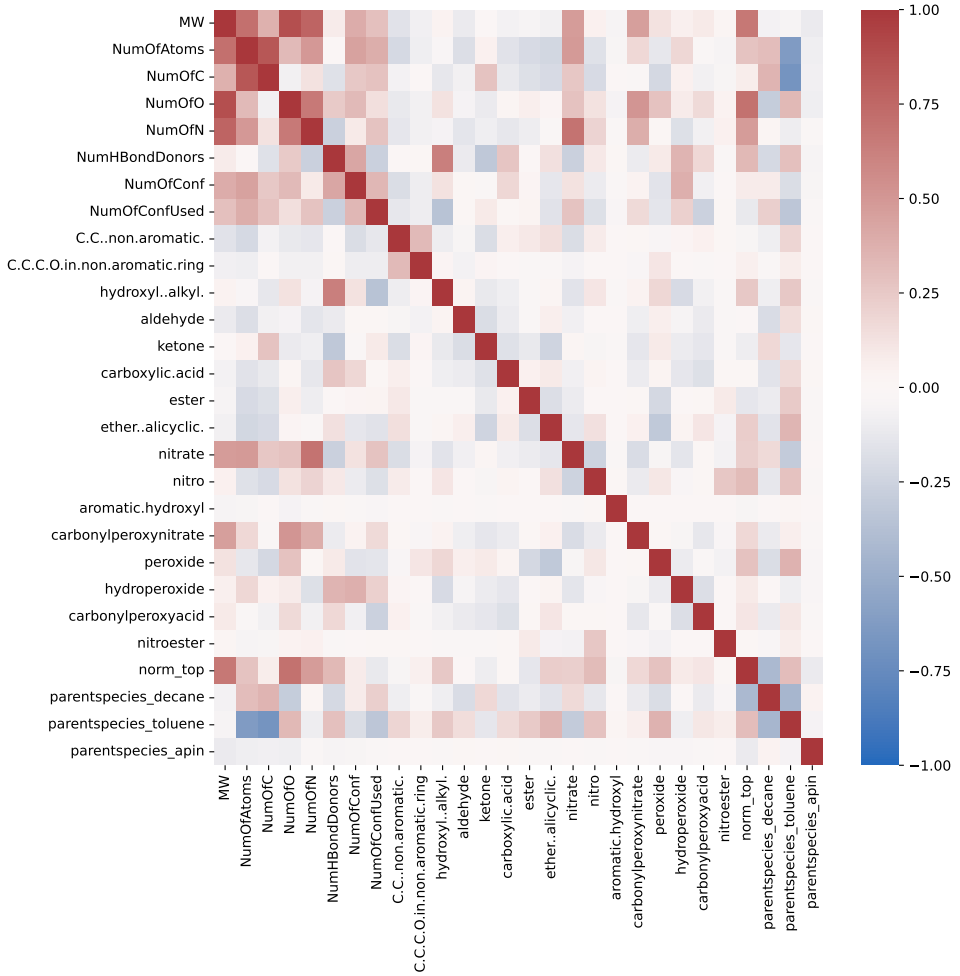
- There is collinearity between certain features such as between `MW` (molecular weight) and `NumOfAtoms` (number of atoms).
- Many of the features are not normally distributed. For example, `NumOfConfUsed` has a huge peak at 40.

Collinearity can be further verified by computing pairwise correlations with `pandas.DataFrame.corr`. Below are 10 feature pairs with the highest correlations.

		correlation
MW	NumOfO	0.881592
NumOfAtoms	NumOfC	0.838139
MW	NumOfN	0.773071
MW	NumOfAtoms	0.708921
norm_top	NumOfO	0.699744
NumOfN	nitrate	0.689724
parentspecies_toluene	NumOfC	0.685387
MW	norm_top	0.666598
NumOfO	NumOfN	0.659296
hydroxyl..alkyl.	NumHBondDonors	0.630017

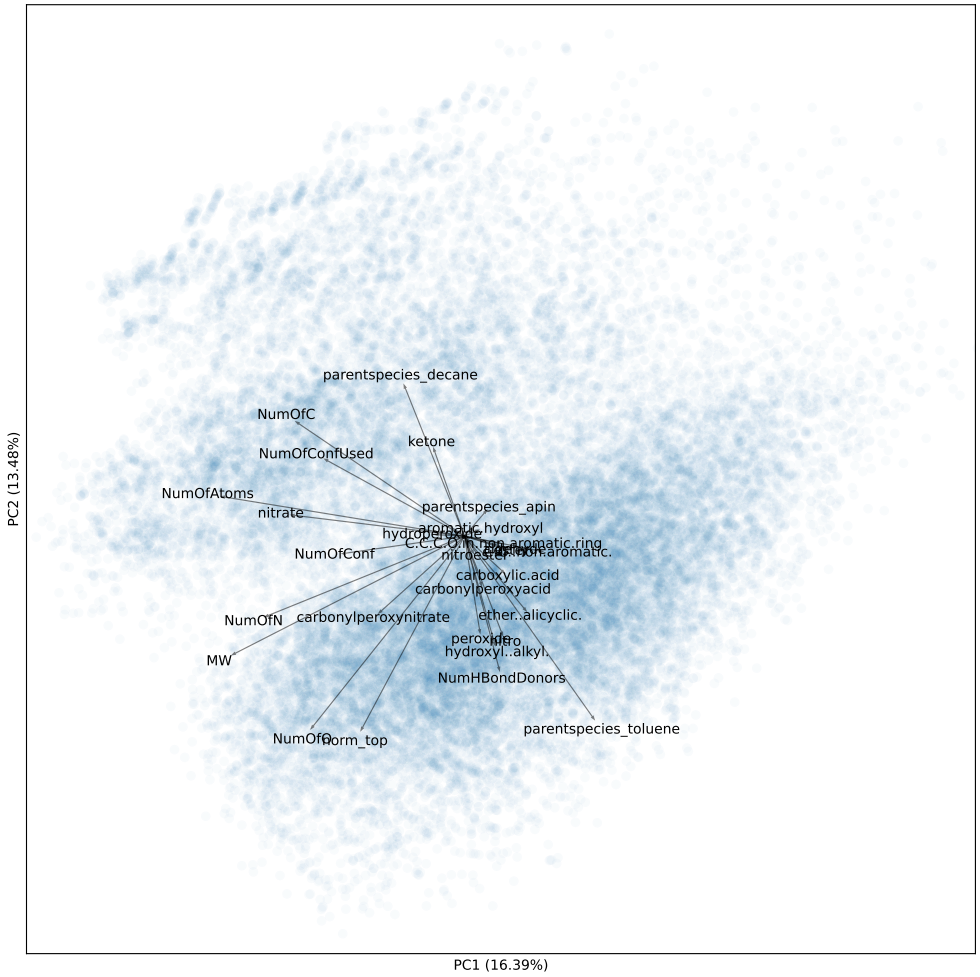
Indeed, among others, `NumOfAtoms`, `NumOfC`, and `NumOfO`, in addition to being correlated with each another, are correlated with the molecular weight, `MW`. Collinearity should be minimized by choosing features with minimal pairwise correlation or by choosing principal components with principal component analysis (PCA).

Another way to get the overall picture of the matter is using a heatmap (`seaborn.heatmap`). Below, using the same data as above, is a heatmap of the pairwise correlations. From it, we can make the same observations as above.



Principal component analysis

Let us perform a principal component analysis (PCA) for the combined train and test data set and a create a biplot (now considering ADV data).

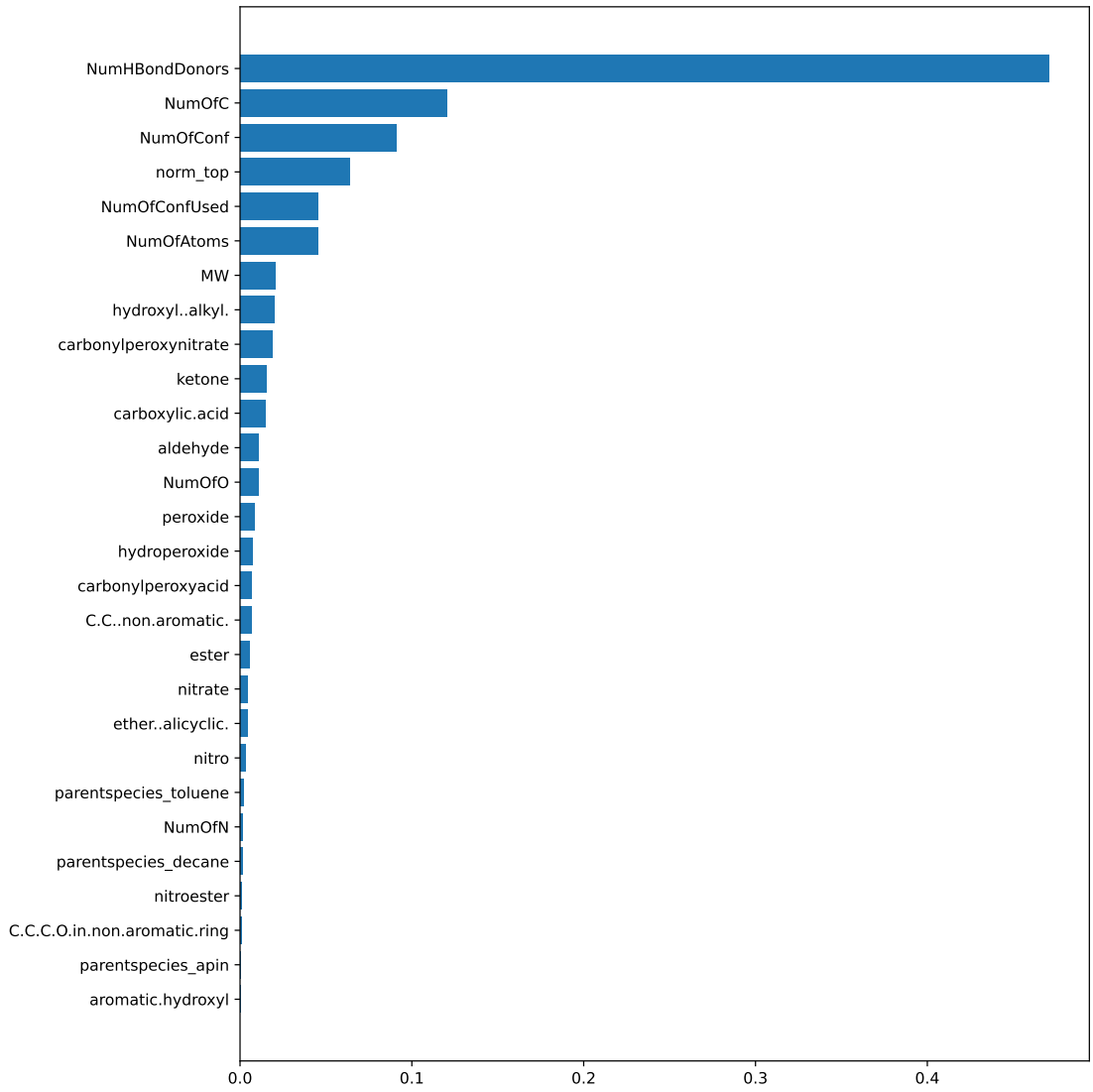


It is apparent that not a single component dominates the explained variance as even the first one contributes less than 17% of the total variance. Additionally, we can see that many features with notable loadings such as `NumOfC`, `NumOfO`, `MW`, `NumOfConfUsed` and `parentspecies_toluene` influence both observed components. On the other hand, `NumOfAtoms` and `nitrate` influence mainly the first component PC1. Some features have significantly lower loadings including `nitroester`, `aromatic.hydroxyl`, `C.C.C.O.in.non.aromatic.ring` and `hydroperoxide`.

> *PCA was covered during the course only after I applied PCA for this project. Instead of using the components in training, I tried to select features based on their loadings for the first or first two components. Obviously, this was not sufficient since their share of the explained variance did not cover even 50%.*

Feature importances

One option to study, which features to choose, is to train a random forest regressor and extract its feature importances. Using the training data for ADV, and default parameters for `sklearn.ensemble.RandomForestRegressor` we fit the model and extract importances with attribute `feature_importances_`. The result is as follows.



`NumHBondDonors` seems to dominate in importance. Interestingly, features with the least importance (e.g. `nitroester`, `C.C.C.O.in.non.aromatic.ring`, `parentspecies_apin`, `aromatic.hydroxyl`) also have small loadings in the biplot shown above. On the other hand, some features with large loadings (`parentspecies_decane`) are not considered important, according to this method.

Insights from literature

Assignment mentions an article (Besel et al. 2023) that provides additional information about the data set. In other words, it is a source to gain *domain knowledge*. For instance, according to Besel et al. (2023) saturation pressure p_{Sat} (`pSat_Pa` in the data set) can be computed with SIMPOL method which is based on

$$\log_{10} p_{Sat} = \sum_k v_k b_k,$$

where v_k is the number of functional groups of type k found in a molecule and b_k is a group-specific parameter that has been fitted to reference data. In the data set, columns representing functional groups are `C.C.C.O.in.non.aromatic.ring`, `hydroxyl..alkyl.`, `aldehyde`, `ketone`, `carboxylic.acid`, `ester`, `ether..alicyclic.`, `nitrate`, `nitro`, `aromatic.hydroxyl`, `carbonylperoxynitrate`, `peroxide`, `hydroperoxide`, `carbonylperoxyacid`, `nitroester`.

This implies there is a linear relationship between the features and the target variable and, thus, even a linear model could perform decently.

However, according to Besel et al. (2023),

> [...] functional groups can establish intermolecular as well as intramolecular interactions. Intermolecular interactions lead to a stabilization of the molecule in the liquid phase, i.e a low p_{Sat}, whereas intramolecular interactions stabilize the molecule in the gas phase and lead to a high p_{Sat}.

For this reason, just as Pankow and Asher (2008) suggest, such properties can be taken into account by introducing "higher-order" groups. Therefore, we may need to introduce polynomial features (at least degree 2) before performing ordinary least squares (OLS) regression. This could be done to only features that are related to functional groups, or apply to all features. Ideally, we would gain more knowledge about which features actually are expected to need interaction terms.

Furthermore, related to computing p_{Sat}, we observed above that `NumOfConfUsed`is peaking at 40 (over 40% of all observations). Based on the instructions, this is a feature that indicates how many conformers are used to compute thermodynamic properties of a molecule. We assume that the higher number indicates more accurate p_{Sat} value. If interpreted correctly, Besel et al. (2023) support this claim (see, for example, figure 4 a). Therefore, `NumOfConfUsed` could be used to create a subset of the data.

Detecting outliers

During the course, outlier detection has not been covered (at least thoroughly). However, clustering methods such as DBSCAN is shown. In fact, that can be used as a simple outlier detection method. First, let us combine train and test data (ADV in this example) and scale the data (`sklearn.preprocessing.StandardScaler`). After this, running `sklearn.cluster.DBSCAN` with chosen hyperparameters, we get clusters in which, ideally, the largest cluster contains all valid samples and other clusters samples that can be excluded. Below are some results considering only train data clustered with the combined train and test data.

hyperparameters	$n_{clusters}$	$n_{train\ in\ largest}$	$n_{train\ in\ largest}/n_{train}$
`eps=0.1, min_samples=5`	18	26839	0.9962
`eps=0.5, min_samples=5`	609	22562	0.8375
`eps=1.0, min_samples=5`	1094	16867	0.6261
`eps=2.0, min_samples=5`	409	2557	0.0949
`eps=0.5, min_samples=1`	20319	38	0.0014
`eps=0.5, min_samples=2`	4936	13174	0.4890
`eps=0.5, min_samples=5`	609	22562	0.8375
`eps=0.5, min_sam...=10`	107	25481	0.9458
`eps=0.5, min_sam...=20`	14	26590	0.9870

As can be seen, choosing correct hyperparameters is essential for this task. It is not trivial. However, without further analysis, it is reasonable to be conservative in labeling samples as outliers and reduce the sample size carefully. Ideally, we would verify that the choice is valid. One, though, not always the most optimal option to validate the choice is to train a model with and without outlier removal and compare the resulted performance scores.

There are plenty of other methods for outlier/anomaly detection available. One simple-to-use option is `sklearn.neighbours.LocalOutlierFactor` which is based on k-nearest neighbors. Below some results considering only train samples analysed using combination of train and test set.

hyperparameters	$n_{outliers}$	$n_{valid\ train}/n_{train}$
`n_neighbors=1`	6782	0.7483
`n_neighbors=5`	1757	0.9348
`n_neighbors=10`	676	0.9749
`n_neighbors=20`	264	0.9902

I wish I had paid more attention to the outlier removal. I should have found better ways to verify the result than just try running the learning pipeline and validate the result or at least investigated the matter in a more systematic way (also verify at Kaggle). Perhaps visual analysis of the distributions could have helped. In the final submissions, it was not included since tested results were better without it or the effect was negligible.

Machine learning pipeline and model selection

The machine learning pipeline consists of four main steps.

1. pre-processing
2. learning
3. validation
4. performing predictions

Pre-processing involves at least extracting features X and targets y. It also includes necessary transformations such as scaling values, possibly creating new or choosing a subset features. Possible outlier removal is performed at this stage as well.

Learning means the phase during which a chosen model (with chosen hyperparameters) is trained using the pre-processed data. Validation, a task aiming to measure the quality of the model, takes place jointly with learning in case cross-validation is used.

Once a model is validated and the results are satisfying, it can be taken into use to perform predictions. In our case, this means that we are confident enough to submit the result to Kaggle. One could argue, that this should be considered as part of validation, though.

Naturally, the actual model selection process consists of multiple iterations of each step. By combining the possible approaches for the steps 1 to 3, we get all possible combinations for the overall pipeline. Ideally, we would validate all possible combinations. However, as we introduce more possible pre-processing tasks and models with each usually having several tunable hyperparameters, we quickly run out of computing capacity. Therefore, only some of the selected combinations are tested. Some tools such as Optuna can be used to help optimization process, though.

I mainly used trial-and-error when selecting pipelines either changing some part of the pre-processing or the learning phase. I also tested Optuna but it was time consuming to create studies and usually the results ended up being only fractions of better than by manually adjusting the values.

Pipeline options

Below are identified options for each step.

Pre-processing

- $y = \log_{10} p_{Sat}$.
- Handle missing values
  - drop rows with missing values
  - impute, `sklearn.impute.KNNImputer`
- Encode categorical
  - one-hot encoding for `parentspecies`

    - handle ambigious categories in `parentspecies`
- ADV: Include fingerprints as L^2-norms.
- Scale values
    - `sklearn.preprocessing.StandardScaler`
- Create polynomial features
    - `sklearn.preprocesssing.PolynomialFeatures` (`degree`=[2,3,4], `interaction_only`=[True, False])
    - apply to all selected features
    - apply only to functional groups
- Choose X:
    - features
        * stepwise selection (involves learning and validation)
        * based on PCA
        * n important features
        * features resulting in minimum collinearity
        * knowledge based (e.g. consider only functional groups)
    - samples
        * filter `NumOfUsedConf` $\geq$ `threshold`

Learning

- Models
    - linear models (`sklearn.linear_model`)
        * ordinary least squares (`LinearRegression`)
        * ridge (`Ridge`, `RidgeCV`)
        * lasso (`Lasso`, `LassoCV`)
        * elastic net (`ElasticNet`, `ElasticNetCV`)
    - nearest neighbors (`sklearn.neighbor`)
        * KNeighborsRegressor (`KNeighborsRegressor`)
    - Ensemble (`sklearn.ensemble`)
        * random forest (`RandomForestRegressor`)
        * custom ensemble (e.g. unique models for chosen subsets)
    - Others (not teached during the course, just for reference)
        * support vector regression (`sklearn.svm.SVR`)
        * gradient boosting (`xgboost.XGBRegressor`, `lightgbm.LGBMRegressor`)
- hyperparameter tuning
    - methods with built-in CV such as `RidgeCV`, `ElasticNetCV`
    - manual tuning based on CV
    - Optuna studies

Validation

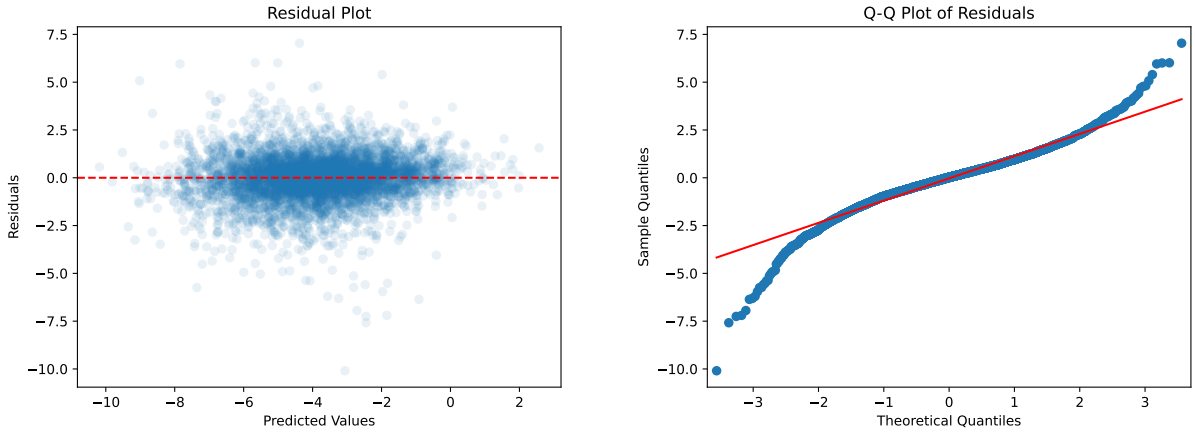
- metric: R^2
- train-test split (`sklearn.model_selection.train_test_split`)
- cross validation (CV, `sklearn.model_selection.cross_val_score`), $k = [5, 10]$.
    - average, minimum, confidence interval: $CI = \overline{x} \pm t_{\alpha/2} \frac{s}{\sqrt{n}}$
- Kaggle score
- visual analysis:
    - Residuals
    - Q-Q plot

Selected pipelines

We select pipelines based on their Kaggle score, for the main and the advanced competitions individually. Turns out that the best scores are achieved with the following pipelines.

Main competition

- Selected model: *elastic net*
- Results (R^2):
  - Training: 0.8045
  - CV (mean): 0.7646
  - Kaggle (public): 0.67587 (#39)
  - Kaggle (private): 0.6889 (#34)
- Pipeline:
  1. Read data
  2. Drop rows with nan (there were not too many compared to sample size)
  3. Extract X and y ($y = \log_{10}(p_{Sat})$)
  4. Encode `parentspecies`
     - one-hot encoding with `pandas.get_dummies`, using `drop_first=True`
     - handling ambiguous categories with `handle_parentspecies` (see above)
  5. Filter `NumOfConfUsed` == 40. (This improved the result a bit. Though, there is a risk that generalization is compromized)
  6. Perform scaling with `StandardScaler` (usually done after polynomial, however, for some reason, this order lead to a better result)
  7. Create polynomial features with `PolynomialFeatures(degree=2, include_bias=False)`
  8. Train a elastic net regressor with `ElasticNetCV(cv=10)`
- Plots



It seems that residuals are over-dispersed relative to a normal distribution, i.e. there is an increased number of outliers.

- Selected features
  - Due to introduced second order features, there are 403 features in total to choose from (including intercept-term). Elastic net sets coefficients for each feature which also determine their significance (further from zero, more significant; if zero, no effect). This can be considered as feature selection.
  - 18 features with coefficients ≤ -0.1 or ≥ 0.1:

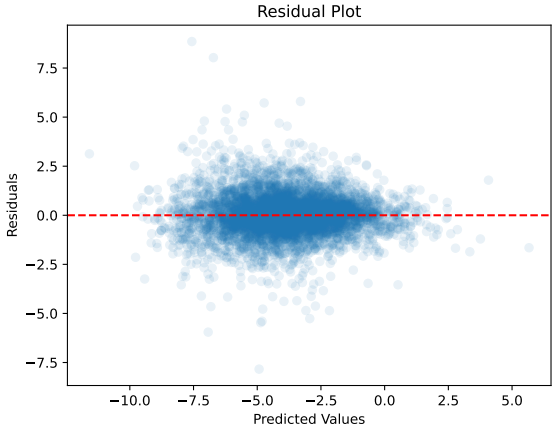
feature	coefficient
intercept	-3.781407
NumOfC	-0.943019
NumHBondDonors	-0.794401
NumOfConf	-0.504439
carboxylic.acid	-0.306650
carbonylperoxynitrate	0.285378
NumOfAtoms NumOfO	-0.217274
parentspecies_decane^2	-0.178482
NumOfO parentspecies_decane	-0.168786
peroxide parentspecies_toluene	-0.165101
parentspecies_decane parentspecies_toluene	0.131421
NumOfO ketone	0.130564
aldehyde ketone	0.124648
NumOfAtoms^2	0.121055
NumOfC ester	0.117905
aldehyde	-0.113658
hydroperoxide	-0.110566
NumOfAtoms	0.108717

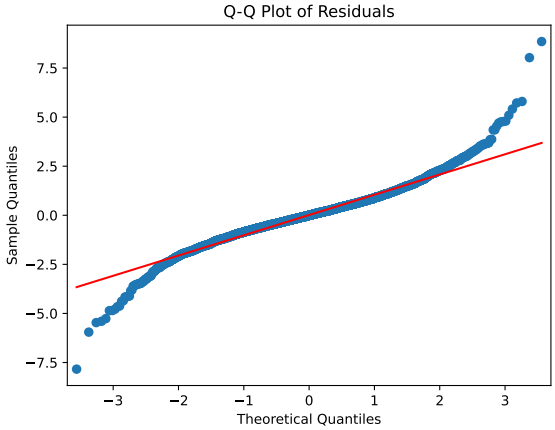
  - 108 features with coefficients within $]-0.10,-0.01]$ or $[0.01,0.10[$
  - 234 features with coefficients that are 0.

- Hyperparameter tuning
  - hyperparameters are tuned by the method itself (`ElasticNetCV`) with build-in CV
  - `alpha=0.0028718434126416`
  - `l1_ratio=0.5`

Advanced competition

- Selected model: *ridge*
- Results (R^2):
  - Training: 0.7663
  - CV: 0.7360
  - Kaggle (public): 0.7719 (#2)
  - Kaggle (private): 0.77129 (#3)
- Pipeline:
  1. Read data, also topographical fingerprints
  2. Take norms of fingerprints and add those as an additional feature to training data
  3. Drop rows with nan (there are not too many compared to sample size)
  4. Extract X and y ($y = \log_{10}(p_{Sat})$)
  5. Encode `parentspecies`
     - one-hot encoding with `pandas.get_dummies`, using `drop_first=True`
  6. Create polynomial features with `PolynomialFeatures(degree=2)`
  7. Train a ridge regressor with `Ridge()`
- Plots





Similar case as with the elastic net above. Though, tails of the residuals' distribution are not perhaps as extended.

- Selected features
  - Similarly as with the previous pipeline, due to polynomial features, there are over several hundreds of features to choose from. To be precise, 529. The higher number is a result of additional `norm_top` feature in the ADV set. Ridge shrinks coefficients of potentially unnecessary features. As a result, some coefficients may become so small that, in practice, they are treated as zero by the computer. This time, however, we do not have standardization of the features which makes it harder to compare the significance of the features as the scales vary.
  - Nonetheless, below are top 20 coefficients:

feature	coefficient
intercept	7.453788
NumHBondDonors	-1.524327
ester nitroester	1.437870
carbonylperoxynitrate	1.348374
nitroester^2	1.167483
ester parentspecies_toluene	-1.157633
ester	-1.157633
NumOfC	-1.103245
ester^2	0.910768
C.C..non.aromatic. ether..alicyclic.	0.846101
peroxide parentspecies_toluene	-0.730877
NumOfO ketone	0.720534
C.C.C.O.in.non.aromatic.ring peroxide	-0.672752
NumOfO aldehyde	0.655866
ketone peroxide	-0.628301
NumOfN	0.621042
NumOfO nitroester	0.617488
aldehyde peroxide	-0.613562
NumOfN aldehyde	-0.574587
NumOfAtoms aromatic.hydroxyl	0.550714

  - 8 features with coefficients ≤ -1.0 or ≥ 1.0
  - 208 features with coefficients $]-1.0, -0.1]$ or $[0.1, 1.0[$
  - 73 features with practically zero coefficients.
- Hyperparameter tuning

– hyperparameter `alpha` of `Ridge` is manually tuned. However, as changes to default value `alpha=1.0` do not seem to have a notable impact, a default value is used.

Discussion

Pros (+) and cons (−) of the selected models are as follows.

(+) models are interpretable

(+) models are fast to train

(+) models predict fast

(+) pipelines are simple and easy to implement

(−) predicting performance could be better; perhaps polynomial regression is not the most optimal after all or more sophisticated feature engineering is needed

(−) pipelines do not involve advanced pre-processing that could make predictions more robust, e.g. outlier removal

(−) due to introduction of polynomial features, depending on the pipeline, there are over 400 or 500 features used in training. Luckily, especially elastic net with standardized values, is able to exclude most of them. Ridge, with used pipeline, seems to be less efficient in deciding the most relevant features.

Observations about the pipelines:

- There are many identified pre-processing options available that could have been included. However, for some reason methods, such as outlier removal or data imputation, that are left out do not have a significant impact to the final score. There is a benefit of keeping the model simple, so without actual utility, it is better to leave such additional steps out. However, it is possbile that better tuning would be needed to realize their potential.
- Feature selection with e.g. step forward method does not seem to result in an improved Kaggle score. Perhaps, with shrinking methods that does not play a vital role.
- It is actually a bit disappointing that, for the advanced competition, such a simple solution (which is actually the first attempt) results to the best individual score.

Learnings:

- Gain domain knowledge and understanding of the data early on as it helps in feature engineering and model selection. Understand what each feature is about. Is it necessary? Have you understood its type correctly? Does it contain missing or ambiguous values? How to handle those properly? Are all values valid or should you take a subset?
- Even though many attempts of various pre-processing methods did not yield to any better R^2 score, many models performed almost equally good as the selected ones but with much less features, for example. This type of competition, where just the R^2 score is monitored, takes the focus from other, sometimes as important aspects of the modelling.
- Even if features are not yet fully engineered, it is worth trying with different models early on, right when the initial pre-processing is ready (relevant features encoded and X and y extracted). Actually, for a long time, anything done after finding the first properly performing model (based on CV) did not have a significant impact to the score.
- Cross-validation is a useful tool. However, it is not enough to observe the average value alone since it usually doesn't reflect the score in Kaggle (at least in case of the main competition). Actually, the smallest value may be much better predictor of the score of Kaggle submission than the average alone. At some point, observing also the lower bound of the confidence interval turned out to be useful (takes variance also into account). Also frequent submission to Kaggle is adviced.
- Model tuning (selecting hyperparameters and features) quickly becomes time consuming especially with models that have many tunable hyperparameters such random forest Therefore, it is better to start with simpler model, try to optimize it first to get some benchmark. Using cloud platforms to

perform the most exhaustive measurements could save time as every extensive evaluation could halt the delevopment for many minutes or even hours.
- One solution may yield satisfying result with one test set, but when applied to another, completely different results occur. This seems to be the case when testing predictions for the main and the advanced competitions. It may be just by chance, or test data sets are, in fact, from different distributions.
- Try to introduce limited number of changes to your pipeline at a time to keep on track of the effects to the score. Be systematic when testing different pipeline combinations. It is reasonable to document the history in a way or another as well. On the other hand, given that there are many pre-processing steps and models with tunable parameters to choose from, the number of combinations explodes. Therefore, sometimes to quickly explore the options, broader changes and less documentation may be justified.
- Helper functions improve the workflow and can reduce the risks appearing when copy-pasting own code multiple times. On the other hand, if such global methods are widely used, it is essential to test them carefully to avoid system wide errors.

Self-grading

Grade: 5

Overall, the project was successful. I provided all of the deliverables on time and according to the guidelines. I am proud that I managed to do all by myself (even though group work was encouraged). I even exceeded myself by giving a presentation to an almost full auditorium.

While the final solutions may not be particularly groundbreaking, those fulfill the requirements and are carefully thought. The placement in the main competition was slightly above the average, while in the advanced, with limited number of contestants, I finished third. My goal was to focus on methods that were taught during the course and to understand the whole pipeline by avoiding anything too complex. On that perspective, the result is decent.

I studied the problem from multiple angles and tried to solve it using various strategies. Performed analysis was backed up with credible sources. However, in hindsight, the actual research could have been conducted in a more systematic way from the start. On the other hand, since this was my first time to study such machine learning problem from start to finish, it is natural that the learning process involved some exploration.

The final report describes the research comprehensively, at the same time, giving enough information to reproduce the implementation but without going too much into details of the actual code. It provides a meaningful discussion showing critical thinking and deepened understanding of the topic. The document is carefully typed to make it look professional and reading experience pleasant. Similarly, the given presentation was well prepared and visually appealing.

References

Besel, Vitus, Milica Todorović, Theo Kurtén, Patrick Rinke, and Hanna Vehkamäki. 2023. "Atomic Structures, Conformers and Thermodynamic Properties of 32k Atmospheric Molecules." *Scientific Data* 10 (1): 450.

Pankow, James F, and William E Asher. 2008. "SIMPOL. 1: A Simple Group Contribution Method for Predicting Vapor Pressures and Enthalpies of Vaporization of Multifunctional Organic Compounds." *Atmospheric Chemistry and Physics* 8 (10): 2773–96.