COURSE PROJECT:

DEEP LEARNING TO PREDICT COVID19 ANTIVIRAL DRUGS

SYSC 4906 - ASSIGNMENT 3

Kevin Dick
Systems & Computer Engineering
Carleton University
Otttawa, Canada
kevin.dick@carleton.ca

James R. Green
Systems & Computer Engineering
Carleton University
Otttawa, Canada
jrgreen@sce.carleton.ca

December 1, 2020

ABSTRACT

*You (and your **partner**) are the lead machine learning researcher(s) among an interdisciplinary team of scientists and virologists during the COVID-19 pandemic. Your team has decided to dedicate their time and resources to identifying candidate anti-viral drugs that may reduce severity of COVID-19 symptoms and the viruses' mortality rate. Unfortunately, your team only has the resources to test a handful of candidate anti-viral drugs among the millions that exist. They turn to you and your team of machine learning engineers and data scientists to generate a deep learning model that can predict which of the millions of drugs is most likely to cure COVID-19 and provide them with a short list of candidate drugs to test in the lab. Time is ticking! Lives are at stake! Can you help bring an end to the pandemic?!*

Assignments 3 (A3) requires that you implement the deep machine learning models that you proposed in A2 for the task of **Drug-Target Interaction** (DTI) prediction. A3 has three deliverables: a **model development Notebook** (`G#_train.ipynb`) that contains all of the sequential code and markdown information used to generate your final model; the **final model** (`G#_model.pkl`) in a serialized Pickle format; and a **Python script/module** (`G#_predict.py`) that is used to generate predictions using your trained model. Details of each deliverable are described below.

As part of a **project reach goal**, the most performant models *may* be combined into an *ensemble predictor* and used to generate predictions for the **49,437** drugs listed in the CAS COVID-19 Antiviral Candidate Compounds Dataset and results *may* be prepared as part of an academic research paper.

***K*eywords** Drug-Target Interaction Prediction · COVID19 · Deep Learning

A3 Due Date: Friday, December 11th, 2020 by 23:59 EST

Assignment 3 Overview

This section outlines general details to consider while preparing your final DTI predictor. It first covers the submitted filename convention, then the datasets you are permitted to use to train your model, then some general advice on generating embeddings using the DeepPurpose framework, and finally the evaluation method for your final predictor.

All Files Named using your Assigned Group Id

You must save your submitted files using the "group number" id that was assigned to you in the A2 feedback. For example, if your feedback document was "G3-Feedback.docx", your group id is "G3" and your model should be saved with the filename `G3_model.pkl`. Similarly, if your feedback document was "G29-Feedback.docx", your group id is "G29" and your model should be saved with the filename `G29_model.pkl`.

Ensure that all instance of “G#" or “G0" are updated to your group id wherever they appear; these are simply placeholders for the template.

Permitted Datasets for Training your Model

The only datasets you may use to train your model are the DAVIS and BindingDB datasets described in the A2 instructions. You are **strictly forbidden** from using the **KIBA dataset** to train your model for the following reasons:

- **Limited Computational Resources**: Given that Google Colab has limited RAM and Disk space, the KIBA dataset is too large to be used effectively to train your models. Please only consider using the DAVIS and/or BindingDB datasets available in the DeepPurpose toolkit.
- **Fair Competition:** as your models will be evaluated against your peers, a team leveraging all three datasets will have a possibly unfair advantage.
- **Independent Test Samples:** A subset of SMILE-Target pairs may be used in the independent evaluation of your methods and it is important that none of these samples bias your models (known as “data leakage").

Therefore, only use the DAVIS and/or BindingDB datasets from the DeepPurpose framework. When loading the datasets, the following options should be used (they are the defaults): `y = 'Kd', binary = False, convert_to_log = True`. External or auxiliary information may also be used, however they must be well described within your training notebook.

Using the DeepPurpose Framework to Encode Sequences

The `utils` module in the DeepPurpose framework contains all of the methods needed to encode the SMILE and target sequences into a numerical representation that can then be used to and evaluate a deep learning model (e.g. in Keras). As you might have seen from the A2 DeepPurpose questions, the CNN encoding method returns a **list of sequence characters** instead of the **one-hot encoding** of those sequences. In order to generate the numerical representation of drug or protein sequence, you can call the two following methods where `x` is each character for the input drug SMILE or protein sequence respectively:

- `drug_2_embed(x)` to one-hot encode a character of the Drug SMILE
- `protein_2_embed(x)` to one-hot encode a character of the Protein sequence

To one-hot encode a single sequence, you can use the following:

```
from DeepPurpose.utils import *
print(f'First Drug SMILE: {X_drug[0]}\nFirst Target seq: {X_target[0]}')

# Generate the one-hot encoding for each sequence
drug_repr   = [drug_2_embed(x) for x in X_drug[0]]
target_repr = [protein_2_embed(x) for x in X_target[0]]
print(f'First Drug Repr.: {drug_repr}\nFirst Target Repr.: {target_repr}')
```

Depending on your implementation, you may need to modify the resulting list of Numpy arrays into another shape (consult StackOverflow if stuck).

All other representations can be generated using the following method for drug SMILES `encode_drug(df_data, drug_encoding)` and for protein amino acid sequences `encode_protein(df_data, target_encoding)` by simply specifying one of the available string types for `drug_encoding` and `target_encoding`, respectively.

To reiterate an excellent suggestion from Ms. Aisha Robinson during T10: to avoid having to **repeatedly setup the DeepPurpose framework** every time the runtime is reconnected, consider **downloading your dataset of representations** after they have been encoded using DeepPurpose and uploading them whenever you implement and train your model (e.g. using Keras). That is, you would perform the following steps:

1. Setup DeepPurpose
2. Load the DAVIS and/or BindingDB dataset
3. Split your dataset into train, validation, and test sets

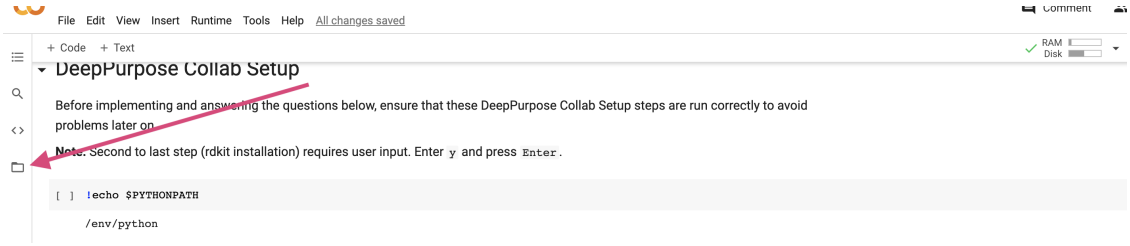

Figure 1: Folder icon to click on to download a file from Google Colab.

4. Used DeepPurpose to generate the representations for each sequence
5. Serialize and download your train, validation, and test sets (e.g. `G#_data_train.pkl`, `G#_data_vali.pkl`, `G#_data_test.pkl`)
6. Download the three sets for later use in implementing and training your model

This will ensure that you don't need to re-setup the DeepPurpose framework each time. The following section describes serialization and downloading/uploading files in Google Colab.

Pickling & Downloading your Model from Google Colab

To serialize your model (convert it into a binarized format that can be later loaded and used), the Python Pickle (Rick ;) library can be used. An example usage to save/"dump" and load/read of an object is demonstrated **at this link.**

To download a file from Google Colab, first click on the folder icon, as shown in Figure 1, then navigate to your file, right-click the file and select "Download". **This video** also shows you how to do this.

To upload data to a Google Colab session, as in the case of uploading the precomputed SMILE and Target representation, an "Upload to Google Session" option is also available, as depicted in Figure 2.

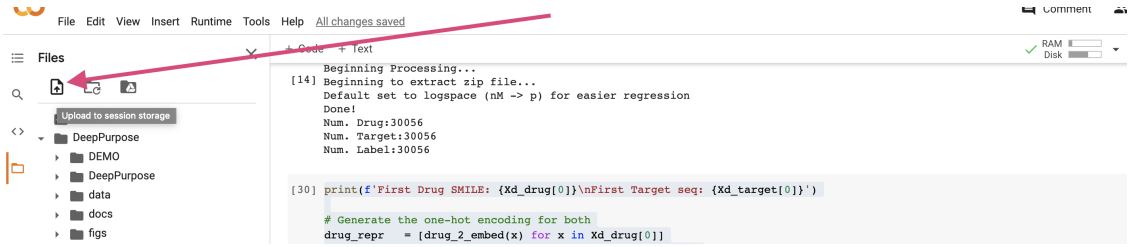

Figure 2: Icon to click on to upload file(s) to Google Colab.

Final Model Evaluation

Your final model performance will be evaluated using Root Mean Squared Error (RMSE) and therefore **must** output a continuous value for the predicted binding affinity, K_d. To ensure that your regression models converge quicker, the binding affinities in your training datasets should be **log-transformed** and the **predicted output binding affinity must also be log-transformed.**

$$\text{RMSE} = \sqrt{\left(\frac{1}{n}\right)\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{1}$$

Your model's performance will be ranked against your peers as well as against the DeepDTA model which will partially contribute to your A3 grade. When evaluating your model, you can compute RMSE using **scikit-learn mean_square_error function** with option `squared=False` for the RMSE.

Assignment 3 Deliverables

1. Model Training Jupyter Notebook; Template: "G#_train.ipynb"

You must submit the notebook (`.ipynb` file) used to train, evaluate, refine, and save your final model. This notebook does not have a prescribed structure; you are free to organize it in the way that makes the most sense to you while implementing and generating your final model. It must, however, adhere to the following criteria:

1. **Code + Markdown:** it should have markdown cells explaining what the following code cells are for (see tutorial notebooks as an example).
2. **Dataset Processing:** it should clearly demonstrate what dataseet(s) was loaded, how it was split, and how it was encoded.
3. **Model Implementation:** it must include the implementation of your model and training outcomes.
4. **Evaluation & Experiments:** it must contain a demonstration of your model evaluation and any experiments performed (e.g. variation of hyperparameters and/or representations). To determine how "good" your model is, consider comparing its RMSE performance to the DeepDTA RMSE on your test dataset.
5. **Model Serialization:** it must show which of the final models was serialized for submission.

Grading: The notebook will be graded based on *completeness* (it covers all requisite criteria), how *well organized* it is, the *logical explanation* of the experiments described/demonstrated, and should *free of typos*. The file should be saved as `G#_train.ipynb` where `G#` is the "group id" you were assigned based on the feedback filename from A2.

2. Your Final Model in Serialized Format, "G#_model.pkl"

The final model developed in the previous step should be serialized in Pickle format and named according to your group id: `G#_model.pkl`.

Grading: At minimum, your model must "work", that is, it should output a float-type value based on an input pair of drug-target sequences. Your grade will then be determined based on its *relative performance* compared to your peers (the top-ranked method achieving a perfect grade and then proportionally reduced by sorted rank). Note: this only accounts for a fraction of the total A3 grade (exact weight TBD).

3. A Model Prediction Script; Template: "G#_predict.py"

A Python script created from the template `G0_predict.py` that implements tthe necessary logic to generate predictions using your trained model. You must address each of the `TODOs` in the script to implement the following methods:

- `encode_pairs(df_pairs)`: An input dataframe of drug SMILES and protein sequences is encoded into the numerical representation required by the `<G#_model.pkl>` model to generate predictions.
- `predict_pairs(df_encode)`: For all pairs in a dataframe containing drug and target encodings, the pre-trained model is used to generate a predicted binding affinity (float value) that is added as a column to the dataframe named "predicted".

Additional details on the expected input/output shape and column names of the dataframes are outlined in the doc-strings within the template.

Grading: At minimum, your script must "work", that is, the `encode_pairs` method must correctly generate your necessary embeddings and the `predict_pairs` method must generate a column of predicted value using the loaded model. Note: it is strongly recommended that you test the functionality of this script yourself with a handful of example drug-target sequences. This script will be imported as a module when evaluating your model, for example:

```
import G3_predict as g3

df = load_evaluation_pairs()
df = g3.encode_pairs(df)
df = g3.predict_pairs(df)

# Calculate RMSE between "Label" and "predicted"
g3_rmse = calculate_RMSE(df)
```

Zero-Tolerance for Competition "Shortcuts": All submitted code will be reviewed for any "shortcuts"/hacks that might result in an overstated performance. For example, if you implement `df_results["predicted"] = df_results["Label"]` within `predict_pairs` resulting in a perfect RMSE=0, your team will be disqualified and receive a grade of 0 in the competition.

The world is counting on you to save lives! Good luck!