Alexander Ratzan, Eshaan Raj Sharma
Big Data Final Project Report
May 11th, 2024

Lightweight Big Data Pipeline & Tools for Alpha Fold Protein Predictions

Summary

Google Deepmind's AlphaFold is a powerful model for predicting a protein's 3D structure from its amino acid sequence alone. This tool is currently revolutionizing the field of computational biology by providing insight into the roles of existing and novel proteins. While functionally very useful, the computational cost of training the model and using it for inference is high. In some cases, the computational cost outweighs the benefit especially when the model makes poor predictions. In this report, we outline an approach to parse and embed protein sequences and learn basic relationships between a protein's sequence and AlphaFold's prediction confidence. Our approach embodies a lightweight data mining, parsing, and embedding pipeline that can be applied to candidate proteins to understand whether AlphaFold will give it a high prediction. A critical outcome of this project are vectorized and embedded forms of protein sequences that can be efficiently stored in appropriate databases, such as vector databases, and analyzed using machine learning and NLP tools for research questions of interest. In sum, we develop a lightweight pipeline and tools that can be paired with the impressive capabilities of the full AlphaFold model.

Background

Biological Significance

Proteins drive the biological processes essential to life for all organisms. Amino acids are the molecular building blocks of proteins. The structure of a protein is a product of the complex folding of the long amino acid sequence. There are 20 unique amino acids that can be sequenced together in long chains to form proteins. Each unique amino acid has its own unique chemical properties. The interactions between amino acids cause the complex formation of proteins; the formation process of proteins is commonly referred to as 'protein folding'.

The three-dimensional structure of a protein determines how it interacts with other molecules, including substrates, inhibitors, and receptors, thus influencing its functional role in the body. Misfolded proteins can lead to severe diseases such as Alzheimer's, Parkinson's, and cystic fibrosis. Understanding how proteins fold and how their structures are related to function can lead to the development of targeted therapies for these and other diseases. Insights into protein folding can accelerate the discovery of new drugs and therapeutic interventions, ultimately improving human health and well-being. Therefore, generating and understanding protein folding predictions carries ample biological significance

Description of AlphaFold

The goal of AlphaFold is to accurately predict a protein's 3D structure from its amino acid sequence using advanced deep learning techniques. It aims to bridge the gap between the vast number of protein sequences available and the relatively small number of experimentally determined protein structures, providing insights into the roles and functions of existing and novel proteins.

AlphaFold's main model is a neural network that leverages a multiple sequence alignment (MSA) as its primary input. This MSA aligns the amino acid sequence of the target protein with other similar sequences to identify evolutionary relationships and covariation between them. It also represents each pairwise amino acid interaction in the protein as a graph. The MSA and pair representations are then used to make a final 3D prediction. The neural network consists of several components, including an Evoformer, which iteratively refines the MSA and pair representations to model the relationships between amino acid residues. Through a continuous flow of information between the MSA and pair representations, AlphaFold can reason about spatial and evolutionary relationships, ultimately refining its predictions. In a broad sense, AlphaFold's model can be thought of as a deep learning model bootstrapped with similarity search and a biophysical representation schema.

Given an amino acid sequence, the outputs of AlphaFold include the predicted 3D structure of the protein, provided in file formats such as .pdb and .cif, which contain the atomic coordinates of the 3D protein structure. Additionally, AlphaFold supplies confidence metrics for its predictions, such as the predicted local distance difference test (pLDDT) and predicted aligned error (PAE). These metrics provide insights into the local confidence and global confidence, respectively, of the model's predictions. For simplicity and overall utility, we focus here on pLDDT at each amino acid position and average pLDDT over the full protein. This measure can also serve as a proxy for protein instability since proteins whose conformational structure changes dynamically are hard to predict and have low prediction confidences. The pLDDT score is a per-residue measure of local confidence scaled from 0 to 100, with higher scores indicating higher confidence with scores above 90 indicating high confidence and accuracy.

Methods & Results

Dataset

The AlphaFold dataset is a comprehensive resource of predicted protein structures, generated by Google DeepMind's AlphaFold model. In our project we specifically work with a publicly available subset of the dataset, containing approximately 20,000 predicted protein structures from the broader set of over 200 million predictions. This subset represents a significant volume of data totaling around 5 GB of files in the .cif format. These files provide detailed 3D structural information for each protein, including the atomic coordinates and confidence metrics such as predicted local distance difference test (pLDDT). The amino acid

sequence and pLDDT information is of primary interest here. As more protein structures are released to the research community, there will be an increasing need for downstream applications that can scale with the growing volume of data. Our dataset provides some proof of concept tools to scale with the data. In our presentation, there is a video on slide 4 showing the output for a relatively small protein.

Data Mining

Prototyping for our pipeline was conducted using a subset of our data (approximately 1 GB/2000 proteins). A single .cif file for a protein can vary a lot in size and can include several thousand lines of data. The goal of our pipeline is to parse through these files efficiently to retain only the relevant data for our downstream purposes. Each protein prediction file includes information regarding all authors who trained AlphaFold and the experimentally validators of the given protein. Other information not of interest in our dataset are alternative representations of the amino acid chain, atomic coordinates and attributes, and side chain information.

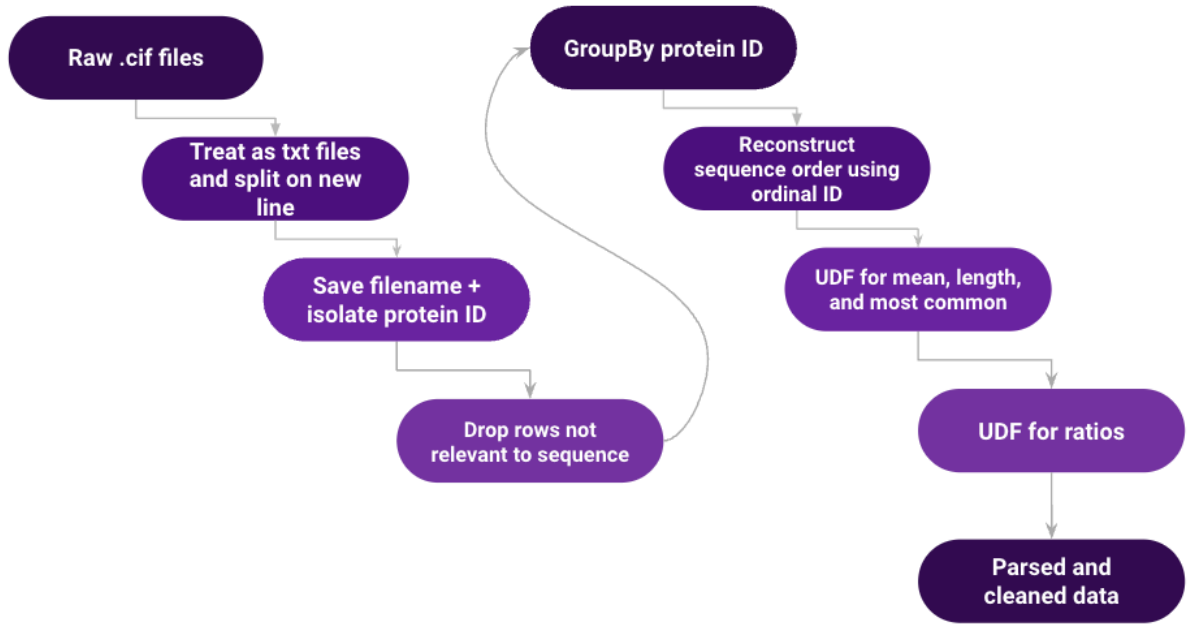
The figure on the right represents the data we were interested in. This data comes from file AF-Q3LI62-F1-model_v4.cif, which means it is the Q3LI62 protein predicted with the AlphaFold F1 model. The data scraping pipeline is outlined in the figure below. As data is read in on a file by file basis, the file name (which contains the protein name) is stored for each line. Then irrelevant is dropped based on line length and other features. As a result each amino acid is stored in a row of the pyspark dataframe with its corresponding protein, order in the protein sequence, and prediction confidence. This results in one row in the final pyspark df per protein.

The processing can be done using any number of partitions. The unique protein ID and ordinal information allows for the protein sequences to be reconstructed using groupbys and sorting.

Relevant .cif Protein Information

```
_ma_qa_metric_global.metric_id    1
_ma_qa_metric_global.metric_value 57.52
_ma_qa_metric_global.model_id     1
_ma_qa_metric_global.ordinal_id   1
#
loop_
_ma_qa_metric_local.label_asym_id
_ma_qa_metric_local.label_comp_id
_ma_qa_metric_local.label_seq_id
_ma_qa_metric_local.metric_id
_ma_qa_metric_local.metric_value
_ma_qa_metric_local.model_id
_ma_qa_metric_local.ordinal_id
A MET 1  2 40.77 1 1
A SER 2  2 42.88 1 2
A TYR 3  2 51.80 1 3
A TYR 4  2 47.52 1 4
A SER 5  2 53.94 1 5
A HIS 6  2 53.11 1 6
A LEU 7  2 51.61 1 7
A SER 8  2 50.27 1 8
A GLY 9  2 46.75 1 9
A GLY 10 2 47.25 1 10
A LEU 11 2 47.21 1 11
A GLY 12 2 47.04 1 12
A CYS 13 2 50.74 1 13
A GLY 14 2 54.01 1 14
A LEU 15 2 61.90 1 15
A ALA 16 2 66.13 1 16
A VAL 17 2 68.36 1 17
A ALA 18 2 75.15 1 18
A VAL 19 2 77.59 1 19
A THR 20 2 83.26 1 20
A MET 21 2 77.34 1 21
A GLY 22 2 82.18 1 22
A ARG 23 2 85.37 1 23
A THR 24 2 72.08 1 24
A VAL 25 2 77.42 1 25
A ALA 26 2 75.36 1 26
A VAL 27 2 71.16 1 27
A ALA 28 2 66.30 1 28
A GLU 29 2 66.00 1 29
A TYR 30 2 57.87 1 30
A GLY 31 2 59.11 1 31
A ARG 32 2 54.71 1 32
A CYS 33 2 55.95 1 33
A ARG 34 2 56.61 1 34
A HIS 35 2 51.10 1 35
A GLY 36 2 48.48 1 36
A CYS 37 2 51.34 1 37
A HIS 38 2 50.43 1 38
A SER 39 2 50.87 1 39
A SER 40 2 47.47 1 40
A TYR 41 2 45.68 1 41
A SER 42 2 41.27 1 42
A ALA 43 2 41.80 1 43
A ARG 44 2 42.89 1 44
```

Preprocessing Pipeline



Feature Engineering

The main goal of this project was to parse and preprocess the original data to extract basic features that could be leveraged for other data science research questions. Once proteins are read in and ordinally reconstructed, we implement several UDFs to compute basic stats such as average confidence, length of the protein, most common amino acid, and ratio of each amino acid in the protein. This can be done in a highly parallelized efficient way that can eventually scale with many thousands of protein .cif files.

This essentially leads to several vectors to represent each individual protein. Contemporary technologies such as vector databases can leverage this data representation for efficient storage and rapid analysis. Furthermore, these basic representations retain sequential information, which is likely highly relevant to protein folding, stability, and prediction confidence.

Parsed and Cleaned PySpark Dataframe

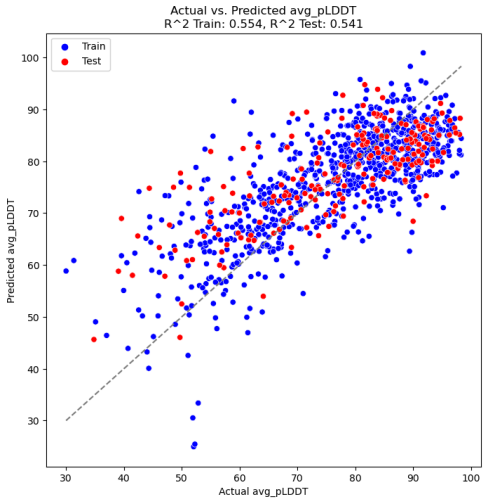
protein_id	protein_seq	protein_seq_len	most_common_aa	aa_pLDDT_list	avg_pLDDT	ALA_ratio	ARG_ratio	ASN_ratio	ASP_ratio	...
P01764	[MET, GLU, PHE, GLY, LEU, SER, TRP, LEU, PHE, ...	117	SER	[41.81999969482422, 45.2599983215332, 54.70999...	91.027008	0.085470	0.042735	0.025641	0.025641	...
P01833	[MET, LEU, LEU, PHE, VAL, LEU, THR, CYS, LEU, ...	764	SER	[52.939998626708984, 50.970001220703125, 50.18...	76.677879	0.075916	0.053665	0.051047	0.049738	...
P02545	[MET, GLU, THR, PRO, SER, GLN, ARG, ARG, ALA, ...	664	LEU	[35.939998626708984, 32.06999969482422, 36.529...	76.273735	0.088855	0.096386	0.031627	0.054217	...

Exploratory Data Analysis

We performed some basic data analysis as a quick first pass to see if statistics like protein length or amino acid ratio are relevant for protein prediction confidence. In short, there are some patterns between how much of an amino acid there is a protein and how confidently AlphaFold will make a prediction of 3D structure. If a protein has relatively large amounts of Isoleucine or Valine it is likely to lead to a more confident prediction than if there is a lot of Serine, which is more likely to give a low confidence prediction. This is based on Pearson's correlation coefficient. Furthermore, pairing ratio data with overall protein length, we trained a linear regression model to show that there is some signal in these basic measures as evidenced by a test set R^2 of 0.54. These modest results motivated a deeper exploration into vector embeddings that encode these basic signals in addition to contextual/sequential information with greater relevance to protein prediction confidence

Amino Acid Ratio Correlation with Confidence and Linear Regression using Length and Ratio

Amino Acid	Correlation Coefficient	p-value
ALA	0.099032	1.341611e-03
ARG	-0.139749	5.716280e-06
ASN	0.189308	6.779652e-10
ASP	0.245841	7.291010e-16
CYS	-0.270056	6.145246e-19
GLN	-0.131247	2.061303e-05
GLU	-0.025427	4.113508e-01
GLY	-0.278491	4.377644e-20
HIS	-0.100018	1.199363e-03
ILE	0.452081	8.031751e-54
LEU	0.321871	1.221766e-26
LYS	0.098032	1.501488e-03
MET	0.282805	1.093568e-20
PHE	0.296939	9.771680e-23
PRO	-0.392591	7.068362e-40
SER	-0.514643	8.427953e-72
THR	0.116329	1.628465e-04
TRP	0.166481	6.103903e-08
TYR	-0.080001	9.640863e-03
VAL	0.466377	1.314936e-57



Associative Rule Mining

Introduction

Since its introduction in 1993, the task of association rule mining has received a great deal of attention. Today the mining of such rules is still one of the most popular pattern-discovery methods in KDD.

In brief, an association rule is an expression X =~ Y, where X and Y are sets of items. The meaning of such rules is quite intuitive: Given a database ~ of transactions - where each transaction T E D is a set of items -, X:=~Y expresses that whenever a transaction T contains X then T probably contains Y also. The probability or rule confidence is defined as the percentage of transactions containing Y in addition to X about the overall number of transactions containing X. That is, the rule confidence can be understood as the conditional probability p(Y C T I X C

T). The idea of mining association rules originates from the analysis of market-basket data where rules like "A customer who buys products xl and x2 will also buy product y with probability c%." are found. Their direct applicability to business problems together with their inherent understandability - even for non-data mining experts - made association rules a popular mining method. Moreover, it became clear that association rules are not restricted to dependency analysis in the context of retail applications, but are successfully applicable to a wide range of business problems.

When mining association rules, there are mainly two problems to deal with: First Of all there is the algorithmic complexity. The number of rules grows exponentially with the number of items. Fortunately, today's algorithms can efficiently prune this immense search space based on mini-real thresholds for quality measures on the rules. Second, interesting rules must be picked from the set of generated rules. This might be quite costly because the generated rule sets normally are quite large- e.g.more 100,000 rules are not uncommon - and in contrast, the percentage of useful rules is typically only a tiny fraction. The work concerning the second problem mainly focuses on supporting the user when browsing the rule set, and the development of further useful quality measures on the rules.

Formal Problem Description

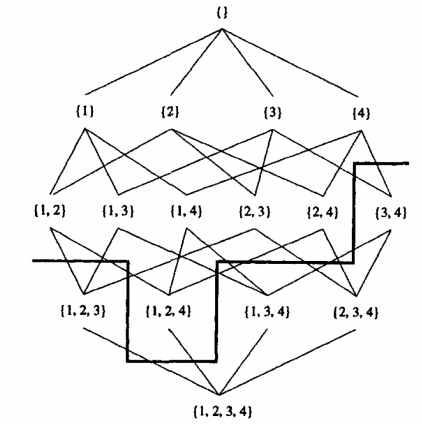
Let Z = {Xl,...,xn} be a set of distinct literals, called items. A set X C Z with k = IXI is called a k-itemset or simply an item. Let a database ~ be a multi-set of subsets of Z. Each T E D is called a transaction. We say that a transaction T E :D supports an itemset X C Z if X C T holds. An association rule is an expression X =~ Y, where X, Y are itemsets and X n Y = O holds. The fraction of transactions T supporting an itemset X concerning database ~Dis called the support of X, supp(X) = [{T6~D[X CT}[/I~[.The support of a rule X =:~Y is defined as supp(X =~ Y) = supp(XUY). The confidence of this rule is defined as conf(X =~ Y) = supp(X UY)/supp(X)

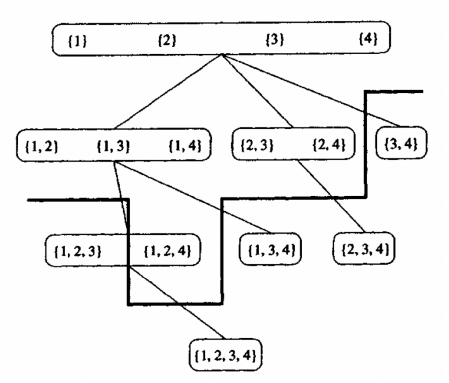
As mentioned before the main challenge when mining association rules is the immense number of rules that theoretically must be considered. The number of rules grows exponentially with [I[. Since it is neither practical nor desirable to mine such a huge set of rules, the rule sets are typically restricted by minimal thresholds for the quality measures that support confidence, and minconf respectively. This restriction allows us to split the problem into two separate parts: An itemset X is frequent if supp(X) > min- supp. Once, T = {X C I I X frequent}, the set of all frequent itemsets together with their support values is known, and deriving the desired association rules is straightforward. For every X E .T check the confidence of all rules X \ Y =~ Y, Y C X, ~ # Y ¢ X and drop those that do not achieve minconf. According to its definition above, it suffices to know all support values of the subsets of X to determine the confidence of each rule. The knowledge about the support values of all subsets of X is ensured by the downward closure property of itemset support: All subsets of a frequent itemset must also be frequent.

With that in mind, the task of association rule mining can be reduced to the problem of finding all itemsets that are frequent concerning a given minimal threshold minimum. The rest of this paper and most of the literature on association rule mining addresses exactly this topic.

Traversing the Search Space

As explained we need to find all itemsets that satisfy min- supp. For practical applications looking at all subsets of I is doomed to failure by the huge search space. A linearly growing number of items still implies an exponentially growing number of itemsets that need to be considered. For the special case I = {1, 2, 3, 4} we visualize the search space that forms a lattice in Figure 1. The frequent itemsets are located in the upper part of the figure whereas the infrequent ones are located in the lower part. Although we do not explicitly specify support values for each of the itemsets, we assume that the bold border separates the frequent from the infrequent itemsets. The existence of such a border is independent of any particular database :D and minsupp. Its existence is solely guaranteed by the downward closure property of itemset support.





The basic principle of the common algorithms is to employ this border to efficiently prune the search space. As soon as the border is found, we can restrict ourselves from determining the support values of the itemsets above the border and ignore the itemsets below.

Let map: I ~ {1,..., lII} be a mapping that maps all items x E I one-to-one onto natural numbers. Now the items can be seen as ordered by the relation "<" between natural numbers. In addition, for X C Z let X.item: {1,..., IXI} ~ I: n ~ X.item be a mapping with X.item,~ denoting the n-th item of the items x E X increasingly sorted by "<". The n-prefix of an itemset X with n _< IX] is then defined by P = {X.item,n I 1 < m < n}. Let the classes E(P), P C I with E(P) = {X C I I IXI = IPI + 1 and P is a prefix of X} be the nodes of a tree. Two nodes are connected by an edge, if all itemsets of a class E can be generated by joining two itemsets of the parent class E', e.g. Figure 2.

Together with the downward closure property of itemset support, this implies the following: If the parent class El of a class E does not contain at least two frequent itemsets then E must also

not contain any frequent itemsets. If we encounter such a class E' on our way down the tree, then we have reached the border separating the infrequent from the frequent itemsets. We do not need to go behind this border so we prune E and all descendants of E from the search space.

The latter procedure allows us to efficiently restrict the number of itemsets to investigate. We simply determine the support values only of those itemsets that we "visit" on our search for the border between frequent and infrequent itemsets. Finally, the actual strategy to search for the border is our own choice. Today's common approaches em- ploy both breadth-first search (BFS) and depth-first search (DFS). With BFS the support values of all (k - 1)-itemsets are determined before counting the support values of the k-itemsets. In contrast, DFS recursively descends following the tree structure defined above.

Implementation of Associative Rule Mining in PySpark

Apache Spark provides powerful implementations of two algorithms for mining frequent patterns: FP-Growth and PrefixSpan. These implementations leverage the distributed computing capabilities of Spark to perform scalable pattern mining, which is critical in handling large datasets efficiently.

FP-Growth

FP-Growth, or Frequent Pattern Growth, is a popular method used to extract frequent item sets from a dataset without the need for candidate generation. This method uses a tree structure known as the FP-tree, which compacts the dataset into a form that is more efficient to work with than the row-by-row transaction format used by earlier algorithms such as Apriori. Spark's implementation of FP-Growth in the spark.ml library allows users to specify parameters such as minimum support, minimum confidence, and the number of partitions for distribution. The algorithm outputs a data frame containing frequent itemsets along with their frequencies and allows the generation of association rules with measures like confidence and lift. It also includes a transform method that applies the rules to new datasets.

```
+----------------+----+
|sequence        |freq|
+----------------+----+
|[[ ASP], [ ALA]]|49  |
|[[ GLY], [ GLY]]|49  |
|[[ ASP], [ TYR]]|49  |
|[[ LEU], [ GLN]]|49  |
|[[ ASP], [ PHE]]|49  |
|[[ LEU], [ ASP]]|49  |
|[[ ASP], [ VAL]]|49  |
|[[ LEU], [ TYR]]|49  |
|[[ ASP], [ SER]]|49  |
|[[ LEU], [ THR]]|49  |
|[[ ASP], [ CYS]]|49  |
|[[ LEU], [ PHE]]|49  |
|[[ LYS], [ LEU]]|49  |
|[[ LEU], [ PRO]]|49  |
|[[ LYS], [ ASP]]|49  |
|[[ LEU], [ SER]]|49  |
|[[ LYS], [ ALA]]|49  |
|[[ LEU], [ ARG]]|49  |
|[[ LYS], [ GLY]]|49  |
|[[ LYS], [ GLU]]|49  |
+----------------+----+
only showing top 20 rows
```

PrefixSpan and Results

PrefixSpan, or Prefix-Projected Sequential Pattern Mining, is another algorithm implemented in Spark for discovering frequent sequences. Unlike FP-Growth, PrefixSpan is designed to handle sequences of itemsets where the order of itemsets is important. The algorithm efficiently finds all frequent subsequences in a dataset and allows the specification of parameters like minimum support, maximum pattern length, and maximum local projected

database size, which helps manage computational resources. The results include sequences that meet the specified support threshold within the given constraints of sequence length.

Both FP-Growth and PrefixSpan in PySpark's machine learning library (spark.ml) provide robust tools for pattern mining which are integral for tasks such as market basket analysis, user behavior modeling, and recommendation systems. These implementations are designed to scale with data volume and complexity, making them suitable for enterprise-level data analysis.

Vector Embedding

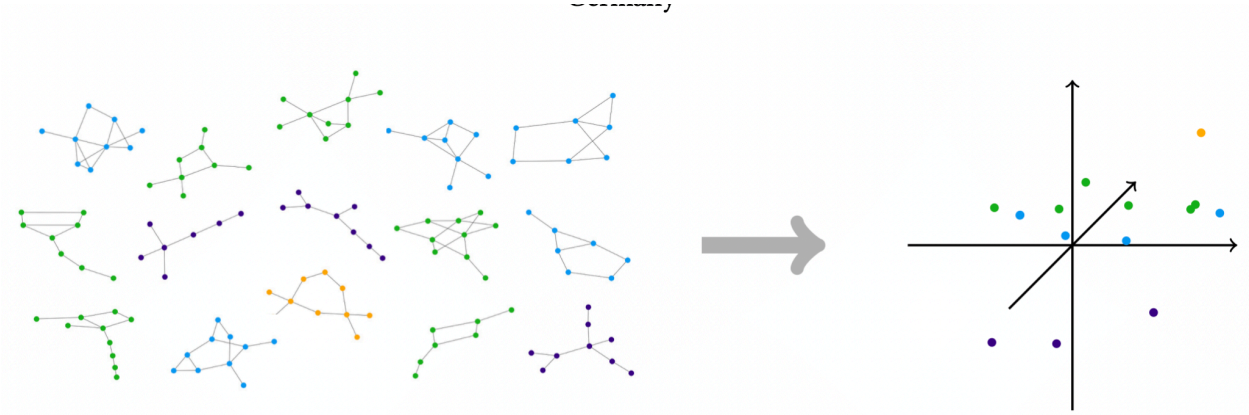


Figure 1: Embedding graphs into a vector space

Typical machine learning algorithms operating on structured data require representations of the often symbolic data as numerical vectors. Vector representations of the data range from handcrafted feature vectors via automatically constructed graph kernels to learned representations, either computed by dedicated embedding algorithms or implicitly computed by learning architectures like graph neural networks. The performance of machine learning methods crucially depends on the quality of the vector representations. Therefore, there is a wealth of research proposing a wide range of vector-embedding methods for various applications. Most of this research is empirical and often geared toward specific application areas. Given the importance of the topic, there is surprisingly little theoretical work on vector embeddings, especially when it comes to representing structural information that goes beyond metric information (that is, distances in a graph).

The goal of this paper is to give an overview of the various embedding techniques for structured data that are used in practice and to introduce theoretical ideas that can, and to some extent have been used to understand and analyze them. The research landscape on vector embeddings is unwieldy, with several communities working largely independently on related questions, motivated by different application areas such as social network analysis, knowledge graphs,

chemoinformatics, computational biology, etc. Therefore, we need to be selective, focussing on common ideas and connections where we see them.

Vector embeddings can bridge the gap between the "discrete" world of relational data and the "differentiable" world of machine learning and for this reason, have a great potential for database research. Yet relatively little work has been done on embeddings of relational data beyond the binary relations of knowledge graphs. Throughout the paper, I will try to point out potential directions for database-related research questions on vector embeddings.

A vector embedding for a class X of objects is a mapping f from X into some vector space, called the latent space, which we usually assume to be a real vector space Rd of finite dimension d. The idea is to define a vector embedding in such a way that geometric relationships in the latent space reflect semantic relationships between the objects in X. Most importantly, we want similar objects in X to be mapped to vectors close to one another about some standard metric on the latent space (say, Euclidean). For example, in an embedding of words of a natural language, we want words with similar meanings, like "shoe" and "boot", to be mapped to vectors that are close to each other. Sometimes, we want further-reaching correspondences between properties of and relations between objects in X and the geometry of their images in latent space. For example, in an embedding f of the entities of a knowledge base, among them Paris, France, Santiago, Chile, we may want t:= f (Paris) − f (France) to be (approximately) equal to f (Santiago) − f (Chile) so that the relation is-capital-of corresponds to the translation by the vector t in latent space.

A difficulty is that the semantic relationships and similarities between the objects in X can rarely be quantified precisely. They usually only have an intuitive meaning that, moreover, may be application-dependent. However, this is not necessarily a problem, because we can learn vector representations in such a way that they yield good results when we use them to solve machine learning tasks (so-called downstream tasks). This way, we never have to make the semantic relationships explicit. As a simple example, we may use a nearest-neighbor-based classification algorithm on the vectors our embedding gives us; if it performs well then the distance between vectors must be relevant for this classification task. This way, we can even use vector embeddings, trained to perform well on certain machine learning tasks, to define semantically meaningful distance measures on our original objects, that is, to define the distance Dist (X, Y) between objects, $Y \in X$ $\| f(X) - f(Y) \|$. We call Distf the distance measure induced by the embedding.

In this paper, the objects $X \in X$ we want to embed either are graphs, possibly labeled or weighted, or more generally relational structures, or they are nodes of a (presumably large) graph or more generally elements or tuples appearing in a relational structure. When we embed entire graphs or structures, we speak of graph embeddings or relational structure embeddings; when we embed only nodes or elements we speak of node embeddings. These two types of embeddings

are related, but there are clear differences. Most importantly, in node embeddings, there are explicit relations such as adjacency and derived relations such as distance between the objects of X (the nodes of a graph), whereas in graph embeddings all relations between objects are implicit or "semantic", for example, "having the same number of vertices" or" having the same girth" (see Figure 1).

The key theoretical questions we will ask about vector embedding of objects in X are the following.

Expressivity: Which properties of objects $X \in X$ are represented by the embedding? What is the meaning of the induced distance measure? Are there geometric properties of the latent space that represent meaningful relations on X?

Complexity: What is the computational cost of computing the vector embedding? What are efficient embedding algorithms? How can we efficiently retrieve semantic information of the embedded data, for example, answer queries?

A third question that relates to both expressivity and complexity is what dimension to choose for the latent space. In general, we expect a trade-off between (high) expressivity and (low) dimension, but it may well be that there is an inherent dimension of the data set. It is an appealing idea to think of "natural" data sets appearing in practice as lying on a low-dimensional manifold in high-dimensional space. Then we can regard the dimension of this manifold as the inherent dimension of the data set.

Reasonably well-understood from a theoretical point of view are node embeddings of graphs that aim to preserve distances between nodes, that is, embeddings $f\ V(G) \to \mathbb{R}^d$ of the vertex set V (G) of some graph G such that $\mathrm{dist}_G(x,y) \approx \| f(x) - f(y) \|$, where the list is the shortest-path distance in G. There is a substantial theory of such metric embeddings (see [64]). In many applications of node embeddings, metric embeddings are indeed what we need.

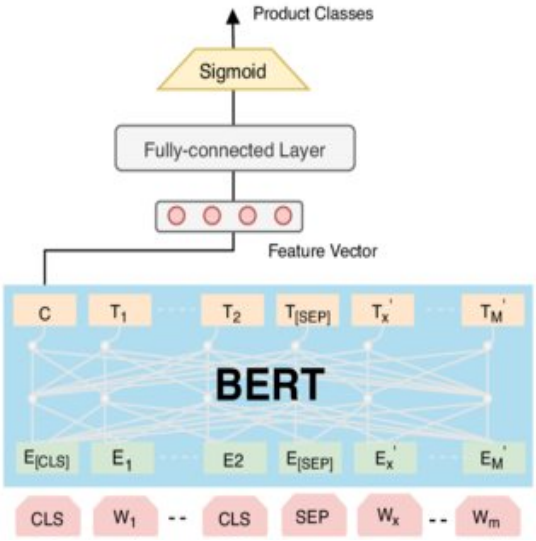
However, the metric is only one aspect of the information carried by a graph or relational structure, and arguably not the most important one from a database perspective. Moreover, if we consider graph embeddings rather than node embeddings, there is no metric to start with. In this paper, we are concerned with structural vector embeddings of graphs, relational structures, and their nodes. Two theoretical ideas that have been shown to help in understanding- ing and even designing vector embeddings of structures are the Weisfeiler-Leman algorithm and various concepts in its context, and homomorphism vectors, which can be seen as a general framework for defining "structural" (as opposed to "metric") embeddings. We will see that these theoretical concepts have a rich theory that connects them to the embedding techniques used in practice in various ways.

The rest of the paper is organized as follows. Section 2 is a very brief survey of some of the embedding techniques that can be found in the machine learning and knowledge representation literature. In Section 3, we introduce the Weisfeiler-Leman algorithm. This algorithm, originally a graph isomorphism test, turns out to be an important link between the embedding techniques described in Section 2 and the theory of homomorphism vectors, which will be discussed in detail in Section 4. Finally, Section 5 is devoted to a discussion of similarity measures for graphs and structures.

ProtBERT

Introduction

ProtBERT is a transformer-based model tailored for protein sequence understanding. It leverages the Bidirectional Encoder Representations from Transformers (BERT) architecture, which has revolutionized natural language processing tasks, to decipher protein sequences. By treating protein sequences like natural language, ProtBERT can comprehend and predict protein properties, classify proteins, and assist in biological research.



Architecture and Training

1. Architecture: ProtBERT is based on Google's BERT architecture, specifically the "bert-large" variant. It contains 24 layers, 16 attention heads, and 340 million parameters.Its input vocabulary is designed for amino acids, and it processes up to 512 amino acids per sequence.

2. Training Process:
   Dataset: ProtBERT was trained using the BFD (Big Fat Database), a comprehensive dataset containing over 2 billion protein sequences.

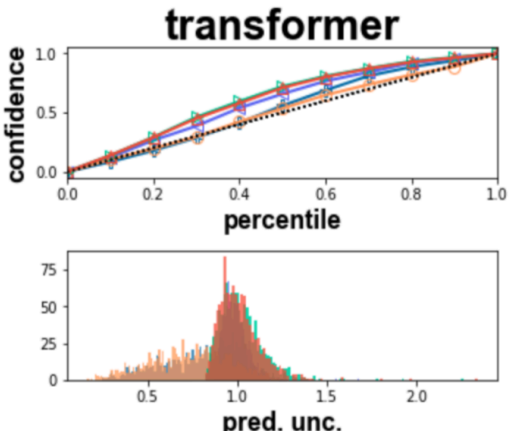
   Training Objective:
   - *Masked Language Modeling (MLM):* Masking a portion of the input tokens to predict them, helping ProtBERT learn protein representations.
   - *Next Sentence Prediction (NSP):* Not applicable due to the structure of protein sequences.

Confidence vs. Percentile Plot:

This plot evaluates the model's calibration by showing predicted confidence across data percentiles. A line close to the diagonal indicates accurate confidence matching true outcomes.

Histogram of Predicted Uncertainty:

Displays the distribution of the model's predicted uncertainties. Most predictions cluster at lower uncertainty levels, suggesting the embedding model is generally confident.



Applications

ProtBERT has demonstrated substantial improvements over traditional sequence-based models in various protein-related tasks:

1. Protein Classification:
    - Effectively classifies protein families, domains, and functions.
    - Identifies homology relationships between sequences.
2. Protein Engineering:
    - Assists in designing proteins with desired properties.
    - Offers predictions for amino acid mutations.
3. Structural and Functional Predictions:
    - Predicts protein secondary structure and solvent accessibility.
    - Infers post-translational modifications and subcellular localization.

Comparison with Other Models

1. ProtTrans:
    - ProtBERT is part of the ProtTrans project, which includes other transformer-based models like T5 and XLNet.
    - ProtBERT's BERT architecture offers robust generalization compared to LSTM-based models like DeepSeq.
2. UniRep:
    - UniRep, based on LSTMs, captures evolutionary information but lags behind ProtBERT in protein sequence understanding due to its relatively limited representation capacity.

Our Methodology

We leverage the Bidirectional Encoder Representations from Transformers (BERT) architecture, which has revolutionized natural language processing tasks, to decipher protein sequences. By treating protein sequences like natural language, ProtBERT can comprehend and predict protein properties, classify proteins, and assist in biological research.

Given the complex similarity of proteins in human biology, we explored a sophisticated state-of-the-art transformer-based model, ProtBERT. ProtBERT works in four levels, which are outlined below. Most of our work was done in the fine-tuning step for embedding protein sequencing. Within this modeling, we have intricate steps, and here is a brief walkthrough of our work.

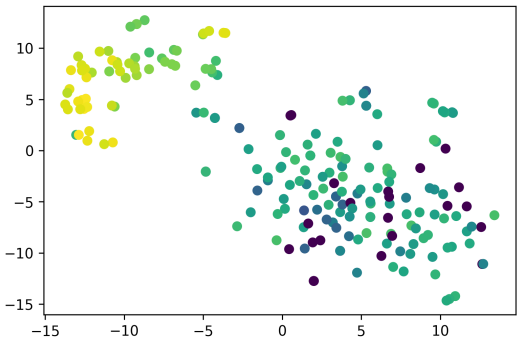
Our Implementation

1. *Input Representation:* ProtBERT treats protein sequences as sentences, where each amino acid corresponds to a "word". Amino acids are converted into tokens, which are then transformed into embeddings that serve as input for the transformer model.
2. *Contextual Embeddings:* The core of ProtBERT comprises transformer blocks that process these embeddings. Each block applies self-attention mechanisms, enabling the model to weigh and consider all amino acids in the sequence. This results in contextual embeddings where the representation of each amino acid dynamically adjusts based on neighboring amino acids.
3. *Pre-training:* ProtBERT is pre-trained on a large corpus of protein sequences using tasks like masked amino acid prediction (analogous to masked word prediction in language). This pre-training helps the model learn a deep, nuanced understanding of protein language.
4. *Fine-tuning:* After pre-training, ProtBERT can be fine-tuned on specific tasks such as protein function prediction, secondary structure prediction, or similarity analysis. Labeled datasets specific to each task are used for fine-tuning.

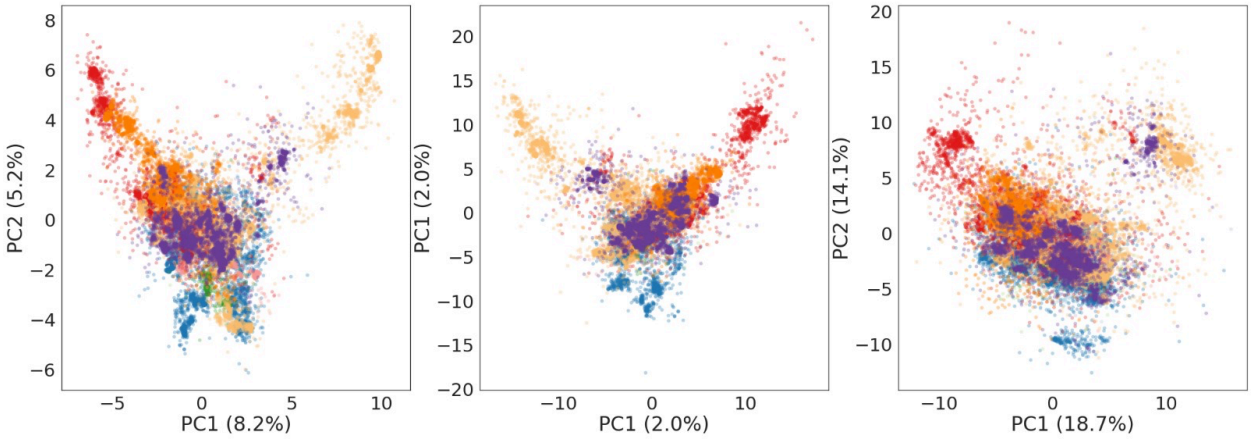
Evaluating Protein Embedding Quality

To assess the performance of the vector embedding model, we relied on two specific metrics: clustering quality and dimensionality reduction visualization.

1. Clustering Quality: Evaluates embeddings based on how well they cluster known families of proteins. Higher homogeneity within clusters and better separation between clusters suggest better embeddings.



2. Dimensionality Reduction Visualization: Techniques like t-SNE or PCA visualize the embeddings in two or three dimensions. Good embeddings should visibly cluster proteins according to their functional or structural properties.

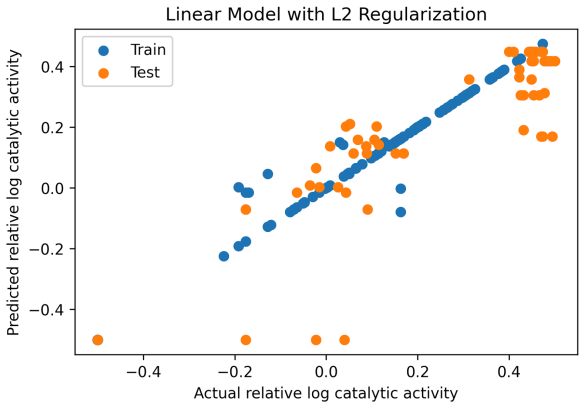


Parallel Training with Dask Incremental PCA: Allows simultaneous training of multiple models to identify the best-performing ProtBERT variation.

Predictive Modeling: Catalytic Activity Prediction

We leveraged vector embeddings derived from protein sequences as the feature set (X variable) and local confidence scores as the target variable (Y variable). Two predictive models were applied:

1. K-Nearest Neighbors (KNN):
    - Non-parametric algorithm that predicts protein properties based on the nearest neighbors in the embedding space.
2. Ridge Regression with Cross-Validation (RidgeCV):
    - Linear model regularized with L2 norm, suitable for high-dimensional data.



Further Improvements

By adding the following three steps in our architecture, we significantly enhanced our model's robustness to data variations:

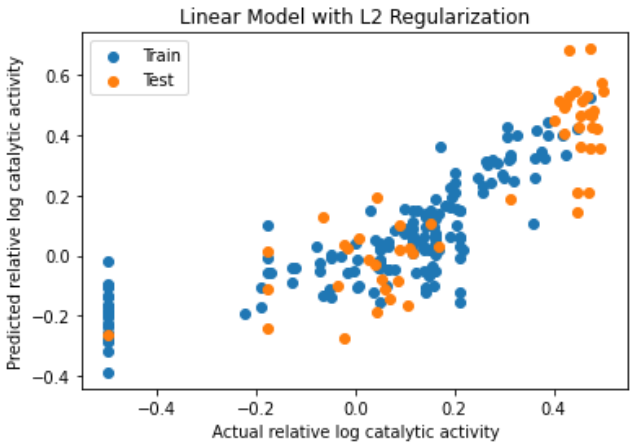
- Our model uses advanced vector embeddings and big data principles to efficiently screen the surge of protein sequence data from AlphaFold 3.
- Vector DB can support everything from data storage and retrieval to complex computational tasks and machine learning integration.

These improvements resulted in more accurate and reliable predictions for protein catalytic activity, demonstrating ProtBERT's potential to revolutionize protein sequence analysis.

Performance Results

The goal was to predict the catalytic activity of proteins based on these inputs. Here are the results:

- Training R^2: 0.6340
- Testing R^2: 0.6867



Conclusion

ProtBERT represents a significant leap in protein sequence modeling by adapting NLP's BERT model for biological applications. Its performance on protein classification, engineering, and structural/functional predictions showcases its potential to reshape bioinformatics research.

Code

- BigDataFinal_EDA.ipynb
- *Basic stats:* BigDataFinal_downstream_analyze.ipynb
- Protein_Vector_Embeddings.ipynb
- Cluster_unknowns.ipynb
- PT5_LoRA_Finetuning_per_residue_reg
- Unknown_cluster_sequence_similarity
- num_clusters

Sources

1. https://www.ebi.ac.uk/training/online/courses/alphafold/
2. Hipp, J., Güntzer, U. and Nakhaeizadeh, G., 2000. Algorithms for association rule mining—a general survey and comparison. *ACM sigkdd explorations newsletter*, *2*(1), pp.58-64.
3. S. Kukreja, T. Kumar, V. Bharate, A. Purohit, A. Dasgupta and D. Guha, "Vector Databases and Vector Embeddings-Review," 2023 International Workshop on Artificial Intelligence and Image Processing (IWAIIP), Yogyakarta, Indonesia, 2023, pp. 231-236, doi: 10.1109/IWAIIP58158.2023.10462847. keywords: {Surveys;Performance evaluation;Costs;Databases;Image processing;Data integrity;Data science;Artificial Intelligence;Computer Vision;Embeddings;Generative AI;Large Language Model},
4. Mirdita, M., Schütze, K., Moriwaki, Y. et al. ColabFold: making protein folding accessible to all. Nat Methods 19, 679–682 (2022). https://doi.org/10.1038/s41592-022-01488-1
5. https://dl.acm.org/doi/pdf/10.1145/3375395.3387641