CS 181 Practical 1

Practical competed by **Tomislav Zabcic-Matic** and **Matthew Leifer**

In this practical, we have explored using various machine learning to predict the HOMO-LUMO Gap for a large set of molecules represeted in SMILES format, with 256 "featues" of the molecules included. Among the tactics used were Linear Regression, Ridge Regression, and Random Forest Regression. We also attempted to use an Elastic Net and the Lasso Method, but did not get good results because we could not get the code to run properly. We also engineered additional features using RDKit.

Method/Process

Initially we tried to use an ElasticNet to model the raw data we were given without any feature extraction. We varied the parameters (α and β) that penalize the l1 and l2 norms of the weights to try to optimize the Elastic net however we ran into some problems. We allowed α and β to take on the values $(1, 10, 10^2, 10^3, 10^4, 10^5, 10^6)$. For some reason no matter which combination of parameters we used, the weights given by the model to optimize the regression did not change, or at least there was no change in the weights that could be measured by my computer. The weights we generated from this process performed about as well as the Linear regression and random forest did in the sample code. Because Elastic Net didn't seem to work, we turned to some other approaches.

We decided we would try to extract some more features from the data to regress upon. We used the Morgan Fingerprint Library to extract 256 extra features to include in the data. Additionally, we used sets with 512, and 1024 features in our tests. We also looked at a few extra properties of the molecules that we thought might also be good predictors for the HOMO-LUMO Gap. We determined these features by looking at the SMILES representations of the molecules with some of the highest and lowest HOMO-LUMO Gaps and tried to see if there were any clear distinctions between the high gap and the low gap molecules. As shown later in this report, these features we added manually significantly improved the quality and strength of our analysis and the predictive power of our model.

Throughout the process, we had to take into consideration the sheer size of the data sets that we were training on and predicting HOMO-LUMO Gaps for. This resulted first in the use of smaller additional feature sets, and later on in techniques used to minimize the amount of RAM taken up by the regression process, and even slightly reduce runtime.

Dealing with Size of Data:

Initially, we attempted to do a linear regression in which we attached a feature set of 1024 Morgan fingerprint features to the back of the Python "pandas DataFrame". This, however, led to issues during the linear regression step. Due to a lack of RAM, even on a 16GB device, the program would always receive a "Killed 9" message during execution of the

`LinearRegression.predict(...)`

step. Discussed in the "Final Comments" section below, we discuss the specifics of how we solved this problem.

Overview of Features

From the start, we planned on using additional features, so we first started by looking for some trends in the training data, primarily certain elements of the SMILES strings that seemed to influence the HOMO-LUMO Gap . We ended up with 9 different elements of SMILES strings, which are listed in the table below:

Feature	**Value Set**
# of Sets of Parentheses	$\mathcal{V}_1 = \{1, 2, \dots\}$
# of [nH] Groups in String	$\mathcal{V}_1 = \{1, 2, \dots\}$
# of [SiH2] Groups in String	$\mathcal{V}_1 = \{1, 2, \dots\}$
# of [se] Groups in String	$\mathcal{V}_1 = \{1, 2, \dots\}$
# of = Signs	$\mathcal{V}_1 = \{1, 2, \dots\}$
# of Standalone Carbons (C) in String	$\mathcal{V}_1 = \{1, 2, \dots\}$
Presence of "ccccc" in String	$\mathcal{V}_2 = \{0, 1\}$
Does string begin with "c1sc"	$\mathcal{V}_2 = \{0, 1\}$
Does string end with "c12"	$\mathcal{V}_2 = \{0, 1\}$

Afterwards, we decided to use RDKit in order to mass engineer many additional features to add to our model.

Results

The first results that we got were for the linear regression model, where we added 256 additional features from the Morgan fingerprint for additional training accuracy. We started with a low set of additional features due to the fact that running machine learning algorithms on a million-line data set, and then with techniques we decided on later, we were able to perform calculations for up to 1024 additional features.

Significant results are summarized in the following table, listed in increasing order of accuracy. We would have hoped to make more trial runs, but the long runtime of the program made for few results when run only on a normal computer.

Approach	**Error**
Linear regression using 256 RDKit features and 9 extra features	Error ≈ 0.2063
Linear regression using 512 RDKit features	Error ≈ 0.1764
Linear regression using 512 RDKit features and 1st additional feature	Error ≈ 0.1722
Linear regression using 512 RDKit features and 2 additional features	Error ≈ 0.1718
Linear regression using 512 RDKit features and 4 additional features	Error ≈ 0.1713
Linear regression using 512 RDKit features and 6 additional features	Error ≈ 0.1705
Linear regression using 512 RDKit features and 9 additional features	Error ≈ 0.1689
Linear regression using 1024 RDKit features and 9 additional features	Error ≈ 0.1327
R.F. regression using 256 RDKit features and 9 additional features	Error ≈ 0.1154
R.F. regression using 512 RDKit features and 9 additional features	Error ≈ 0.0817

Final Comments

In order to combat this, we initially simply used smaller sets of additional features with the goal of making the process less RAM intensive. This, of course, worked, and allowed us to use feature sets of 256 and 512 additional features.

As we performed more iterations, we wanted to decrease the runtime and RAM usage of the program, so we explored shrinking the size of the data (in terms of bytes) so that the data would not take up as much memory, and so that it could be iterated through faster. Although this step took a time, it was a constant that the decreased the memory usage and runtime of later steps. The code was a simple iteration of

```
df_all = df_all.astype("uint8")
```

executed after dropping the "smiles" column. The above code made it possible for us to use the computationally expensive 1024 additional features, and even made it possible for us to run the Random Forest Regression calculation, which we previously had never completed because it was taking far too much time to execute.