

Open in app ↗

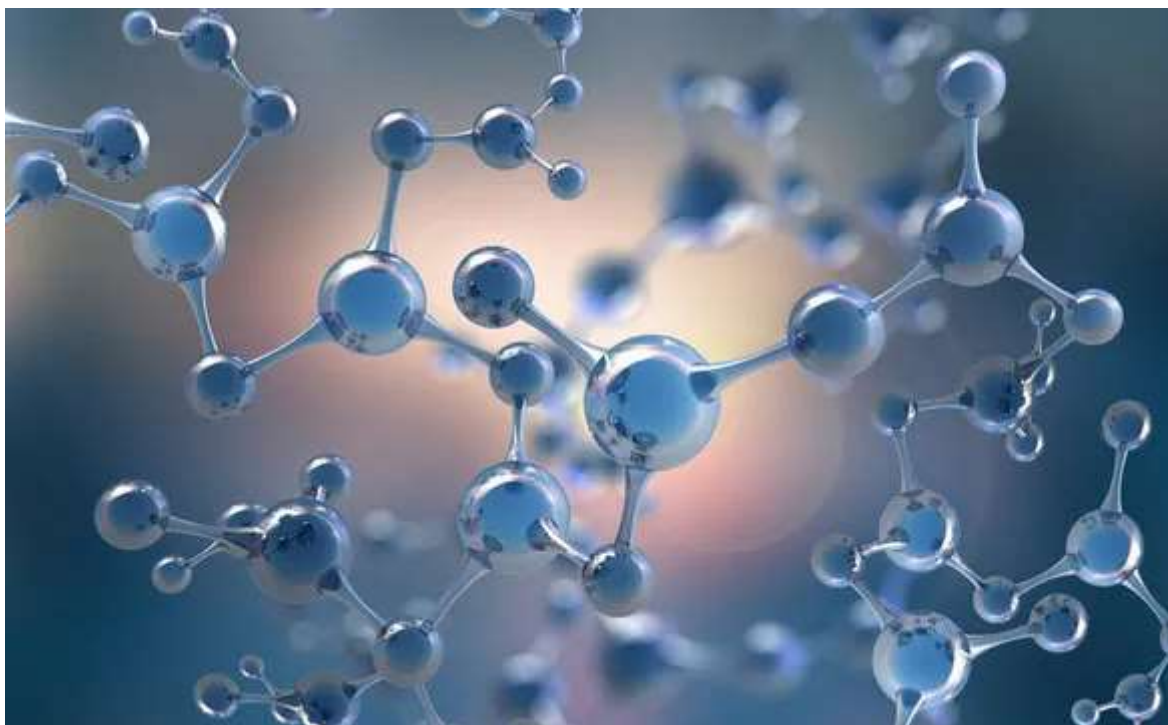


Search

Write



★ Get unlimited access to the best of Medium for less than \$1/week. [Become a member](#)



SMILES Toxicity Prediction



2040 Final Project · [Follow](#)

8 min read · Apr 6, 2021



3



Part One

By [Cameron Webster](#), [Jillian Green](#), [Chris Rohlicek](#), and [Akshay Shah](#)

We are graduate students in the Data Science Master's program at Brown University. As part of our Deep Learning and Special Topics in Data Science course (Data 2040), we worked on processing molecular compounds using sequence models.

Project Overview

Chemistry is a field that has recently experienced a great surge of research attention from the machine learning community. This crossover between disciplines has been spurred by the fact that many of the modern text-processing techniques that have been developed in machine learning are very useful in extracting information from chemical compounds. While this can feel like a pretty big conceptual leap to go from text to chemistry, we just have to ask: what's the difference to a machine learning model between a sequence of letters and a sequence of atoms?

With the barrier between language and chemistry broken down, we enter into a rich field of research investigating different methods of processing and extracting useful properties from a chemical compound expressed as a sequence of characters. One of the standard ways of writing compounds in this way is using the SMILES (Simplified Molecular-Input Line-Entry System) format. With SMILES we use different characters to symbolize different kinds of atoms, bonds, rings, etc. to get expressions see Figure 1.


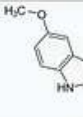




| Molecule | Structure | SMILES formula |
|-------------------------------------|---|---|
| Dinitrogen | $N=N$ | <chem>N#N</chem> |
| Methyl isocyanate (MIC) | $CH_3-N=C=O$ | <chem>CN=C=O</chem> |
| Copper(II) sulfate | $Cu^{2+}SO_4^{2-}$ | <chem>[Cu+2].[O-]S(=O)(=O)[O-]</chem> |
| Vanillin |  | <chem>O=Cc1ccc(OC)c(O)c1</chem> <chem>COc1cc(C=O)ccc1O</chem> |
| Melatonin ($C_{13}H_{16}N_2O_2$) |  | <chem>CC(=O)NCCC1=CNc2c1cc(OC)cc2</chem> <chem>CC(=O)NCCc1c[nH]c2ccc(OC)cc12</chem> |
| Flavopereirin ($C_{17}H_{15}N_2$) |  | <chem>CCc1c1ccc2[n+](c1)ccc3c2[nH]c4c3cccc4</chem> <chem>CCc1c[n+](c1)ccc3c4cccc4[nH]c3c2cc1</chem> |
| Nicotine ($C_{10}H_{14}N_2$) |  | <chem>CN1CCC[C@H]1c2cccnc2</chem> |
| Oenanthotoxin ($C_{17}H_{22}O_2$) |  | <chem>CCC[C@@H](O)CC\C=C\C=C\C=C\CC#C\C=C\C=CCO</chem> <chem>CCC[C@@H](O)CC/C=C/C=C/C=C\CC#C/C=C/C/CO</chem> |
| Pyrethrin II ($C_{22}H_{28}O_5$) |  | <chem>CC1=C(C(=O)C[C@@H]1OC(=O)[C@@H]2[C@@H](C2(C)C)/C=C\C=C/C(=O)OC)C/C=C\C=C</chem> |

Figure 1: Example of SMILES data in format of molecule, visual structure, and SMILES [9].

The problem that we're tackling in this project is how to predict molecular toxicity from a molecule's SMILES expression. The data we're starting with is the Tox21 data set from the National Institutes of Health [1,10] and we're using presence of the SR-p53 protein as a proxy for toxicity (more on this later in the EDA section).

In terms of approaches for this type of molecular analysis, the literature is broad and expanding quickly but focuses on the standard set of sequence methods that have led to many of the advances in NLP in recent years. These include models like RNNs, GRUs, and LSTMs, as well as more powerful models like transformers and variational auto-encoders to learn more sophisticated sequence embeddings through the use of attention.

Intro To Tokenization

When working with SMILES data, the first preprocessing step that we need to do to transform our data into a sequence format that can be used by our model is *tokenization*. Tokenization is a standard preprocessing step in any sequence-learning task; in the case of text processing tokenization may

looking like breaking up a sentence into a sequence of letters, and likewise tokenization in our problem looks like breaking up SMILES compounds into elements of our molecular alphabet.

Exploratory Data Analysis (EDA)

Before creating our model, we analyze and investigate the dataset and summarize our findings. This is an important step in data science models, because it allows us to better understand the data, the outliers, and any errors. We begin with high level summaries and then dive deeper into more specific analyses.

The dataset used for a preliminary baseline model comes from the Tox21 Data Challenge 2014. This competition, whose purpose was to crowdsource data analysis on how certain chemicals do or do not interfere with certain biological pathways. The datasets for this competition contained tables of SMILE data-label pairs where the labels for each dataset represented binary outcomes indicating whether or not a chemical interferes with a particular essential molecule's function. Of the several datasets available, we chose the one with labels indicating interference in the function of p53, a tumor-suppressing protein that shows increased bodily prevalence in the face of abnormal cells propagating from replication. Known as the "guardian of the genome", p53 plays a vital role in binding to DNA to prevent genetic mutations [7]. Below is a graphic demonstrating the diversity of use-cases for p53 and its derivatives.

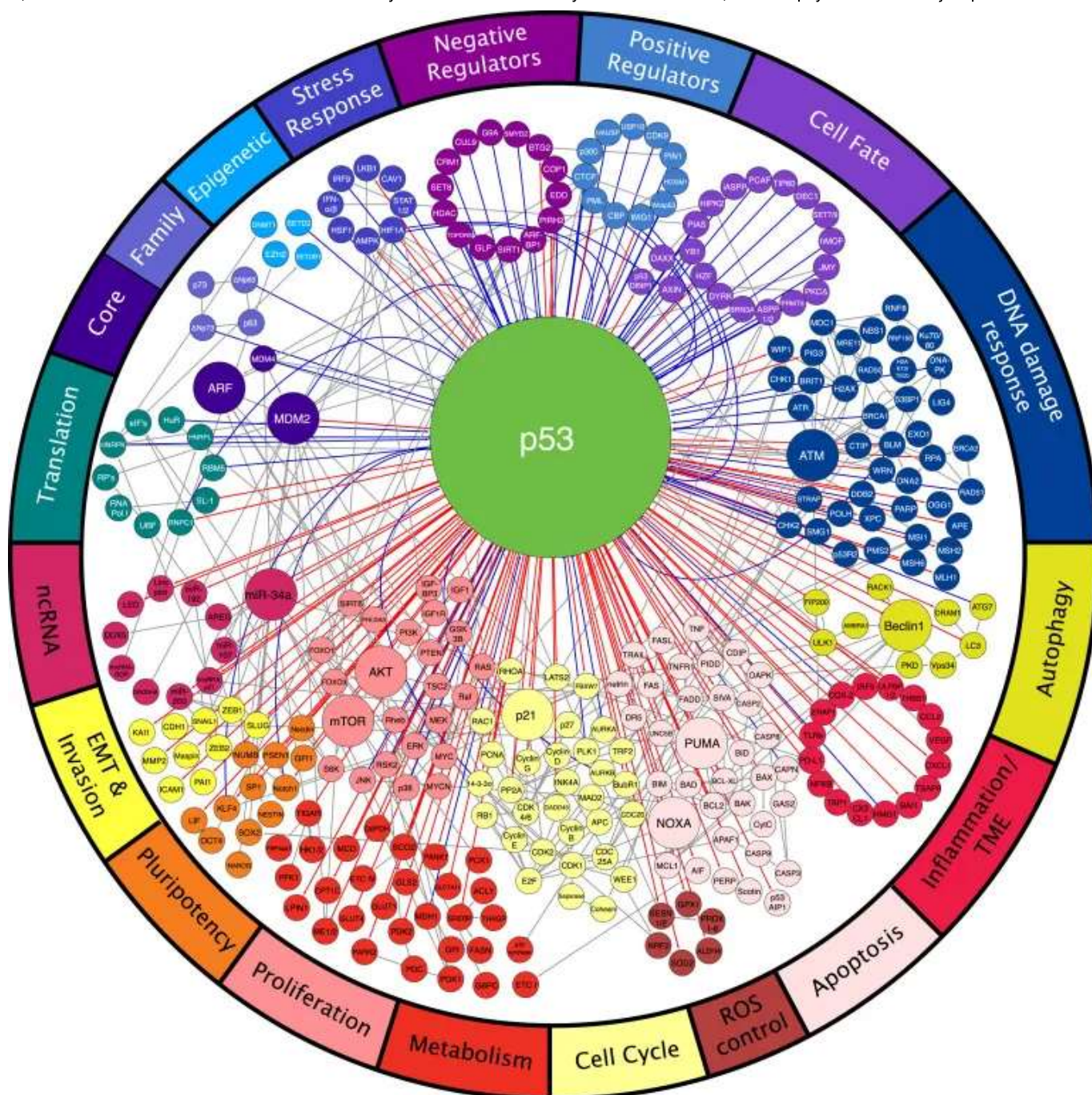


Figure 2: p53 and its derivatives [8].

Here are a few visualizations from our exploratory data analysis of this dataset.

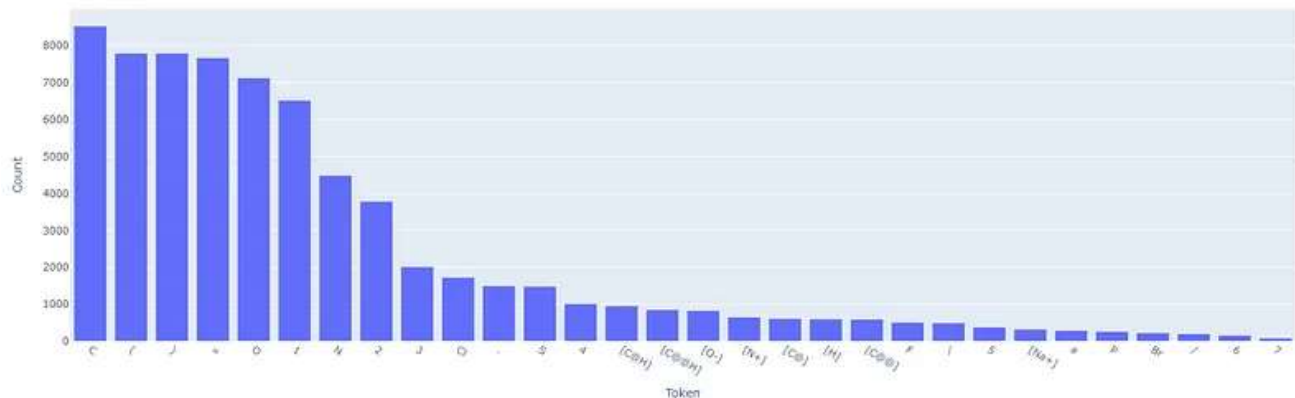


Figure 3 shows the top 30 molecules in terms of occurrence at least once in a given molecule. In descending order, the 8 most frequently occurring SMILES tokens are carbon, the opening and closing statements of a new branch, the double bond, oxygen, a formal charge of plus or minus 1, nitrogen, and the formal charge of plus or minus 2. This makes intuitive sense: the atoms listed are three of the most commonly occurring elements in organic compounds and the other tokens represent fundamental concepts in chemistry.

Figure 4 shows the top 30 molecules in terms of total occurrence in the dataset, note the logarithmically scaled vertical axis. We see that carbon’s

dominance in this figure is more pronounced than the previous figure. This likely has to do with the presence of complex carbon structures, like rings, where multiple carbon atoms appear in succession. Additionally, the double bond occurs more frequently overall than the branching shapes even though it occurs less frequently in an at-least-once context. This likely has to do with the fact that while many molecules contain at least one branch, many molecules will contain multiple double bonds if they do contain any given the charges of atoms within the molecules.

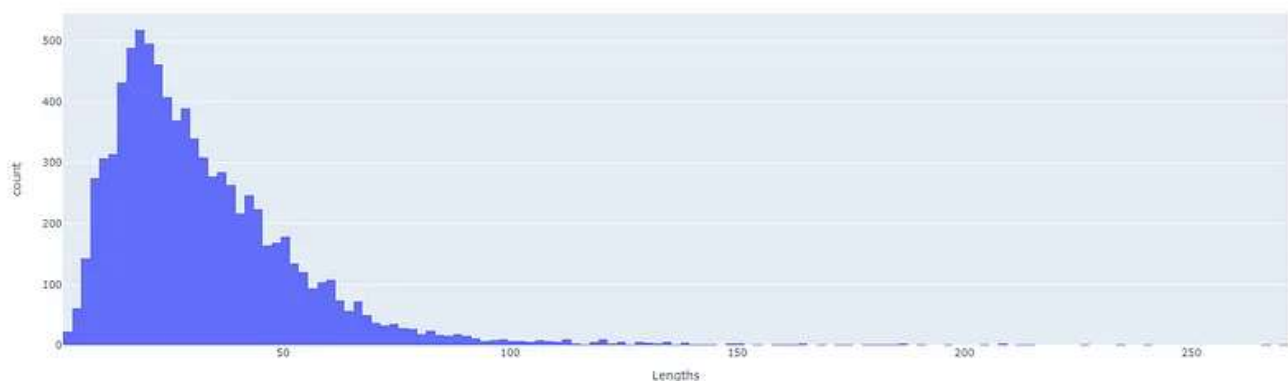


Figure 5: Distribution of sequence lengths.

Figure 5 shows the distribution of sequence lengths in terms of number of SMILES tokens. While we can see the distribution of lengths is centered around 20 tokens, the distribution is nonetheless heavily skewed in the positive direction. Given the highly variable character of sequence lengths and the presence of outliers in this respect, we'll have to add an additional preprocessing step that transforms the inputs into sequences of constant length. We'll discuss this step in more detail when we cover the baseline model.

Additionally, we created a few visualizations of the two-dimensional structures of molecules based on their SMILES sequences to get a sense of

how the mappings look:

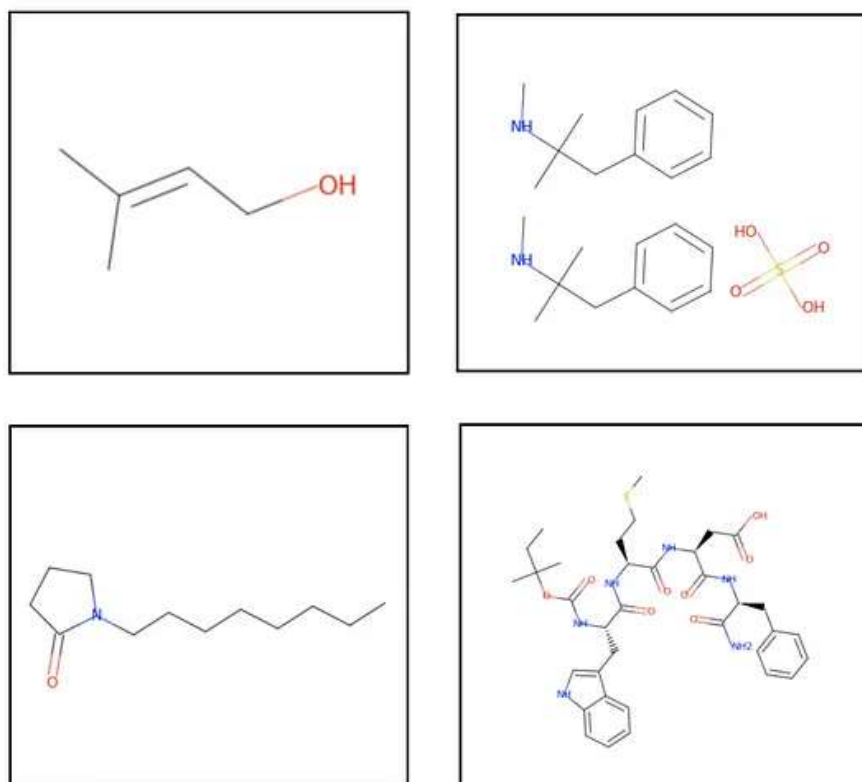


Figure 6: Visual representations of SMILES. Top left: CC(C)=CCO, top right: CNC(C)Cc1ccccc1.CNC(C)Cc1ccccc1.O=S(=O)(O)O, bottom left: CCCCCCCCN1CCCC1=O, bottom right: CCC(C)@OC(=O)N[C@@H](Cc1c[nH]c2ccccc12)C(=O)N[C@@H](CCSC)C(=O)N[C@@H](CC(=O)O)C(=O)N[C@@H](Cc1ccccc1)C(N)=O

Preprocessing & Baseline Model

We begin by preprocessing our sequence data.

Our first preprocessing step was dealing with unseen tokens in the test set, that are not present in the training set. With the help of Vinoj John Hosan, we create a new preprocessing tool called `LabelEncoderExt`, a build on `LabelEncoder` that is able to handle new classes (unseen tokens). Overall, `LabelEncoderExt` works by replacing an unseen token with `<Unknown>` by

being passed through fit and transform functions. More specifically, the fit function takes in a list of data, fits the encoder using all the unique values, and adds “Unknown” to the label encoder list. The transform function takes a data list, transforms it to an id list by assigning all new values the class “Unknown”. We use LabelEncoderExt to encode the tokenized training and test data. To do this we traverse through the tokenized training samples and store a NumPy array of integer-encoded samples [5].

Next we preprocessed our encoded training and test data through padding. While most sequences are of a length less than 60, the longest sequence in our training data is 240 characters. We pad the encoded training and test data to be length 240 (we add 0s to each sequence until it is the desired length). Our padded and encoded data becomes X_train, our padded and encoded test data becomes X_test, and our y_train and y_test are the target columns in the original dataset (0 for non-toxic, 1 for toxic).

Before implementing our baseline model, we initial Weights and Biases (wandb), which allows us to display training progress and diagnose issues earlier on. We create a sequential model with a Masking, Embedding, SimpleRNN and one Dense layer.

Figure 7: Code to build model.

Model: "sequential_2"

| Layer (type) | Output Shape | Param # |
|--------------------------|------------------|---------|
| masking_2 (Masking) | (None, 240) | 0 |
| embedding_2 (Embedding) | (None, 240, 128) | 16384 |
| simple_rnn_2 (SimpleRNN) | (None, 100) | 22900 |
| dense_2 (Dense) | (None, 1) | 101 |
| Total params: 39,385 | | |
| Trainable params: 39,385 | | |
| Non-trainable params: 0 | | |

Figure 8: Summary of Model.

The **masking layer** “masks” the value of 0, telling the RNN to ignore the padding characters.

The **embedding layer** is the first hidden layer in a network, where `input_dim` is the length of our vocabulary (length of `train_alphabet` + 1 to account for the unknown token), the `output_dim` is the vector space size in which words will be embedded (length of `train_alphabet` + 1), and the `input_length` is the length of input sequences (240) [6].

The **SimpleRNN** layer is a fully connected RNN where the output is fed back into the input, we chose a dimensionality of 100 as the output space.

We end with a **dense** layer that has an output size of 1, and uses sigmoid as the activation.



Figure 9: Compiling and Training Baseline Model.

To compile our model, we use `binary_crossentropy` as the loss, `SGD` as the optimizer, and both accuracy and AUC as the metrics. After training our model on 10 epochs with a batch size of 32, we get a `val_accuracy` of 0.8955 and a `val_auc` of 0.500.

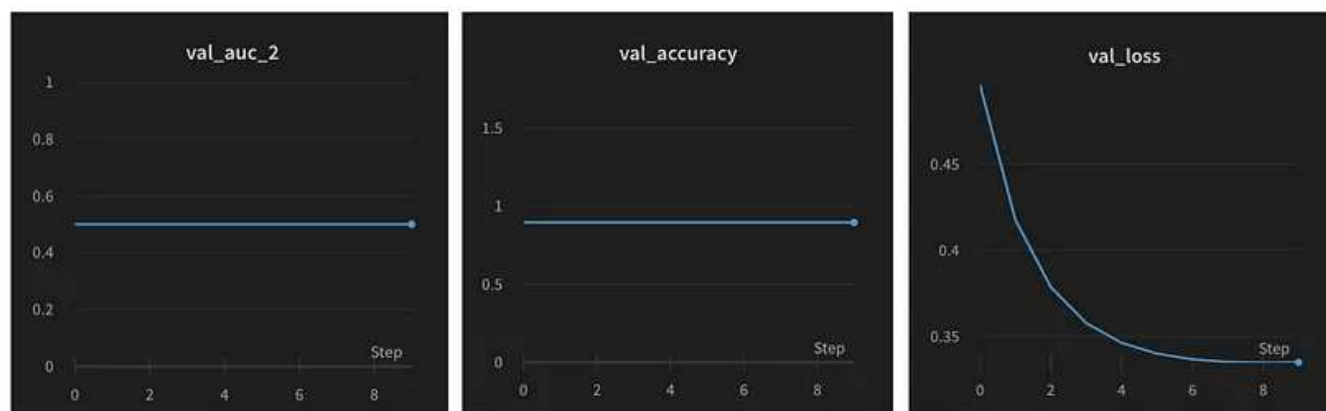


Figure 10: Training graphs from wandb.

```
Epoch 1/10
270/270 [=====] - 25s 88ms/step - loss: 0.5979 - accuracy: 0.9377 - auc_2: 0.5060 - val_loss: 0.4960 - val_accuracy: 0.8955 - val_auc_2: 0.5000
Epoch 2/10
270/270 [=====] - 23s 87ms/step - loss: 0.4419 - accuracy: 0.9377 - auc_2: 0.4921 - val_loss: 0.4176 - val_accuracy: 0.8955 - val_auc_2: 0.5000
Epoch 3/10
270/270 [=====] - 24s 88ms/step - loss: 0.3586 - accuracy: 0.9396 - auc_2: 0.4972 - val_loss: 0.3786 - val_accuracy: 0.8955 - val_auc_2: 0.5000
Epoch 4/10
270/270 [=====] - 23s 87ms/step - loss: 0.3143 - accuracy: 0.9387 - auc_2: 0.5181 - val_loss: 0.3578 - val_accuracy: 0.8955 - val_auc_2: 0.5000
Epoch 5/10
270/270 [=====] - 23s 87ms/step - loss: 0.2940 - accuracy: 0.9344 - auc_2: 0.5118 - val_loss: 0.3463 - val_accuracy: 0.8955 - val_auc_2: 0.5000
Epoch 6/10
270/270 [=====] - 23s 86ms/step - loss: 0.2693 - accuracy: 0.9391 - auc_2: 0.4993 - val_loss: 0.3401 - val_accuracy: 0.8955 - val_auc_2: 0.5000
Epoch 7/10
270/270 [=====] - 23s 86ms/step - loss: 0.2608 - accuracy: 0.9376 - auc_2: 0.5127 - val_loss: 0.3368 - val_accuracy: 0.8955 - val_auc_2: 0.5000
Epoch 8/10
270/270 [=====] - 24s 87ms/step - loss: 0.2537 - accuracy: 0.9374 - auc_2: 0.5008 - val_loss: 0.3353 - val_accuracy: 0.8955 - val_auc_2: 0.5000
Epoch 9/10
270/270 [=====] - 23s 86ms/step - loss: 0.2389 - accuracy: 0.9419 - auc_2: 0.4994 - val_loss: 0.3348 - val_accuracy: 0.8955 - val_auc_2: 0.5000
Epoch 10/10
270/270 [=====] - 23s 87ms/step - loss: 0.2535 - accuracy: 0.9333 - auc_2: 0.5141 - val_loss: 0.3350 - val_accuracy: 0.8955 - val_auc_2: 0.5000
```

Figure 11: Keras training history.

Toxicity is much less common than non-toxicity samples in our data (at a ratio of 9:1). Thus, it is not surprising that the data is very skewed in favor of non-toxicity. To combat for this and get an informative judge of classification quality, we use AUROC (resilient to skewed data). Using AUROC, we can interpret our validation score of 0.5 as a random classifier.

Conclusion & Next Steps

Looking forward, the most immediate next steps are to create a stronger model to improve on our baseline. The main scoring metric we are focusing on is the AUROC, as is the convention when dealing with two-class classification and unbalanced data. In terms of architectural next steps, we

will explore different methods of creating embeddings of our SMILES sequences possibly using techniques like byte-pair encoding or other methods of alphabet extension, as well as LSTMs and transformers to learn more sophisticated embeddings based on context and attention.

The current baseline model is extremely simplistic including only an embedding layer (for the purposes of data reshaping), simple RNN layer, and dense layer. Our next steps will explore the use of LSTMs and other carefully selected model parameters.

Outside of simple RNNs, we are planning to explore CNNs, transformers, and attention-based language models. Attention based language models like BERT and GPT-3 have recently emerged as interesting modes of classification or seq2seq prediction for SMILES data. The progress of deep learning algorithms has yielded great progress in the field of molecular property classification. To that end, the goal of our project is to come up with innovative ways of encoding SMILES as sequences for predicting toxicity.

All code can be found in our [GitHub](#) repository.

Blog Post 2: [here](#)!

Blog Post 3: [here](#)!

References

- [1] <https://tox21.gov/overview/about-tox21/>
- [2] <https://pypi.org/project/SmilesPE/>
- [3] <http://web.stanford.edu/class/cs224w/project/26424513.pdf>
- [4] http://cs230.stanford.edu/projects_spring_2019/reports/18677586.pdf
- [5] <https://stackoverflow.com/questions/21057621/sklearn-labelencoder-with-never-seen-before-values>

- [6] <https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/#:~:text=The%20Embedding%20layer%20is%20defined,vocabulary%20would%20be%2011%20words>
- [7] [scirp.org/\(S\(i43dyn45teexjx455qlt3d2q\)\)/reference/ReferencesPapers.aspx?ReferenceID=230944](https://scirp.org/(S(i43dyn45teexjx455qlt3d2q))/reference/ReferencesPapers.aspx?ReferenceID=230944)
- [8] [https://www.cell.com/cell/comments/S0092-8674\(17\)30953-4](https://www.cell.com/cell/comments/S0092-8674(17)30953-4)
- [9] https://en.wikipedia.org/wiki/Simplified_molecular-input_line-entry_system
- [10] <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6311897/>

Data Science

Chemistry

Deep Learning

Recurrent Neural Network

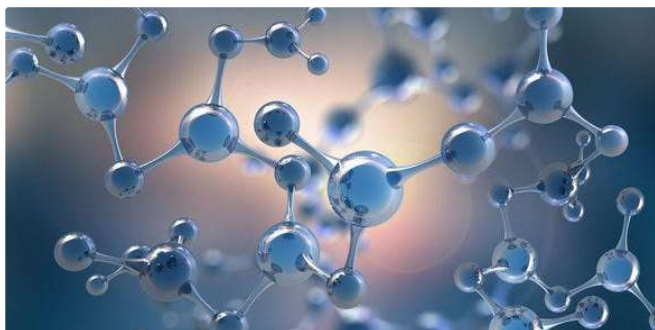
NLP

**Written by 2040 Final Project**

3 Followers

Follow

**More from 2040 Final Project**

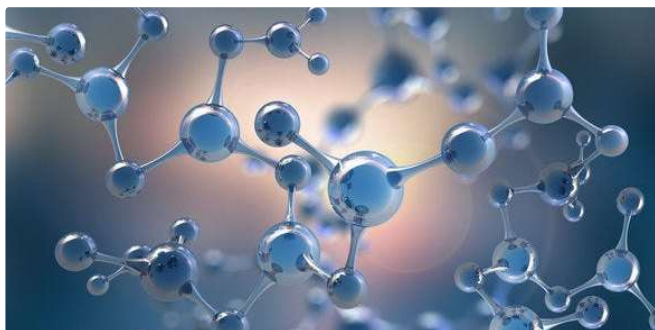


F 2040 Final Project

SMILES Toxicity Prediction

Part Two By Cameron Webster, Jillian Green, Chris Rohlicek, and Akshay Shah

5 min read · Apr 12, 2021



F 2040 Final Project

SMILES Toxicity Prediction

Part Three By Cameron Webster, Jillian Green, Chris Rohlicek, and Akshay Shah

9 min read · Apr 20, 2021



See all from 2040 Final Project

Recommended from Medium





Epiphyte Corp

Revolutionizing Agritech with IoT: A New Era for Agriculture

Introduction: The Dawn of a Digital Farming Revolution

3 min read · Dec 12



62



1



...



Tushar Aggarwal in Python in Plain English

LIME : Explaining Machine Learning Models with Confidence

Machine learning models have become increasingly complex and accurate over the...

7 min read · Nov 2



38



...

Lists



Predictive Modeling w/ Python

20 stories · 705 saves



Practical Guides to Machine Learning

10 stories · 805 saves



Natural Language Processing

1006 stories · 489 saves



data science and AI

38 stories · 13 saves



The Journey

Forecasting of Crop Yield using Remote Sensing Data, Agrarian...

Hello...Hello...! I am a ML researcher & I always get surprised how much of AI & ML can be...

3 min read · Jun 30



Kaan Erden

Forecasting Success: Employee Performance Prediction

In this project, we will explore a dataset comprising different attributes relevant to...

9 min read · Sep 6



36



1



Denisy Maulidya Azahra

Forecast Sales Using Machine Learning (Case : Store Item...

Step-by-Step Guide to Forecast Sales Using Python and Reinforcement Learning

16 min read · Jul 30



3



128



n models

pre-trained on large amounts of data so that you can perform a wide range of tasks, such as text generation, text summarization and question answering. Select the model you'd like to power :

k Models

always active, and you can start a conversation with them at any time.

i 2

gpt

a Titan

by OpenAI

AI Claude Instant

by Anthropic

AI Jurassic-2 Ultra

by AI21 Labs

c-2 Mid

gpt

aker JumpStart Models

available open-source models in Amazon SageMaker JumpStart. Start them to power your conversation.

-7B-Instruct

by Hugging Face

Falcon-40B-Instruct

by Technology Innovation Institute

MPT-7B-Instruct

by MosaicML



Meenakshisundaram Thandavarayan

Building Generative AI applications has never been this easy.

From comparing model performance to building RAG based application, Amazon...

3 min read · Oct 27



3



128



See more recommendations