Enzyme Prediction Main Titel

Malte Alexander Weyrich, Thanh Truc Bui, Jan Philipp Hummel and Sofiia Kozoriz

Abstract

Motivation:
Results:
Availability:
Contact: name@bio.com
Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Abstract

The accurate prediction of enzyme commission numbers (EC numbers) is not only crucial for the classification and understanding of newly discovered enzymes but also for completing the annotation of already known enzymes. Therefore, developing a reliable method for predicting EC numbers is of great importance.

However, due to insufficient data, enzyme function prediction using machine learning remains challenging. In this paper, we propose several methods for predicting enzymes in three different problem categories. Each category goes further into depth of the enzyme classification system, starting with a binary classification (level 0) of enzymes and non-enzymes, followed by a multi-classification (level 1 & 2) of enzymes into main classes (e.g. EC X.-.-.-) and subclasses (e.g EC X.X.-.-). Throughout the developing of our models, we used a variety of different input features and machine learning algorithms, of which the best will be thoroughly reviewed in this paper (see table 1).

Table 1. A table showing the best-performing models per level

Level	Model	Weighted f1 score	MCC score
0	Random Forest	score	score
1	Feedforward Neuronal Network	score	score
2	Feedforward Neuronal Network	score	score

2 Introduction

Enzymes play a vital role as biological catalysts, facilitating essential biochemical reactions in organisms. Without enzyme catalysis, these reactions would either occur too slowly or be practically impossible. Predominantly proteins, enzymes are designated specific functions; for instance, Oxidoreductases drive redox reactions, and Isomerases convert molecules into their isomers. Therefore, a comprehensive understanding and a precise classification of enzymes are fundamental.

In the past, scientists attempted to categorize enzymes into groups and develop a logical rule set for naming. However, the efforts were hindered by ambiguity. A significant milestone occurred in 1956 with the establishment of an official international commission on enzyme classification (Wikipedia contributors (2023)). This marked the initiation of the contemporary enzyme classification system that forms the basis for our understanding of enzymes today.

In modern research, computational methods have become invaluable for enzyme classification; this shift has transformed the traditionally labor-intensive process. Machine learning and data-driven models, in particular, have assumed a leading role, holding the potential for enhanced accuracy and efficiency in annotating enzymes within vast genomic data.

Several effective methods have been developed to address this challenge. One notable example is DeepEC (Ryu *et al.* (2019)), a deep learning-based computational framework designed for the precise and high-throughput prediction of EC numbers for protein sequences. DeepEC employs three convolutional neural networks (CNNs) as a primary engine for EC number prediction and incorporates homology analysis for cases not classified by the CNNs.

Another noteworthy method is CLEAN (Yu *et al.* (2023a)), which stands for "contrastive learning–enabled enzyme annotation". CLEAN adopts a unique training objective, aiming to learn an embedding space for enzymes where the Euclidean distance reflects functional similarities. In CLEAN's task, sequences sharing the same EC number exhibit a small Euclidean distance, while sequences with different EC numbers have a larger distance. The model is trained using contrastive losses with supervision to achieve effective enzyme function prediction.

In our research, we developed several classification models for each level, using a variety of machine learning algorithms and input features. The best performing models per level will be thoroughly reviewed in this paper. The first is a binary classifier that determines whether a given protein functions as an enzyme. The second and third models are multi-classifiers, tasked with categorizing enzymes into different classes and subclasses based on their specific functionalities. Our primary objective is to remarkably outperform the random baseline classifiers in each category.

3 Methods

3.1 Binary classification of enzymes

In the course of assigning EC numbers to protein sequences, our initial step is to determine whether a given protein sequence is enzymatic or a non-enzymatic (level 0). To identify the most effective approach for this task, we conducted training the following binary classification models and compared their performance to select the optimal one:

(1) k-Nearest-Neighbors
(2) Random Forest
(3) Support Vector Machine

The chosen method for binary classification will be detailed in the subsequent section, while section 7.1 will outline the methods that were not selected.

3.2 Multi-classification of enzymes

For the second and third level, we used a feedforward neural network (FNN) to classify the enzymes into their respective main classes and subclasses. The FNN is a type of artificial neural network (ANN) that uses a series of layers to extract features from the input data and classify it into different categories, thus making it a suitable choice for our multi-classification task. The library used to build the FNN is Tensorflow, which is an open-source library for machine learning.

3.3 Data preprocessing in general

The protein data sets were taken from the UniProt database and were also used in the CLEAN publication on Enzyme function prediction (see Yu *et al.* (2023a)). The complete data pool of enzymes (called Split100 in Yu *et al.* (2023b)) contains 224.693 sequences. This pool was used to derive several smaller subsets (SplitX, $X \in \{10, 30, 50, 70\}$) where X stands for the amount of similarity allowed between the sequences. For reference, Split30 only contains sequences that share at most 30% similarity with each other. Since we want to avoid homology-based inference as much as possible while maintaining a reasonable amount of data to train our models on, we opted to use the Split30 dataset for our research, containing 9.186 enzymes. This subset was combined with a non-enzyme dataset holding 37.347 sequences, resulting in 46.533 sequences (see figure 1).

Note that we additionally removed all sequences containing an ambiguous amino acid, such as 'U' or 'O', from the dataset, as well as sequences with a length longer than 1022 amino acids. This is due to the fact that the esm2b model used for the embeddings only accepts sequences with a maximum length of 1022 amino acids.

3.4 Data preprocessing for level 0

As a random forest model relies on multiple decision trees, and each decision tree requires various features, we opted to extract additional information from both the protein sequence and the esm2 embeddings Lin *et al.* (2023).

From the amino acid mass table (see Bio (2021)) TODO: find author of mass table / another source we computed the mass of protein sequences by adding up the individual masses of their amino acid components. The esm2 embeddings, each represented by a 2560-dimensional vector, underwent statistical analysis. We computed the median, standard deviation, and vector magnitude by aggregating values across all 2560 dimensions for each protein. To simplify the embeddings, we applied Principal Component Analysis (Pedregosa *et al.* (2011)) separately to enzyme and non-enzyme datasets, retaining 90% of the variance. This process yielded reduced dimensions of 397 for enzymes and 369 for non-enzymes. Therefore, we reduced the dimensions to 397, providing a

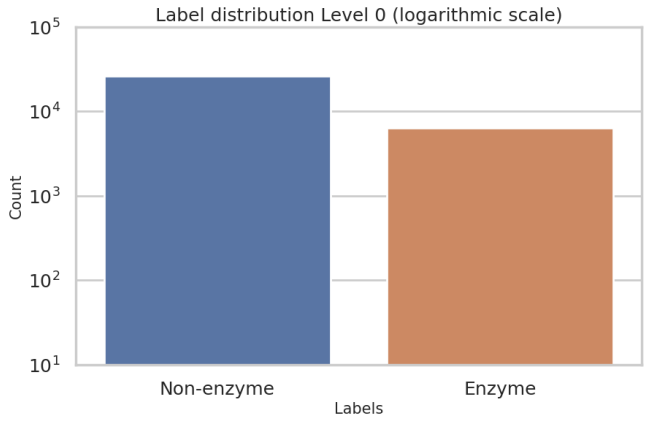


Fig. 1. Barplot showing the amount of sequences per class in the complete data pool

streamlined representation of the protein embeddings while retaining crucial information for both enzyme (see figure 2) and non-enzyme (see figure 3) datasets.

By combining the information from the proteins' sequences, masses, and embeddings, we created a Pandas DataFrame that concatenates enzymes and non-enzymes, including 401 features: mass, embeddings median, embeddings standard deviation (std), embeddings magnitude, and dimension 1 to 397 of the reduced embeddings.

We also used the method SelectFromModel from the scikit-learn library (see Pedregosa *et al.* (2011)) to select features based on importance weights. The refined set of input features contains 'Mass,' 'Emb median,'

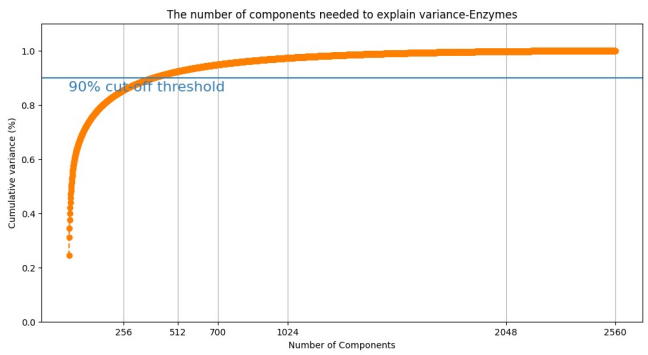


Fig. 2. The number of components needed to explain the variance in the enzyme dataset

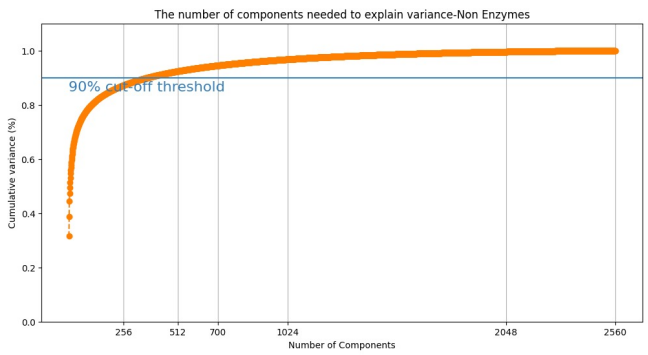


Fig. 3. The number of components needed to explain the variance in the non enzyme dataset

'Emb std,' 'Emb magnitude,' and 'PCA 1' through 'PCA 47' (see table 2) excluding 'PCA 29', 'PCA 31', 'PCA 35', 'PCA 36', 'PCA 39' to 'PCA 42', 'PCA 44' and 'PCA 46' (where 'Emb' represents embeddings and 'PCA X' signifies the X-th dimension of the reduced embeddings).

Table 2. Input features for the Random Forest model

Mass	Emb median	Emb std.	Emb magnitude	PCA 1	…	PCA 47
60153,711	-0,002189	0,227337	11,502516	0,450330	…	-0,050497
81547,542	-0.002620	0,240143	12,150464	-0,087801	…	0,220343
…	…	…	…	…	…	…

3.5 Training procedure

For all our models we divided the data table into two parts: a training set and a validation set, using a random state of 42 for consistency. The training set contains 70% of the original data, while the validation set holds the remaining 30%. This separation allows us to train our models on a subset of the data and then assess its performance on a different subset to ensure its generalization to new, unseen data.

3.5.1 Random Forest

A straightforward yet powerful machine learning method is the Random Forest algorithm. The choice of Random Forest is driven by its effectiveness in handling classification tasks, making it a well-suited option for our specific protein classification objective. We built a Random Forest Classifier using the scikit-learn library with specific parameters, which will be explained in the training procedure. For the Random Forest classifier we addressed the imbalance between the number of non-enzymes and enzymes, where non-enzymes outnumber enzymes approximately fourfold, by duplicating the enzyme data in the training set four times. This duplication ensures a more balanced dataset, allowing the model to be trained on an equal representation of both classes.

The classifier consists of a total of 200 decision trees. Each tree has a maximum depth of 16, and a node is designated as a leaf only if it has a minimum of 8 samples. The random state parameter is set to 42, ensuring reproducibility in the model's construction.

3.5.2 Level 1 FNN

3.5.3 Level 2 FNN

3.6 Validation on test dataset

3.6.1 Scoring metrics

$$\textbf{precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{1}$$

$$\textbf{recall} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{2}$$

$$\textbf{f1-score} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} \tag{3}$$

$$\textbf{weighted f1-score} = \sum_{i=i}^{N} \frac{\text{Number of samples in class } i}{\text{Total number of samples}} \times \text{f1-score}_i$$

(where N is the number of classes) (4)

$$\textbf{MCC score} = \frac{\mathrm{TP} \cdot \mathrm{TN} - \mathrm{FP} \cdot \mathrm{FN}}{\sqrt{(\mathrm{TP} + \mathrm{FP}) \cdot (\mathrm{TP} + \mathrm{FN}) \cdot (\mathrm{TN} + \mathrm{FP}) \cdot (\mathrm{TN} + \mathrm{FN})}} \tag{5}$$

The *MCC* score serves as an indicator for the accuracy of binary classifications, with values ranging from -1 to 1. A score of 1 signifies a flawless prediction, -1 indicates a completely inaccurate prediction, and 0 suggests predictions at random.

To assess the performance of our models, we emphasize the F1 score and MCC score. Given the imbalance in the test set, we also considered the weighted F1 score to ensure a more equitable evaluation of the model's effectiveness.

4 Results

4.1 Random Forest Level 0

Using the Random Forest model on the "new" test set, we achieved accurate predictions for about 90% of positive cases (enzymes) and 98% of negative cases (non-enzymes), even with an imbalanced test set comprising 392 enzymes and 9876 non-enzymes. As a reference we provided the confusion matrix in figure 4.

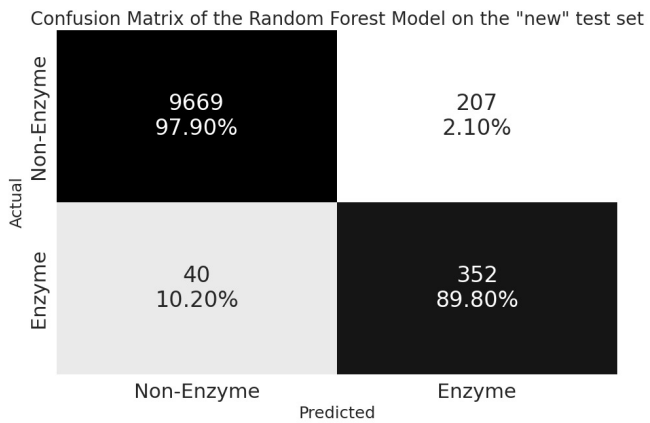


Fig. 4. Confusion Matrix of the Random Forest Model on the "new" dataset

Additionally, we attained an Accuracy of 97.6%, a weighted f1-score of 97.8%, and an MCC-score of 74.1%. Figure 5 illustrates that our Random Forest model significantly outperformed the random baseline.

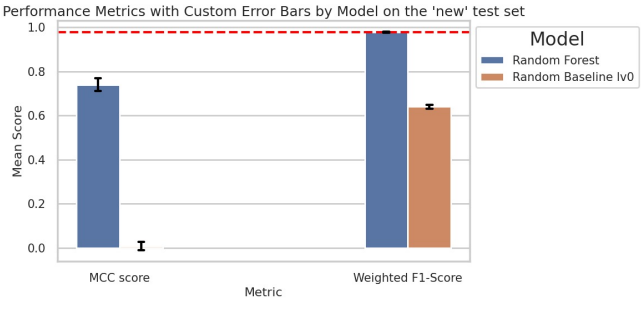


Fig. 5. Level 0 model comparison of Random Forest and baseline on "new" dataset

Furthermore, when we tested our model on a distinct test set, the "price" dataset, it demonstrated a strong performance: only one enzyme was misclassified (see figure 6). This reinforces the robustness and generalizability of our Random Forest model across diverse datasets.

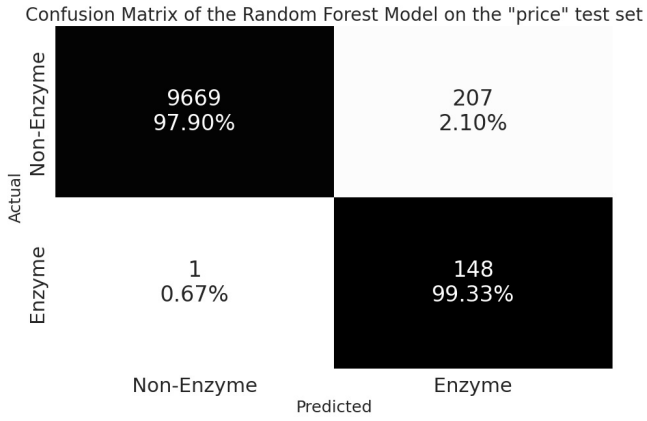


Fig. 6. Confusion Matrix of the Random Forest Model on the "price" dataset

4.1.1 Support Vector Machine

4.2 Level 1 performance

5 Discussion

6 Conclusion

7 Supplementary Information

7.1 Additional binary classification models

As we thoroughly reviewed the Random Forest model in section 4.1, we will now outline the other binary classification models we considered for this task.

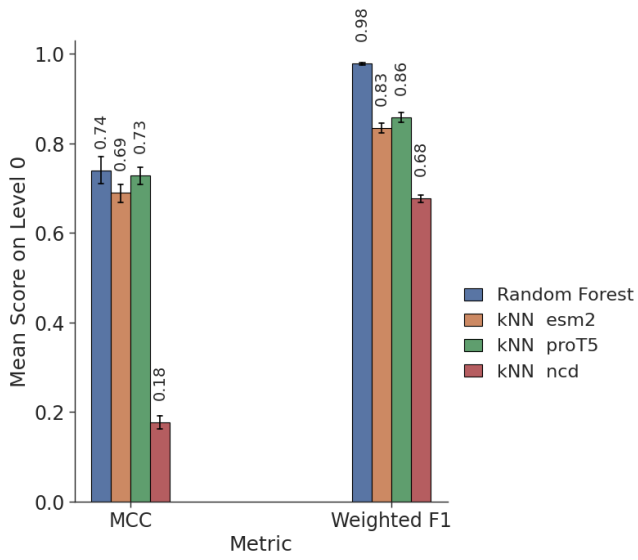
7.1.1 k-Nearest-Neighbors (kNN)

kNN is a non-parametric classification method, which assigns a new object to the most common class amongst the most similar k objects in the data set (Hand (2007)). We implemented three kNN models in Python using scikit-learns Nearest Neighbors library and these models using three distinct types of input features:

(1) Protein embedding vectors encoded by ProtT5 (see Elnaggar *et al.* (2021))
(2) Protein embedding vectors encoded by ESM2 (see Lin *et al.* (2023))
(3) Normalized compression distance vectors (see Jiang *et al.* (2023))

(1) kNN using ProtT5 encoded embedding vectors: Each protein sequence, represented as a numerical vector of 1024 dimensions, is placed as a data point within a 1024-dimensional space. The kNN classifier, using k=7, categorizes each data point to the class that is most common among its k nearest neighbors in the 1024-dimensional space. Figure illustrates the performance comparison between the kNN algorithm using ProtT5 encoded embedding vectors versus a random baseline. We observe that the mean F1 score for this model demonstrates a slight improvement over the random baseline, recording 0. 857 compared to 0.843. This implies that this approach showcases superior precision and recall in contrast to the random baseline. This models Mathews Correlation Coefficient of 0.728 indicates that this approach preforms significantly better than random guessing.

(2) kNN using ESM2 encoded embedding vectors: Each protein sequence, represented as a numerical vector of 2560 dimensions, is placed as a data point within a 2560-dimensional space. The kNN classifier, using k=7, categorizes each data point to the class that is most common among its k nearest neighbors in the 2560-dimensional space. Figure illustrates the performance comparison between the kNN algorithm using ESM2 encoded embedding vectors versus a random baseline. Unlike above,



we observe that the mean F1 score for this model demonstrates a slight regression over the random baseline, recording 0.833 compared to 0.843. This implies that this approach showcases inferior precision and recall in contrast to the random baseline. Despite being lower than the previous case, this models Mathews Correlation Coefficient of 0.689 still indicates that this approach preforms better than random guessing.

(3) kNN using normalized compression distance vectors: The normalized compression distance (ncd) algorithm transforms string like input features into numerical values and is based on the concept of measuring the similarity of two strings by the amount of information needed to describe the one string given the other string. Given two strings x and y, the ncd is defined as follows:

$$ncd(x, y) = \frac{C(xy) - \min(C(x), C(y))}{\max(C(x), C(y))} \tag{6}$$

where $C(x)$ is the length of the compressed string x, $C(xy)$ is the length of the concatenated string xy.

We implemented this algorithm in python using *gzip*, which is a loss less compression algorithm based on a combination of LZ77 and Huffman encoding (Rigler *et al.* (2007)). The ncd algorithm transformed amino acid sequences into numerical vectors by comparing each sequence

with all others in the training dataset. This transformation yielded an n-dimensional numerical vector for each sequence, where n represents the number of sequences in the training dataset. Each position in the input vector signifies the ncd of the sequence concerning the corresponding sequence in the training dataset. These vectors were then used as input for the k-nearest neighbors algorithm. Due to the exponential computational complexity of the ncd algorithm, we used under sampling of the non-enzyme dataset to match the sample size of our enzyme dataset, ensuring balance in the positive instances within the training dataset.

When inferring unseen data, the ncd input vector was calculated by comparing it to all sequences in the training data set, resulting in a n-dimensional numerical vector. Consequently, the performance on new data heavily relies on the characteristics of the training dataset.

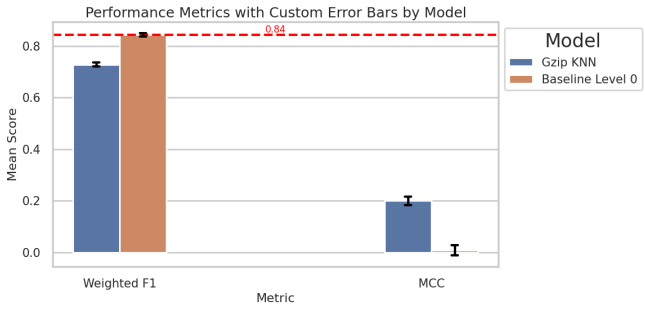


Fig. 7. Performance on test dataset compared to random baseline

Figure 7 illustrates the performance comparison between the kNN algorithm using ncd vectors versus a random baseline. Despite the mean F1 score of the kNN lying at 0.728, it did not perform better than the random baseline, which achieved a F1 score of 0.843. This observation suggests that the ncd approach exhibits inferior precision and recall compared to the random baseline. Additionally, both classifiers exhibit a low MCC score of 0.2 and 0.01, respectively, indicating that neither classifier performs better than random guessing.

The reason for the poor performance of the k-nearest neighbors algorithm using ncd vectors is most likely due to the ncd algorithm not being suited for protein sequences as shown in Matsumoto *et al.* (2000) as well as the test dataset not being balanced, while the training dataset was.

7.1.2 Support Vector Machine (SVM)

SVM is an algorithm that assigns labels to objects by learning from examples (Boser *et al.* (1992)). We implemented three SVM models in Python using scikit-learns C-Support Vector classification library and these models using three distinct types of input features:

(1) Protein embedding vectors encoded by ProtT5 (Elnaggar *et al.* (2021))
(2) Protein embedding vectors encoded by ESM2 (Lin *et al.* (2023))
(3) One hot encoded protein sequence vectors (Wang *et al.* (2021))

(1) SVM using ProtT5 encoded embedding vectors:

(2) SVM using ESM2 encoded embedding vectors:

(3) SVM using one hot encoded protein sequence vectors:

References

(2021).

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152.

Elnaggar, A., Heinzinger, M., Dallago, C., Rehawi, G., Wang, Y., Jones, L., Gibbs, T., Feher, T., Angerer, C., Steinegger, M., *et al.* (2021). Prottrans: Toward understanding the language of life through self-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, **44**(10), 7112–7127.

Hand, D. J. (2007). Principles of data mining. *Drug safety*, **30**, 621–622.

Jiang, Z., Yang, M., Tsirlin, M., Tang, R., Dai, Y., and Lin, J. (2023). "low-resource" text classification: A parameter-free classification method with compressors. In A. Rogers, J. Boyd-Graber, and N. Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 6810–6828, Toronto, Canada. Association for Computational Linguistics.

Lin, Z., Akin, H., Rao, R., Hie, B., Zhu, Z., Lu, W., Smetanin, N., Verkuil, R., Kabeli, O., Shmueli, Y., *et al.* (2023). Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, **379**(6637), 1123–1130.

Matsumoto, T., Sadakane, K., and Imai, H. (2000). Biological sequence compression algorithms. *Genome Informatics*, **11**, 50–51.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, **12**, 2825–2830.

Rigler, S., Bishop, W., and Kennings, A. (2007). Fpga-based lossless data compression using huffman and lz77 algorithms. In *2007 Canadian Conference on Electrical and Computer Engineering*, pages 1235–1238.

Ryu, J. Y., Kim, H. U., and Lee, S. Y. (2019). Deep learning enables high-quality and high-throughput prediction of enzyme commission numbers. *Proceedings of the National Academy of Sciences*, **116**(28), 13996–14001.

Wang, Y., Li, Z., Zhang, Y., Ma, Y., Huang, Q., Chen, X., Dai, Z., and Zou, X. (2021). Performance improvement for a 2d convolutional neural network by using ssc encoding on protein–protein interaction tasks. *BMC bioinformatics*, **22**(1), 1–16.

Wikipedia contributors (2023). Enzyme commission number — Wikipedia, the free encyclopedia.

Yu, T., Cui, H., Li, J. C., Luo, Y., Jiang, G., and Zhao, H. (2023a). Enzyme function prediction using contrastive learning. *Science*, **379**(6639), 1358–1363.

Yu, T., Cui, H., Li, J. C., Luo, Y., Jiang, G., and Zhao, H. (2023b). Enzyme function prediction using contrastive learning. *Science*, **379**(6639), 1358–1363.