Data Mining: Final Project: Molecular energy prediction

Tuesday 10th May, 2022

For the project you are going to work in groups of 2.

Aim

In this project you are going to work with molecular data and your goal is to predict the electronic energy of the coordinates (Energy_) and the free energy corrected energy at room temperature (Energy DG).

Dataset

The given dataset (data.csv) consists of a set of formatted values for 10.000 molecules/SMILES but for each molecule there are more than one coordinate set. This results in multiple descriptors and energies for each molecule. For your analysis you should use only the instance with the lowest energy (Energy_) per input molecule.

data.csv

The dataset consists of the following attributes:

- The simplified molecular-input line-entry system (SMILES) for the input molecules (see)
  - Chiral_Molecular_SMILES - This is the fully defined SMILES for the molecule
- the following energy values:
  - Energy_(kcal/mol) - This is the electronic energy of the coordinates
  - Zero_point_energy_(kcal/mol@0K) - This is the thermal correction at 0 K. It accounts for vibrations of the molecule

  - Enthalpy_(kcal/mol@298K) - This is the thermal correction at room temperature. It accounts for the population of vibrational states at this temperature
  - Gibbs_energy_(kcal/mol@298K) - This is the thermal correction including entropy at room temperature
  - Energy DG (kcal/mol) - This is the sum of Energy and Gibbs_energy,i.e. free energy corrected energy at room temperature

- electronic descriptors derived from quantum chemistry:
  - Core_Core_Repulsion_(kcal/mol) - The interaction energy of the nuclei
  - Core_Hamiltonian_(kcal/mol) - Interaction of 1 electron with each orbital summed
  - Two_Electron_Integral_(kcal/mol) - interaction of 2 electrons with the nuclei and each other
  - Electronic_Energy_(kcal/mol) Heat_of_Formation_ (kcal/mol) - All interactions considered
  - HOMO_Energy__(eV) - The potential of the highest orbital with electrons (Highest Occupied Molecular Orbital)
  - LUMO_Energy_(eV) - The potential of the lowest unoccupied orbital
  - H-L_Gap_(eV) - The difference in these values already computed

- 3D descriptors directly from RDKit:
  - PMI1
  - PMI2
  - PMI3
  - NPR1
  - NPR2
  - Radius_of_Gyration
  - Inertial_Shape_Factor Spherocity_Index
  - Asphericity
  - Eccentricity

Model

Before starting your analysis it is very important to understand the data and the objective of this project.

You are free to chose the representation of the molecules that you are going to use. If you want to work with one-hot-encoding molecular data you can find the code to convert a single smile string to a one-hot encoding in smile_to_hot()

function at utils.py. In order to predict the targets you can also use extra properties and descriptors (number of bonds, number of atoms, number of C, number of O e.t.c.) that you can extract using RDKit. (The properties that we mention are random examples, there are much more properties and descriptors for the molecules).

You are free to try different approaches and models, you can use ready libraries for your algorithm. The main focus is to see the model you have come up but also the other approaches that you tried. You have to understand deeply all the algorithms that you have tried.

After cleaning the data, using load_data() function in utils.py split the data into train and test. Use cross-validation for hyper-parameter optimization and select the best model based on statistical significant test. Only for the best model you will use the test data (of course you have to include in your report and present all the models that you have tried). **It is mandatory to use three or more different algorithms in addition to a baseline**[1].

The final model we would expect is a model that can work on universal data, which means it can give a reasonable prediction on different molecular datasets.

Report

You have to submit a formal report . Your report has to include the approaches that you followed and main results not only for your final model but for all the models you have tried. We want a full picture of what exactly you have done and how. You should also discuss the different performances you have with your methods and explain why these work or not. What is important is to show us that you have a good understanding of the problem and of how to model it, what are the problems you encountered and how you solved them.

Note that this is an open project, you can try many different approaches as long as they make sense to get the best performance (creative ideas are always welcome).

Final submission

For your final submission you have to submit on Moodle a folder named using your names (ex. NAME1_NAME2_DM_project) which should include your code (all the scripts), the dataset and your report (the report should also be saved using your names: NAME1_NAME2_DM_project.pdf). If the size of your submission is big you can upload your submission on SWITCH drive and put on Moodle the shared link.

[1]The baseline is a naive baseline algorithm, providing context on just how good a given method actually is.

Installation Instructions

Below you will find some instructions about how to install some packages that required for the project through Conda. Conda is a package manager that works on all the popular operating systems. If you do not already have it installed (e.g. through Anaconda) you can install it via Miniconda by following the instructions here – it doesn't matter which version of Python you pick at this stage. You can then setup the particular environment using the Conda yml file I put in the folder of the project.

Assuming you have Conda installed, install the environment by:

1. conda env create -f DM_project_env.yml
2. conda activate DM_project
3. And then finally check it worked correctly by running $ conda env list .

To activate your environment type: conda activate DM_project and to deactivate it you should type: conda deactivate. If you you need to add an additional package to your environment you can follow the instruction here.

APPENDIX

RDKit

RDKit is an open source Cheminformatics Software. It is a collection of cheminformatics and machine learning tools written in C++ and Python. It allows to work with many representations of chemical data and has a power to extract almost each chemical descriptor from the data you have. Some examples of using RDKit in order to visualize molecules and to extract different molecular descriptors you can find in smiles-rdkit.ipynb file. For more Informations about how to use RDKit and examples, you can find in the RDKit documentation.

SMILES representations

A molecule can be considered as an undirected graph G with a set of edges E and set of vertices V . There are several ways to represent graphs for machine learning. The most popular way is the SMILES string representation. SMILES strings are a non-unique representation which encode the molecular graph into a sequence of ASCII characters using a depth-first graph traversal. Atoms of chemical elements are represented by chemical symbols in capital letter. The hydrogen is usually ignored. Single bonds are not displayed; for double, triple and quadruple bonds we shall use '=', '#', '$' respectively. Atoms that are bonded must stand nearby. Ring structures are written by breaking each ring at an arbitrary point (although some choices will lead to a more legible SMILES than others) to make a 'straight non-ring' structure (as if it wasn't a ring) and adding numerical ring closure labels to show connectivity between non-adjacent atoms. Aromacity is commonly illustrated by writing the constituent B, C, N, O, P and S atoms in lower-case forms b, c, n, o, p and s, respectively. SMILES are typically first converted into a one-hot based representation. The representation of a molecule through a SMILE string is not unique and the non- uniqueness of SMILES arises from a fundamental ambiguity about which atom to start the SMILES string construction. However, it is possible to transform a smile into canonical form (using specific tools such as RDKit).

One Hot Encoding

One-hot encoding is a sparse way of representing data in a binary string in which only a single bit can be 1, while all others are 0. In the case of molecular graphs represented as a string, one-hot-encoding consists of encoding each character of the string with a binary vector.