# Machine Learning Project with Quantum Chemistry

The aim of this project is to set up a method to find significant clusters of 25116 molecules described by 8 scalar features. This README describes the code that has been used to obtain the results in the report (see `run.py`) and describes the command line system (see `main.py`) that can be used to obtain supplementary results.

**Table of Contents**



# Getting started

## Prerequisites

You need a recent python installation (`> 3.5`). The required python packages can be found in `requirements.txt` and easily be installed with:

```
pip install -r requirements.txt
```

We recommend using a virtual environment.

For machine learning methodes, the library scikit-learn is used

For plotting, `matplotlib` calls TeX so you also need to have a recent and sufficient TeX distribution installed on your system in order to get all the figures.

## Data

The data to be analysed by this project can be found in `data/` where two files are present: `proposition_of_features.txt` (molecules and their features) and `comp_BoB_reordered_energies.txt` (Delta G (Rxn A) energies associated to the molecules). You can, of course, provide your own data if you have it.

# Reproducing the results

To reproduce the results, you can simply run:

```
python run.py
```

or, if you have a sufficient TeX distribution installed on your system:

```
python run.py --usetex
```

The first command generates all the figures that do not depend on TeX. The second command generates all the figures. They will be saved to the `plots/` directory.

To show the figures on the run, specify the `-S` or `--show` option. That, however, requires user interaction (closing figure windows) during the run.

Use `-N` or `--no-plot` to suppress plotting. (This has no use for revision and correction of the project.)

Specifiy `-h` or `--help` for a help message.

# Command-line interface

This program provides a command-line interface for analysis in `main.py`. `run.py` is only a wrapper for result reproduction.

With the interface, it is, e. g., possible to provide other data or to adjust visualization options. The interface already gives the *default* values, i. e., the values used or found during the analysis to facilitate reproduction of the results.

## Command-line options

To start the pogram, you need to call it with python and specify what to do:

```
python main.py [OPTIONS...] COMMAND [COMMAND_OPTIONS...]
```

`[OPTIONS...]` need to be specified after `main.py` but not after `COMMAND`. `[COMMAND_OPTIONS...]` need to be specified after `COMMAND`.

**REMINDER:** The interface already gives the *default* values, i. e., the values used or found during the analysis to facilitate reproduction of the results.

## Help messages

If you need the help message for `main.py`, run it with `-h` or `--help`:

```
python main.py -h
```

If you need the help message for a command, run it with `-h` or `--help` *after* the `COMMAND`:

```
python main.py COMMAND -h
```

## Positional arguments for `main.py` (`COMMAND`s)

`COMMAND` needs to be one of {`rankestimation`, `kdistance`, `pcadbscan`, `plot`}:

- `rankestimation`: estimate the rank (number of meaningful dimensions) for the given data. This produces a plot of Gabriel error vs dimensions
- `kdistance`: builds k indices for the k-fold rows or columns partition. It is needed for the Gabriel cross-validation. Plots a k-distance graph
- `pcadbscan`: does a PCA on the data and then clusters it with DBSCAN. Afterwards, the data can be projected onto another dimension with another PCA.
- `plot`: starts different plotting functions which need to be specified.

## Optional arguments for `main.py` (`[OPTIONS...]`)

These options can be used to change the program flow. The default values are recommended and yield the results of the project.

- `-h`, `--help`: show help message and exit
- `-p PATH`, `--path PATH`: The path to the data folder. Default: ./data.
- `-f FILE`, `--file FILE`: The file that contains the proposition of features. Default: proposition_of_features.txt.
- `-e ENERGIES`, `--energies ENERGIES`: The file that contains the energies. Default: `comp_BoB_reordered_energies.txt`.
- `-t {png,pdf,ps,eps,svg}`, `--file-type {png,pdf,ps,eps,svg}`: Specifies the file type of the figures. Only used if `-P` or `--plot` is specified. Must be one of `["png", "pdf", "ps", "eps", "svg"]`.
- `-F FIGURE_PATH`, `--figure-path FIGURE_PATH`: Specifies the path to which the figures are saved. Only used if `-P` or `--plot` is specified. Default: `./plots`.
- `-P`, `--plot`: If specified, data plots will be generated and saved to files. Default: `False`.
- `-D DPI`, `--dpi DPI`: Specifies the resolution in dpi for the plots. Default: `300`.
- `-S`, `--show`: If specified, data plots will be shown on the run. This will require user interaction (closing windows) during the run. Default: `False`.
- `--figsize FIGSIZE`: The size of the figures. Default: (10, 8).

- `--fontsize FONTSIZE`: The font size in the figures. Default: 24.
- `--colormap COLORMAP`: The standard colormap for the figures. Default: `gist_rainbow`.
- `--usetex`: Use TeX for figure text. This requires a sufficient TeX distribution. It is needed for some figures. Default: `False`.
- `--tex-font-family TEX_FONT_FAMILY`: The TeX font for figure text. Only used if `--usetex` is specified. Default: `serif`.
- `--tex-text-size TEX_TEXT_SIZE`: The TeX text size for figure text. Only used if `--usetex` is specified. Default: `20`.
- `-E [ELEMENTS [ELEMENTS ...]]`, `--elements [ELEMENTS [ELEMENTS ...]]`: The list of elements to be excluded from or used for the program.
- `-r`, `--remove`: If specified, the given `--elements` list will be removed from the data. Otherwise, only the given elements will be used. Default: `False`.
- `-A`, `--add-information`: Specifies whether element information from `periodictable` should be added to the data. Default: `False`.
- `-i [INFORMATION [INFORMATION ...]]`, `--information [INFORMATION [INFORMATION ...]]`: Specifies which element information from periodictable should be added. Only considered if `-A` or `--add-information` is given.
- `-N`, `--no-normalize`: Specifies whether the data is normalized before calculations. Default: `Normalize`.
- `--expand`: Specifies whether the features should be expanded. Give the degree with -d POSITIVE_INTEGER (default: `2`). Default: `False`.
- `-d DEGREE`, `--degree DEGREE`: The maximum degree of expansion. Default: `2`.
- `-L {DEBUG,INFO,WARNING,ERROR,CRITICAL}`, `--logging {DEBUG,INFO,WARNING,ERROR,CRITICAL}`: Define the console logging level. Default: `CRITICAL`.

## Optional arguments for `COMMAND`s (`[COMMAND_OPTIONS...]`)

### Optional arguments for `rankestimation`

- `-h`, `--help`: show help message and exit
- `-N NUM_SEEDS`, `--num-seeds NUM_SEEDS`: The number of times the gabriel cross validation scheme has to be repeated. Default: `1`.
- `-R K_FOLD_ROWS`, `--k-fold-rows K_FOLD_ROWS`: The number of row-indices subsets to be considered. Default: `23`.
- `-C K_FOLD_COLS`, `--k-fold-cols K_FOLD_COLS`: The number of column-indices subsets to be considered. Default: `8`.

### Optional arguments for `kdistance`

- `-h`, `--help`: show help message and exit
- `-R RANK`, `--rank RANK`: The desired rank for the PCA. Default: `2`.

### Optional arguments for `pcadbscan`

- `-h`, `--help`: show help message and exit
- `--epsilon EPSILON`: The epsilon (radius of the hypersphere) for DBSCAN regions. It must be

a positive float. Default: `0.3`.
- `--min-samples MIN_SAMPLES`: The minimum number of data points that are needed in a hypersphere around a single data point such that this point is considered attractive. It must be a positive integer. Default: `12`.
- `-R RANK`, `--rank RANK`: The desired rank for the PCA. Default: `2`.
- `--projection-dimension PROJECTION_DIMENSION`: The number of dimensions to which the data is projected for visualization. Default: `2`. Do not change this value if you do not really know what you are doing or do not provide proper means of visualization!

**Optional arguments for `plots`**

- `-h`, `--help`: show help message and exit
- `--plots {energies,ligands,molecularweight,combination} [{energies,ligands,molecularweight,combination} ...]`: Specifies the wanted plots. Default: `energies`.

# Structure of the files

This README provides a short description of the source code and its structure.
For the sake of having a readable README file, the complete description (descriptions, arguments, returns, notes) of the functions has not been included. The more complete documentation (documented with python docstrings) can be found directly in the code.

## src/cluster_analysis.py

`cluster_analysis.py` contains all functions necessary for reducing the dimensionality and for clustering the data.

- **pca**: applies PCA on the data matrix x (alias function of scikit PCA). The dimension of the transformed matrix is specified as an argument.
- **dbscan**: performs a DBSCAN on the data matrix x. It returns a label for each row of x to express to which cluster each row has been assigned (alias function of scikit DBSCAN).
- **pca_dbscan**: calls `pca` to reduce the number of dimensions and then applies `dbscan` in this space.
- **pca_dbscan_projection**: calls `pca_dbscan` and then calls `pca` to project the data in 2 dimensions. This is necessary for visualization.

## src/debug.py

`debug.py` contains the functions specifying the logging mode.

- **init_log**: Init the logging system at the specified login level.
- **log**: Function call to output a warning message. It is shown in DEBUG mode.
- **error_exit**: Function call when an error occurs. It prints an error messageand exits the program.

# src/element_information.py

`element_information.py` contains all functions relative to select the data relative only to 1 of the 6 metal. In addition, it allows to add to the data matrix other chemical features relative to the metal.

- **get_element_by_symbol**: retrieves the element's periodictable object by its `symbol`.
- **create_lookup_symbol**: creates a one- or two-character string containing the element `symbol` that is used to look the respective element's information up
- **get_element_information_for_symbols**: retrieves the information for the specified elements.
- **flatten_element_information**: flattens the information given in the periodictable element object to a NumPy array.
- **flatten_element_array_information**: flattens the element information for a list of element objects.
- **get_element_information_array**: gets flattened element information for a list of element symbols.
- **add_element_information_to_data**: adds specified element information to given element information data.
- **select_element_indices_to_remove_from_data**: defines rows to delete from a list of element symbols and its corresponding data array. Useful to analyse the dataset considering only one of the 6 metals.
- **select_elements_from_data**: Removes rows for a set of symbols from a list of symbols and the corresponding data array.
- **select_energies_from_data**: Removes rows for a set of symbols from a list of symbols and the corresponding energy array.

# src/ligand_information.py

`ligand_information.py` contains all functions relative to extract the ligand number present inside the name of the molecule

- **split**: Custom function to split a string of the format XX_[Number1].[Number2].xyz into an array(Number1, Number2)
- **get_ligands_array**: Takes the a list of molecules names and outputs an array with the left and right ligand written in the name
- **get_ligand_distribution**: Takes a list of ligands (left and right) and calculates the distribution of the left and right ligand

# src/load_data.py

`load_data.py` contains all functions relative to load the data (features of the molecules) and energies Delta G (Rxn A)

- **load_features**: Loads the molecules matrix and their features

- **load_energies**: Loads the associated Delta G (Rxn A) energies of the molecules
- **build_poly**: Builds for each column a of the input matrix tx the following powers : a, a^2, ..., a^degree and returns the original matrix tx along with the new features.
- **expand_features**: Builds for each column a,b of the input matrix tx (with shape (N,D) ) the columns corresponding to the cross terms of the following polynomial : (a + b)^degree and returns the original matrix tx along with the new features columns.
- **split_data**: Splits (and shuffles) the data of the output vector y and input matrix tx into two set : training set and test set

# src/main.py

`main.py` reads the input args from the comman line. According to them, it uses the functions in `load_data.py` to prepare the data matrix. Then, it processes the given arguments and then calls the required functionality given by the other modules.

- **load**: Load the needed data according to the arguments given. If needed, given elements are removed.
- **main**: It calls the modules necessary to perform the task specified by the args.

# src/model_validation.py

`model_validation.py` contains functions useful to determine the number of dimensions to be kept, the DBSCAN parameters setting and the cluster quality assessment.

- **k_distance**: produces the sorted k-dist graph (k-th neighborhood distance (k-dist) versus points sorted in a decreasing k-dist order). It is useful to set the DBSCAN parameters `Eps` and `NumPoints`.
- **build_k_indices**: builds k indices for the k-fold rows or columns partition. It is needed for the Gabriel cross-validation.
- **plot_gabriel_holds_out**: plots the gabriel average prediction error versus the number of dimensions.
- **gabriel_holds_out**: uses the row and column indices partition to perform the Gabriel cros validation.
- **f_test**: performs the statistical test on a quantity passed as an argument.
- **individual_f_tests**: calls f_test passing as an argument the quantity at equation 2 of the report for each combination of the columns and rows subsets.
- **estimate_rank**: looks at the smallest dimension with an high number of significant individual p-values. This is considered as ideal rank for DBSCAN.
- **silhouette_score**: returns the silhouette score to assess the cluster quality

# src/option_interface.py

`option_interface.py` provides means to process command-line or module-call arguments.

- **check**: checks the given options for wrong values.

- **read_options**: processes the given options and returns them as attributes of an argparse `Namepace` object

## src/plots.py

`energies.py` contains all functions relative to

- **prepare_data**: loads and prepares the data necessary for the plotting
- **scatter_plot_energies**: plots the 2D PCA of the data colouring them according to the associated energies.
- **plot_cluster_ligand_count**: plots the histogram describing the ligand distribution within each individual cluster found by DBSCAN
- **scatter_plot_molecular_weight**: plots the 2D PCA of the molecules colored by their molecular weight.
- **plot_combination_colors**: tries to color code the last four features in the data set and apply it to the each point of the 2D PCA.
- **plot_visualization**: takes as an argument the list `plots` that specifies which graphs are needed. This function calls the appropriate function to generate them.

## src/visualization.py

`visualization.py` is a file to help for visualization options in plotting.

- **read_options**: Reads the command line options for visualization
- **do_plot**: Tells whether it should be plotted during the run.
- **get_option**: Gives an option value for a key of the visualization-options dictionary.
- **get_visualize_options**: Returns the option that were specified for the visualize function from the visualization options dictionary. Gives a default value for not given values.
- **visualize**: Visualizes a figure. I.e. saves it as file and shows it if specified.

# Authors


- **Joachim Koerfer**, currently in Master of Computational Science and Engineering, EPFL
- **Luca Viano**, currently in Master of Computational Science and Engineering, EPFL
- **Jannik Reichert**, currently in Bachelor in Computer Science, Technische Universität Berlin