CAFA 5 Protein Function Prediction

Name - Sahil Tomar
Roll Number - 210898
Department - BSBE
IITK Email - stomar21@iitk.ac.in
Kaggle ID - stomar21
Kaggle Leaderboard Rank - 459 / 735

Introduction to the problem - In the problem, we are given a train_sequences.fasta file which contains the id, gene name for the protein and its amino-acid sequence. We have also been given the file train_taxonomy.tsv which contains protein id and taxonomy id to know what species the protein is from. Also given, train_terms.tsv which contains labels for the unique protein ids, which serve as ground truth labels as this is a multi-label classifier problem. Testsuperset.fasta contains the protein sequences from which the test set will be chosen and we need to annotate or label those protein sequences with a function. testsuperset-taxon-list.tsv contains the taxon IDs for the proteins in the test superset. We need to annotate the protein sequences given in the test superset with one or more GO-terms and then our submission will be according to Information Accretion for each term. So, according to me, this is a multilabel classification problem where we need to relate protein sequences to one/many GO-terms. Gene Ontology is a concept hierarchy which describes the biological function of genes and gene products as Directed Acyclic Graph.GO contains three subgraphs (sub-ontologies): Molecular Function, Biological Processes and Cellular Components.
GO has a DAG with nodes that are functional descriptors (terms or classes), connected by relations between them such as is_a, is_part_of, positively_regulates, negatively_regulates, etc. GO includes annotations by linking specific gene products to GO terms. This allows the connection between genes and GO-terms for deriving organism-specific information. It is common for genes to be associated with multiple terms simultaneously. As a result of the hierarchical relationships between the terms, the annotations of these genes propagate upwards to higher-level terms, ultimately resulting in the formation of gene sets associated with the nodes. Therefore, a subset of one or more sub-ontologies represents the protein's function. A protein may get annotated by one or more terms from the same or different sub-ontologies. We will use the Gene Ontology data to predict protein function terms and we can refer to the GO data to get a sense of what functions are categorised how and then come up with better solutions to the problem.

Literature Review -

- https://advaitabio.com/faq-items/understanding-gene-ontology/ - http://geneontology.org/docs/ontology-documentation/
  Used to understand what Gene Ontology is. GO is described in the above section on ‘Introduction to the problem’.
- https://storage.googleapis.com/kaggle-forum-message-attachments/2229862/19019/Introduction_to_protein_prediction-4.pdf
  Introduction to What is ‘Protein Function Prediction’ and what happened in CAFA 2, CAFA 3 and CAFA 4.
- https://towardsdatascience.com/feature-extraction-techniques-d619b56e31be -

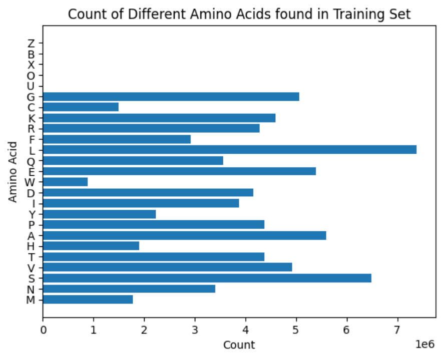
Used to study about various dimension reduction or feature extraction techniques
Includes

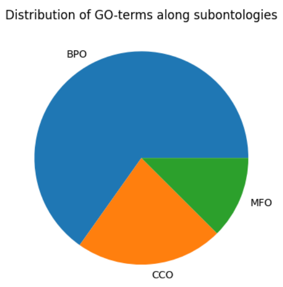
1. Principal Components Analysis(PCA) - where we take as input our original data and try to find a combination of the input features which can best summarize the original data distribution so that to reduce its original dimensions.
2. Independent Components Analysis (ICA) - ICA is a linear dimensionality reduction method which takes as input data a mixture of independent components and it aims to correctly identify each of them (deleting all the unnecessary noise).
3. Locally Linear Embedding (LLE) - Locally Linear Embedding is a dimensionality reduction technique based on Manifold Learning. A Manifold is an object of D dimensions which is embedded in a higher-dimensional space. Manifold Learning aims then to make this object representable in its original D dimensions instead of being represented in an unnecessary greater space.
4. Autoencoders - Autoencoders are a family of Machine Learning algorithms which can be used as a dimensionality reduction technique. The main difference between Autoencoders and other dimensionality reduction techniques is that Autoencoders use non-linear transformations to project data from a high dimension to a lower one.

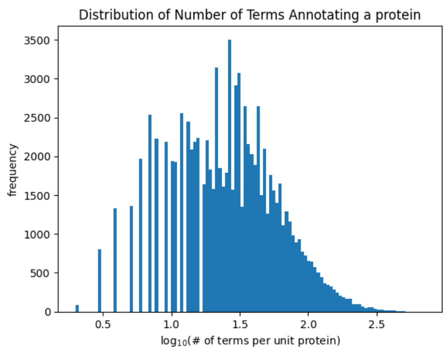
- https://www.coursera.org/learn/convolutional-neural-networks/home/week/2 - The course helped me in understanding what CNNs are and how to implement them for image processing.
- https://medium.com/voice-tech-podcast/text-classification-using-cnn-9ade8155dfb9 - Gave me a brief introduction about how to use CNN for text classification, as the course was about image processing mainly.
- https://wandb.ai/site/tutorial/text-classification-using-cnns - I used the resources and the video in this article to learn how to create and then how to use various text embeddings and I chose **Charachter One-Hot Encodings** for my project.
- https://towardsdatascience.com/protein-sequence-classification-99c80d0ad2df - This article explores the use of CNN and one-hot encodings to make LSTM and CNNs to identify protein families using the Pfam Database. I used it to study the architecture of the model and bring up an appropriate architecture for CAFA5 GO-labels classification.

Experimentation

- https://www.kaggle.com/code/stomar21/eda-cafa-5-protein-function-analysis - **(THIS NOTEBOOK WAS NOT USED FOR ANY SUBMISSION TO KAGGLE)**

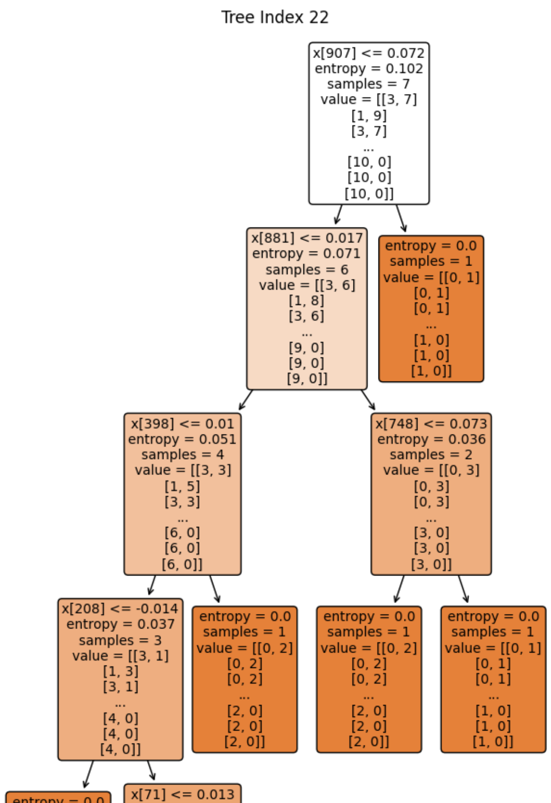






- In this notebook I tried to do just a basic Exploratory Data Analysis of the Data.
- I used OBONET library in python to actually map the GO - Graph for a specific GO-term and the nodes within a particular range of the node.
- I then plotted the Number of protein sequences in the training database, then plotted the distribution of the amino acids in the sequences, and plotted the counts of various amino acids in the proteins.
- I plotted and found relationships in the GO-labels as in how many proteins were in which particular GO-sub ontology
- Then I carried out protein-per-GO-term and GO-terms-per-protein analysis where I plotted or calculated the following - number and distributions of proteins annotated by the same GO-term and same for the number of GO-terms annotating a particular protein.
- I also performed the above analysis group-wise where every group was one of the three sub-ontologies.
- Lastly I also took a look at the weights assigned to each of the GO-labels by the Information Accreditation criterion.
- Some of the interesting facts and figures about the data is noted after most of the analyses in the notebook

- https://www.kaggle.com/code/stomar21/cafa5-randomforest -
(THIS NOTEBOOK WAS NOT USED FOR ANY SUBMISSION TO KAGGLE)

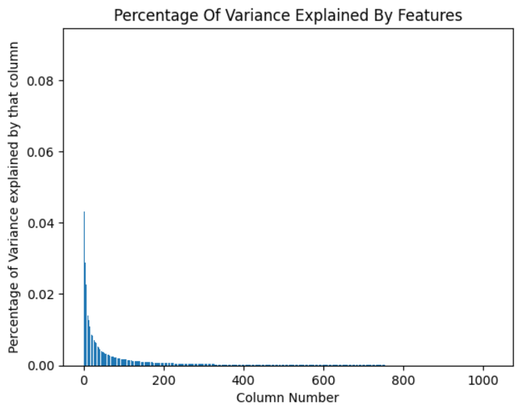


- In this notebook, I tried to implement a RandomForestClassifier to the protein data in order to get the label predictions.
- The first step, however, was to prepare feature matrices out of the data and then create label data for the proteins. Here, I took the help of t6_embeds Dataset available at Kaggle. This dataset has created 1024 - features long embeddings for all the proteins using the t5 algorithm.
- The embeddings needed labels for GO-terms which

were not provided in the t5_embeds. I chose to take only the top 1500 GO-terms as labels. This decision was taken as the top 1500 terms covered most of the Dataset, which was discovered in the earlier EDA steps.

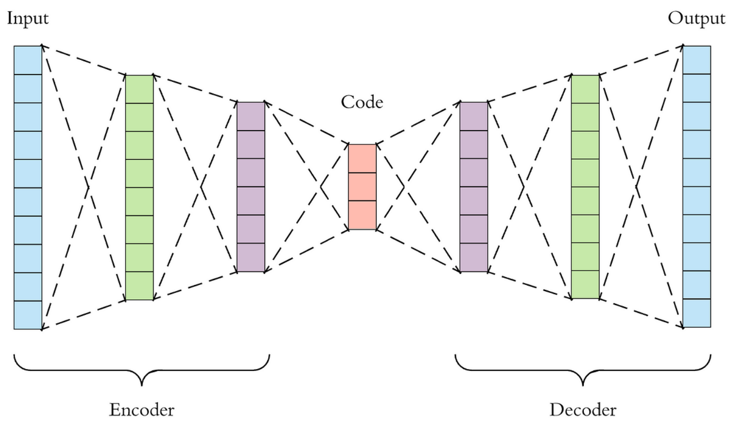
  - I then manually created a dataset of 142246 x 1024 shape, which contained OneHotEncodings of the labels. 0 meant that the GO term was not annotating the protein while 1 meant that it was.
  - The t5_embeds and then the GO_labels data were too large and the GO_labels data took 15 minutes to create, so I stored it into a Private Kaggle Dataset and imported it as and when needed.
  - Now the dataset was still pretty heavy and RandomForest is a relatively involved algorithm so I had to use Kaggle GPU. The problem however is that Multi-Label Classification is not yet implementable with GPUs. So I decided to train each of the 1500 label columns separately and then again concatenate the resulting vectors.
  - After working for about a week or two, I was finally able to come up with a prediction for the labels with a hamming_loss of 0.016, which was very good.
  - I later realised that the predictions could not be submitted as they were merely predictions of the GO-annotations while Kaggle wanted us to include a probability value for all the terms.

- https://www.kaggle.com/code/stomar21/randomforest-cafa5-pca - **(THIS NOTEBOOK WAS NOT USED FOR ANY SUBMISSION TO KAGGLE)**



  - In this notebook I tried to implement A Principal Component Analysis on the data to shorten the feature embeddings from 1024 to 256 features. The decision for 256 was taken by first creating a PCA for all 1024 features and then realising that most of the variance in the data can be explained by about 250 features,
  - After performing the PCA analysis, I implemented my RandomForestClassifier again for the reduced Dataset and found a similar hamming_loss of 0.018. The training time, however, was reduced by a factor of 3-4.

- https://www.kaggle.com/code/stomar21/cafa5-autoencoder/ -

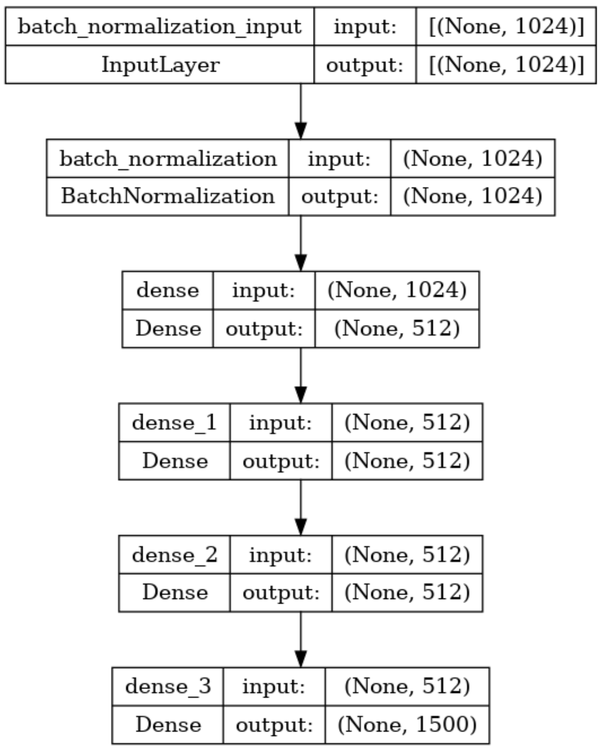


(THIS NOTEBOOK WAS NOT USED FOR ANY SUBMISSION TO KAGGLE)

- I also tried to implement an Autoencoder which is basically an Artificial Neural Network having two parts - An Encoder and A Decoder.
- I started by developing the encoder to have a structure of 1024, 512, 256 units with ReLU activation function and then the Decoder has 512, 1024 units with ReLU activation function. The decoder also has a Sigmoid output Layer.
- The Kaggle Notebook kept running into 'MemoryLimitExceeded Error' and I had to leave this project to start training a MultiLayerPerceptron which will automatically encode the data.

- https://www.kaggle.com/code/stomar21/cafa5-multilayerperceptron?scriptVersionId=133709154 ,https://www.kaggle.com/code/stomar21/cafa5-mlp?scriptVersionId=134236231 -

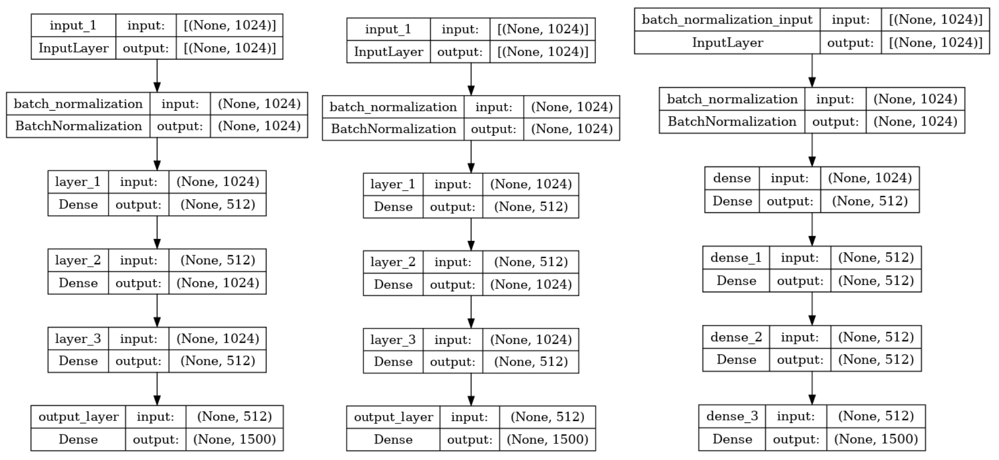
-	Hamming Loss	Kaggle Score
Submission	**0.01625**	**0.45579**


batch_normalization_input | input: | [(None, 1024)]
InputLayer | output: | [(None, 1024)]
batch_normalization | input: | (None, 1024)
BatchNormalization | output: | (None, 1024)
dense | input: | (None, 1024)
Dense | output: | (None, 512)
dense_1 | input: | (None, 512)
Dense | output: | (None, 512)
dense_2 | input: | (None, 512)
Dense | output: | (None, 512)
dense_3 | input: | (None, 512)
Dense | output: | (None, 1500)

- I trained my first submission-level MultiLayerPerceptron in the first Notebook link given. I created a simple architecture with An input BatchNormalisation Layer, then following are 3 Dense Layers with 512 units and ReLU activation Function. Lastly the ANN or MLP had an output layer with Sigmoid Activation.
- I had to train the model in the first notebook and then had to shift to the second Notebook in order to submit the notebook and the submission.csv (Massive 4.7GB file) to kaggle. I managed to get a Kaggle Submission Score of 0.45579, which gave me the leaderboard rank of 521.

    - By this time as I was running into ‘Memory Limit Exceeded Error’ many times, I figured out a way to compress my original GO_labels.csv file from 800MBs, which when saved as a numpy array with 64-bit floating point numbers took GBs of space. I compressed the file down to 24MB using h5py compressors and managed to save a lot of RAM space by loading only relevant parts of the data, that too in the form of numpy arrays of 8-bit integers whenever possible.
- https://www.kaggle.com/stomar21/kerastuner -

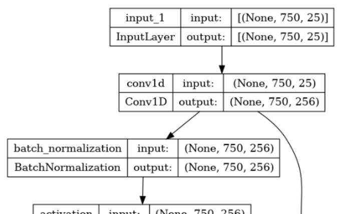
-	Hamming Loss	Kaggle Score
Submission	**0.02025**	**0.44999**


input_1 | InputLayer | input: [(None, 1024)] | output: [(None, 1024)]
batch_normalization | BatchNormalization | input: (None, 1024) | output: (None, 1024)
layer_1 | Dense | input: (None, 1024) | output: (None, 512)
layer_2 | Dense | input: (None, 512) | output: (None, 1024)
layer_3 | Dense | input: (None, 1024) | output: (None, 512)
output_layer | Dense | input: (None, 512) | output: (None, 1500)
input_1 | InputLayer | input: [(None, 1024)] | output: [(None, 1024)]
batch_normalization | BatchNormalization | input: (None, 1024) | output: (None, 1024)
layer_1 | Dense | input: (None, 1024) | output: (None, 512)
layer_2 | Dense | input: (None, 512) | output: (None, 1024)
layer_3 | Dense | input: (None, 1024) | output: (None, 512)
output_layer | Dense | input: (None, 512) | output: (None, 1500)
batch_normalization_input | InputLayer | input: [(None, 1024)] | output: [(None, 1024)]
batch_normalization | BatchNormalization | input: (None, 1024) | output: (None, 1024)
dense | Dense | input: (None, 1024) | output: (None, 512)
dense_1 | Dense | input: (None, 512) | output: (None, 512)
dense_2 | Dense | input: (None, 512) | output: (None, 512)
dense_3 | Dense | input: (None, 512) | output: (None, 1500)

    - I used the KerasTuner library to perform hyperparameter tuning for the hyperparameters - learning_rate, activation function and the number of units in the three NN layers.
    - I made a custom model-building function and then created a HyperBand object in kerastuner to make the model search for the best combination of parameters available to the model. I split the training data for this purpose.
    - After searching through many different combinations, it takes more than 45 minutes to search for combinations that make the binary accuracy minimum.
    - The search gave me a different model, which I again trained and submitted only to find out that the newer model performs worse than the earlier model, when submitted.
    - It turns out that kaggle is using a different metric than hamming_loss or binary_accuracy to evaluate the models. My newer model performs better on the binary_accuracy and hamming_loss but slightly lesser on the kaggle submission with a score of 0.44999 compared to the earlier score of 0.45579.

- https://www.kaggle.com/code/stomar21/cafa5-convolutional-neural-network

-	Hamming Loss	Kaggle Score
Submission	**0.02235**	**0.28651**


input_1 | InputLayer | input: [(None, 750, 25)] | output: [(None, 750, 25)]
conv1d | Conv1D | input: (None, 750, 25) | output: (None, 750, 256)
batch_normalization | BatchNormalization | input: (None, 750, 256) | output: (None, 750, 256)

- To create my custom one-hot encodings for the protein sequence data, I ran into a problem - CNNs require a fixed length vectors or 2D images when implementing convolutions, so I needed to take only a fixed length of protein sequences and had to increase the sequences somehow to get to a fixed length of the sequences and then one-hot encode it.
- I carefully looked at the distribution and decided to fix the lengths of the sequences to 750 as 80% of the proteins had sequences less than 750 and I could use almost all of the data available in the sequences of these proteins. So, I clipped (cut) the sequences which were more than 750 in length.
- I decided to repeat the sequences of smaller proteins many times until they reached 750 and again clipped it. Also, I rejected any proteins which had a length less than 150 because then the proteins would have exceeded more than 5 whole repetitions of the sequence, unnecessarily disturbing the distribution of the dataset.
- I used a scikit-learn OneHotEncoder to encode the sequences and implemented all of the above steps in a single class so as to make the code reusable for both training and testing data.
- I implemented a Convolutional Neural Network for protein functional annotation. I also implemented / used residual blocks in the model and used a dropout of 0.5 to reduce training time and over-fitting.
- Kaggle notebooks quite frequently run out of memory, and I had to use only 100000 proteins instead of the total 142246 proteins available for training. I split the dataset into 80% train / 10% dev / 10% test so as to gauge the performance of the model for which I wrote custom plotting and evaluating functions.
- My model gave around 0.022 hamming loss for train/dev/test sets and after saving model predictions and then submitting to kaggle, I got a Kaggle Score of 0.28651 which is not better than my MLP but given the method I used, it's still an acceptable score to me.