Predicting Molecular Properties Using Machine Learning

Anthony Yazdani
University of Geneva
MSc Statistics
Anthony.Yazdani@etu.unige.ch

***Abstract*—In this project, we present different methods to predict certain molecular properties from molecules in SMILES format. We focus on the fundamentals, such as loss functions, optimization, regularization, validation and model selection. We discuss generalized linear models and the multilayer perceptron. We also present a fully supervised Transformer-based model, which we call the Self Attention Model.**

I. Introduction

The objective of this project is to use molecular data to estimate four molecular properties. The molecular properties are the octanol-water partition coefficient, the number of rotational bonds, the molecular weight and the number of the rings. Molecules are represented as SMILES (Simplified Molecular Input Line-Entry System) strings [12]. To achieve our goal, we will begin with an exploratory analysis of the data. In a second part, we will describe the theoretical foundations of our models, including the loss functions we used and a detailed explanation of the model architectures. In a third part, we will discuss the learning framework, including hyper-parameters and optimization. Finally, we will conclude with a discussion of the results and propose further improvements.

II. Data set

As response variables we have, the $logP$ which represent a measure of the tendency of a compound to move from the aqueous phase into lipids, RBN which represent the number of bonds which allow free rotation around themselves, MW which represent the weight of a molecule based on the atomic masses of all atoms in the molecule and RN which represent the number of connected sets of atoms and bonds in which every atom and bond are a member of a cycle.

In this project, we started by shuffling and dividing our data into three sets. The training set which consists of 92428 instances, the validation set, and the test set both consisting of 19806 observations.

A. *Exploratory Data Analysis*

We have established that $logP$ and MW appear to follow a normal distribution, while RBN and RN appear to follow a Poisson distribution. In fact, RBN seems to have an excess of zero. We propose some extensions to this in the discussion section. You will find more details of this analysis in the dedicated code.

Now that we have a better understanding of the distribution of the response variables, we can discuss the appropriate loss functions. Prior to that, we will discuss how we intend to represent our SMILES strings.

B. *Feature Extraction and Data Representation*

Machine learning tasks such as regression requires mathematically and computationally tractable input data. In doing so, we need to find a way to mathematically represent the SMILES strings. We will discuss two methods. The first method, on the scale of the molecule itself, is called count vectorization. The second method, on the atomic scale, is to learn a vector representation for each unique character in the SMILES strings.

The first method involves transforming the text into an array of token counts. Each molecule will be represented by a vector, and its dimensionality will be equal to the vocabulary size (the number of unique characters in the set of training strings). In this project, the vocabulary consists of the following thirteen tokens

$$voc = [\#, (,), 1, 2, 3, 4, 5, =, C, F, N, O] \, .$$

As an example, molecules such as $M_1 \equiv NCC(C\#N)N1CCC1$ or $M_2 \equiv C$, will be represented as follows

$$M_1 \equiv [1, 1, 1, 2, 0, 0, 0, 0, 0, 6, 0, 3, 0] \, ,$$

$$M_2 \equiv [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0] \, .$$

The second method consists in automatically learning a vector representation of each token (token embedding). To do so, we assign to each token a randomly initialized N dimensional vector. Each of these vectors are then treated as parameters in the optimization procedure.

III. Models

A. *Loss functions*

As explained in the exploratory data analysis, the labels seem to follow either a Gaussian distribution or a Poisson distribution. For this reason, we will derive the loss functions adapted to the distribution of these labels. From a probabilistic point of view, we are looking for the parameter vector $\mathbf{w}$ such that

$$\begin{aligned}\mathbf{w}^{\star} &= \arg\min_{\mathbf{w}} -\log p(\mathbf{w} \mid \mathbf{X}, \mathbf{y}) \\ &= \arg\min_{\mathbf{w}} -\log \frac{p(\mathbf{y}, \mathbf{X} \mid \mathbf{w}) p(\mathbf{w})}{p(\mathbf{y}, \mathbf{X})} \\ &= \arg\min_{\mathbf{w}} -\log p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) p(\mathbf{w}).\end{aligned} \tag{1}$$

Assuming that $\mathbf{w}$ can take values in $\mathbb{R}$ uniformly, we want

$$\mathbf{w}^{\star} = \arg\min_{\mathbf{w}} -\log p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}). \tag{2}$$

For the Gaussian model, we have the following scalar conditional density

$$p(y_i \mid \mathbf{x}_i, \mathbf{w}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - f_{\mathbf{w}}(\mathbf{x}_i))^2}{2\sigma^2}}. \tag{3}$$

Therefore, we want to find the parameters $\mathbf{w}_{\mathcal{N}}$ such that

$$\mathbf{w}_{\mathcal{N}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} (y_i - f_{\mathbf{w}}(\mathbf{x}_i))^2. \tag{4}$$

For the Poisson model, we have

$$p(y_i \mid \mathbf{x}_i, \mathbf{w}) = \frac{\left(e^{f_{\mathbf{w}}(\mathbf{x}_i)}\right)^{y_i} e^{-e^{f_{\mathbf{w}}(\mathbf{x}_i)}}}{y_i!}. \tag{5}$$

As a result, we need to figure out the parameters $\mathbf{w}_{\mathcal{P}}$ such that

$$\mathbf{w}_{\mathcal{P}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} e^{f_{\mathbf{w}}(\mathbf{x}_i)} - y_i f_{\mathbf{w}}(\mathbf{x}_i). \tag{6}$$

B. *Baseline Models*

In this project, we will use the constant model as our baseline model. In the context of generalized linear models, this is the best (log) bias your model can produce given that it has not seen any features. This model is therefore simply the empirical mean over the training samples.

C. *Generalized Linear Models*

It is important to know if we can solve our problem in a simple way. If this is the case, it is not necessary to produce extremely complex models. Under the roof of Generalized Linear Models (GLM) we simply fix that $f_{\mathbf{w}}(\mathbf{x}_i) = \mathbf{x}_i^{\top}\mathbf{w}$.

D. *Multilayer Perceptron*

One of the strong assumptions with GLMs is that the conditional expectation of our response variable is linearly related to the features. One solution to increase the flexibility of our models would be to use a universal approximator [2], [6]. The MultiLayer Perceptron (MLP) is an artificial neural network composed of several layers in which information flows only from the input layer to the output layer. Mathematically, the output $x^{(l)}$ of the l^{th} layer is a function of the output of the previous layer, i.e.,

$$\mathbf{x}^{(l)} = f^{(l)}\left(\mathbf{x}^{(l-1)}\right) = \phi\left(\mathbf{W}^{(l)}\mathbf{x}^{(l-1)} + \mathbf{b}^{(l)}\right), \tag{7}$$

where ϕ is the activation function, $\mathbf{W}^{(l)}$ is the l^{th} weight matrix and $\mathbf{b}^{(l)}$ is the l^{th} bias vector.

The final function that is applied, denoted by $f^{(L)}$, depends on the underlying task. It can be adjusted to perform classification or regression. In our case, the last layer will be a simple linear layer that directly produces the estimated label for the Gaussian model, or the logarithmic estimate for the Poisson model.

To avoid the vanishing gradient effect, we will restrict ourselves to ReLu activation functions. More precisely, we will use the leaky ReLu function ($\mathcal{LR}(x)$), known to be robust to the vanishing gradient effect and to have a non-zero derivative on $\mathbb{R}^-$.

$$\mathcal{LR}(x) = \begin{cases} x & \text{if } x \geq 0 \\ -|\alpha|\, x & \text{otherwise.} \end{cases} \tag{8}$$

E. *Self Attention Model*

Another model that could be interesting would be a model that would allow more flexibility when representing a molecule. To do so, we can take advantage of token embedding. Moreover, it can be interesting to vary the representation of each token according to the other tokens in a molecule. This is called contextualized embedding. To do so, we propose to focus on an attention-based model, as in the Transformer [11] or BERT [3] model. We call this model the Self Attention Model (SAM) and explain it component by component in the following.

Let E_i be a $1 \times N$ real-valued embedding vector of the i^{th} token in a SMILES string. We represent the whole molecule as a matrix E of dimension $A \times N$ of stacked embeddings, where A is the maximum molecule length in our data set. Note that E is always of dimension $A \times N$, and that we will have to introduce a padding token.

In a second step, we add a position embedding to each token embedding.

The idea is to add a vector P_i of dimension $dim(E_i)$ to the token E_i, which is meant to represent the order of this token in the molecule. Positional embeddings allow our model to know that the data it receives may have an order, or that the distance between tokens may be an important parameter. Therefore, we have

$$\tilde{E} = P + E, \tag{9}$$

where P is treated as a parameter and is always the same for any embedded molecule E.

The essential next step is to compute three linear transformations of $\tilde{E}$ as follows

$$Q = \tilde{E} \cdot M_Q, \tag{10}$$

$$K = \tilde{E} \cdot M_K, \tag{11}$$

$$V = \tilde{E} \cdot M_V, \tag{12}$$

where M_K is called the key matrix, M_Q the query matrix and M_V the value matrix. All M matrices are of dimension $N \times N$.

Then, the self attention output is obtained as follows

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{N}}\right) V, \tag{13}$$

where $\sqrt{N}$ act as a scaling factor for numerical stability.

We have that each token embedding in the molecule changes with respect to the other tokens. By doing this, we say that we have created contextual embeddings.

Note that all operations where a padding token is involved in QK^T are replaced by $-\infty$. After the softmax function is applied, the padding token do not influence the self attention mechanism. This method is known as masking and was first introduced in [11].

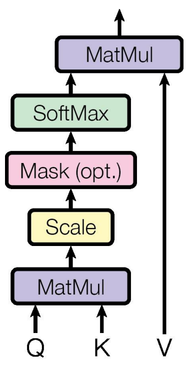

Figure 1. Self Attention Computational Graph. Source: [11]

The attention mechanism is only one element of what we call the Transformer block. Once we have obtained $\mathrm{Attention}(Q, K, V)$, the latter output is fed to a feed-forward layer. In addition, the Transformer block applies layer normalizations [1] and introduces additive skip connections. Here follows the computational graph

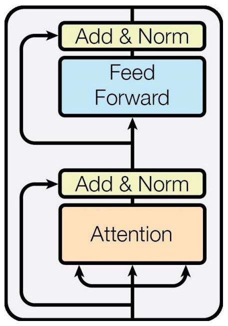

Figure 2. Transformer Block Computational Graph. Source: [11]

The feed-forward layer is a slightly unusual one. For more details, see the code or the BERT paper [3].

The output of the Transformer block then enters another Transformer block, for C consecutive times. The number of blocks usually depends on the complexity of the underlying task.

The last crucial step is to take a linear combination of the embeddings and pass it to an MLP as described in the section III-D. Here is a summary of the whole model in the form of a computational graph

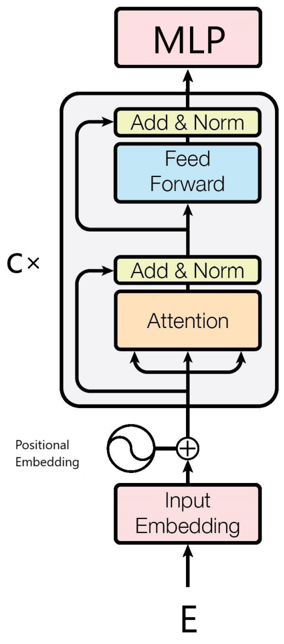

Figure 3. Self Attention Model Computational Graph. Source: [11]

IV. Optimization & Regularization

Mini-batch Gradient Descent (MGD) is problematic in regions where the loss surface exhibits steeper curvature in one dimension than in the others [10]. The point is that a small derivative may indicate a gently curving part of the surface that requires large steps, while a large derivative may indicate a steep part of the surface that requires smaller steps to avoid instability. The problem with MGD is that it uses the same learning rate for all dimensions. This is why we will use the Adaptive moment estimation (Adam) algorithm [4].

While this may not be useful for simple models such as GLMs, it is very effective for models such as SAMs. We hypothesize that this is due to a very complex loss landscape for this type of model. To further facilitate convergence, we also had to implement learning schedules for some of the tasks. Full details of these schedules can be found in the corresponding Jupyter notebooks. The learning rates depend mainly on the underlying task. In general, we needed a small learning rate such as $1e-05$. To efficiently train the SAMs, we also needed a lot of epochs (between 100 and 200) and a small batch size (64).

We also implemented validation-based early stopping to avoid over-fitting. We define early stopping as no improvement on the validation loss for 5 consecutive epochs.

No regularization has been implemented for GLMs, nor for MLPs. However, we implemented dropout [9] for the SAMs.

V. Model quality & selection

During training, the model quality is monitored using the validation loss. The optimization process is stopped when the quality of the models no longer improves. In doing so, we make a biased choice, i.e., we choose the best model for a particular data set. To further evaluate the generalization error, we end up evaluating our model on the test set.

For model selection, we need to check whether a model is the best one by using statistical tests. To do this, we need several evaluations on a test set (in order to estimate the mean and variance). We base our approach on dividing the test set into 1000 independent blocks.

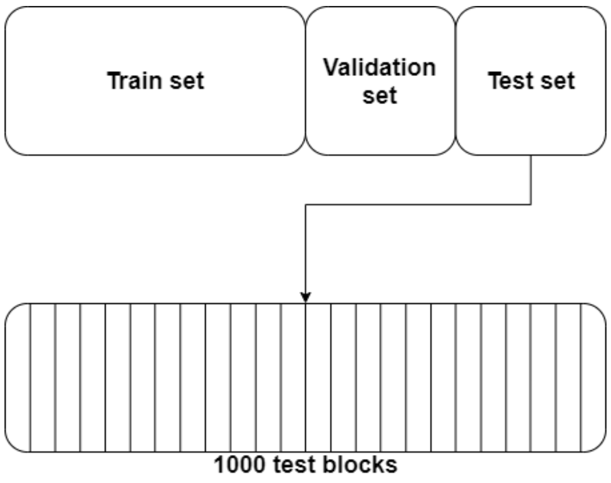

Figure 4. Data Splitting Strategy.

Once we obtained all the test losses on the test blocks, we proceed as follows.

For a model A and B, we assume that if the average test loss of A is higher and statistically different from that of B, then we should use B instead of A. Let $(\bar{A}, \hat{\sigma_A})$ and $(\bar{B}, \hat{\sigma_B})$ be the empirical means and estimated standard deviations of the test losses of A and B.

Assuming $\bar{A} \perp \bar{B}$, we compute the p_{value} of the following statistical test: $H_0 : \bar{A} = \bar{B}$ and $H_a : \bar{A} \neq \bar{B}$.

By the law of large numbers, the corresponding p_{value} is:

$$p_{value} = 2 \times \mathbf{P}\left(Z > \frac{|\bar{A} - \bar{B}|}{\sqrt{\frac{\hat{\sigma_A}^2}{1000} + \frac{\hat{\sigma_B}^2}{1000}}} \right),$$

where $Z \sim N(0, 1)$.

Note that we do not use a Student distribution to implement this test. This is possible because we have a large number of test blocks.

In addition, we emphasize that it is not useful to compare all models one by one. Suppose that $\bar{A} > \bar{B}$ and $\bar{B} > \bar{C}$, then we choose $\bar{C}$ without testing whether $\bar{A} > \bar{C}$. As for the level of significance, we can use the standard rate of 5%. But we also insist that this rate is totally arbitrary and that it is not necessarily the right thing to do. In fact, we suggest interpreting the p_{value} rather than not using a model because the difference is not statistically significant. In other words, we prefer to use the p_{value} as a measure of stochastic distance and not naively conclude that a model is not better because it does not reach our arbitrary significance level.

VI. Experimental setting

A. Software and Hardware

For this project, we essentially implemented all models with NumPy and PyTorch on Windows 10. We chose to use PyTorch for its GPU-accelerated computations, auto-differentiation and ease of implementation. As for the hardware, we used a computer with 32GB of ram and an NVIDIA GeForce GTX 1660. The batch size being small and the data rather light, all models can be implemented on a machine with much less memory.

B. Model Hyper-parameters

We will discuss here the hyper-parameters of our models. They were all chosen by grid search. No hyper-parameters are needed for the default models or for the GLMs, except for the learning rate (full details can be found in the code). We will only discuss the MLP and SAM for each response variable.

1) Octanol-water partition coefficient:
For this first response variable, we used a 5 hidden layer MLP. For the SAM, we used an embedding of dimension 300, 3 Transformer blocks, dropout with probability 0 and 2 hidden layers in the output MLP.

2) Rotational bonds:
For this second response variable, we used a 2 hidden layer MLP. For the SAM, we also used an embedding of dimension 300, 3 Transformer blocks, dropout with probability 0 and 2 hidden layers in the output MLP.

3) Molecular weight:
For this third response variable, we used a 2 hidden layer MLP. For the SAM, we used an embedding of dimension 10, 1 Transformer block, dropout with probability 0 and 1 hidden layer in the output MLP.

4) Number of the rings:
For this last response variable, we used a 5 hidden layer MLP. For the SAM, we used an embedding of dimension 300, 3 Transformer blocks, dropout with probability 0 and 2 hidden layers in the output MLP.

VII. Results

In this section, we present tables of the average block test losses and their corresponding standard deviations for each model. When a response variable follows a Poisson distribution, we will also show the obtained test accuracy. In most cases, the results are self-explanatory. We elaborate when necessary. We also show the results of statistical tests, as described in the section V.

A. *Octanol-water partition coefficient*

	Default	GLM	MLP	SAM
Loss mean	0,896094	0,231529	0,159881	0,000916
Loss std	0,276749	0,069964	0,059519	0,003071

Table I
Octanol-water partition coefficient block test metrics.

	Default vs GLM	GLM vs MLP	MLP vs SAM
p_{value}	0	0	0

Table II
Octanol-water partition coefficient p_{values}.

B. *Rotational bonds*

	Default	GLM	MLP	SAM
Loss mean	0,995458	0,707407	0,674125	0,314547
Loss std	0,02296	0,146625	0,147699	0,179624
Accuracy	28,70%	52,17%	55,44%	99,35%

Table III
Rotational bonds block test metrics.

	Default vs GLM	GLM vs MLP	MLP vs SAM
p_{value}	0	4.2e-07	0

Table IV
Rotational bonds p_{values}.

C. *Molecular weight*

	Default	GLM	MLP	SAM
Loss mean	57,408415	1,07E-07	0,000026	0,236044
Loss std	45,235691	7,06E-08	0,000579	5,844503

Table V
Molecular weight block test metrics.

	Default vs SAM	SAM vs MLP	MLP vs GLM
p_{value}	0	0,2013	0,1572

Table VI
Molecular weight p_{values}.

We see that the p_{value} are high while the average error seems to be very different. This is due to the high variance of the worst model compared to the best model in the statistical test (e.g. $7,06e-08$ vs $0,000579$). Since the variance is high (relative to the other model), an even more extreme difference in means is required for the test to yield a small p_{value}. We should choose the GLM for this task. We know that MW represents the weight of a molecule based on the atomic masses of all atoms in the molecule. Therefore, it is not surprising that we should use a linear model given how we represented the data.

D. *number of the rings*

	Default	GLM	MLP	SAM
Loss mean	0,767414	0,406849	0,346643	0,321948
Loss std	0,158669	0,215042	0,220024	0,436491
Accuracy	27,67%	85,83%	86,22%	99,55%

Table VII
Number of the rings block test metrics.

	Default vs GLM	GLM vs MLP	MLP vs SAM
p_{value}	0	5.96e-10	0.1099

Table VIII
Number of the rings p_{values}.

Here the last p_{value} is also high. This is due to the small difference between the average test losses. However, in this case, a small difference in the loss function represents a large difference in terms of accuracy. Therefore, we concluded that the SAM was the best model.

VIII. Conclusion and Discussion

One of the fundamental challenges of ML is to find an appropriate representation of the data. We were able to implement two methods to accomplish this task. We concluded that token embedding was particularly effective, especially because it allows for much more powerful models. We believe that working on such methods is crucial given the amount of data generated that are intractable in their raw format. We also discussed the importance of understanding the distribution of the data in order to formulate appropriate loss functions. In this project, we were able to examine several models and show that using a neural network without trying a simpler model could be a bad idea, as shown by our models for molecular weight. We also concluded that some tasks required a more powerful model and that the SAM could be particularly effective. However, this model is not without its flaws as it is particularly difficult to optimize.

Although the results are satisfactory, we can still suggest some improvements. We believe that it is essential to find models that are easier to train. We could explore other approaches than Transformer-type models, such as Convolutional Neural Networks (CNN) or Recurrent Neural Networks (RNN). However, we can also try to solve this problem by reducing the number of trainable parameters in our self attention models. To do so, we could explore other context-based embedding methods than the attention mechanism. Recently, a new approach has been proposed by a team of researchers at Google. This new method is based on the decomposition of the embedding matrix with the fast Fourier transform [5]. Another possibility to improve the convergence speed would be to pre-train the embeddings in an unsupervised way. In this framework, we would not need labeled data and could take advantage of much larger data sets. This can be done with methods such as GloVe [8] or Word2Vec [7].

Another idea that could be interesting is to use loss functions adapted to the excess of zero for the RBN task. These include the hurdle or the zero inflated model.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[4] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[5] James Lee-Thorp, Joshua Ainslie, Ilya Eckstein, and Santiago Ontanon. Fnet: Mixing tokens with fourier transforms. *arXiv preprint arXiv:2105.03824*, 2021.

[6] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[7] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[8] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[10] Richard Sutton. Two problems with back propagation and other steepest descent learning procedures for networks. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society, 1986*, pages 823–832, 1986.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[12] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36, 1988.