User Documentation

Overview

This program was created during the winter semester 2024/25 at the **University of Vienna** for the course **"2024W 053531-1 Softwareentwicklungsprojekt Bioinformatik"**, by **Theodora Beshara** and **Sebastian Rossböck**.

The program reads protein IDs from a **FASTA-formatted** file and retrieves annotations for each protein from **UniProt** and **Pfam**. Using **BioBERT**, it generates embeddings for these annotations and then compares and visualizes the results.

Compatibility

This project was set up using **Python 3.11.9** and **Visual Studio**. If you choose to run this program in a different environment, please disregard the next section.

The libraries listed in **requirements.txt** are required to run this program. Please install them using your preferred method.

Setting up the environment

When **Python** and **Visual Studio Code** are installed, we will need to set up a Python environment and install the libraries we will use.

In **Visual Studio Code**, open the folder that contains the repository.

In the terminal (**Str+Ö** or **View** → **Terminal**) type:

```
python -m venv .venv
```

This sets up the Python environment.

After that, install the necessary libraries by typing:

```
pip install -r requirements.txt
```

in the terminal.

Running the program

In general, steps one through six will create a list of **AnnotationData** objects that collect the relevant information for each protein. The list of objects will be serialized at various points in the

program. With the configuration settings, it is possible to set up the program to skip steps that have already been completed. Fetching the annotations can be a time-consuming task.

1. Config

The **biobert.ini** file contains multiple parameters to configure the program. These settings are loaded at the start of execution.

- **fasta_file** : The file location for the FASTA file that contains the IDs of the proteins to be processed.
- **annotation_file_input** : The file location for the annotation data when loading the data from a file.
- **annotation_file_output** : The file location for the annotation data without embeddings.
- **annotation_embedding_file_output** : The file location for the annotation data, including embeddings.
- **data_eval_output** : The file location for the result of the data evaluation.
- **loadAnnotationsFromFile** (boolean) : Specifies whether the annotation data should be loaded from a file or fetched from the APIs.
- **getPfamEmbeddings** (boolean) : Specifies whether embeddings for the Pfam annotations should be generated.
- **getUniProtEmbeddings** (boolean) : Specifies whether embeddings for the UniProt annotations should be generated.
- **model** : The name of the model that will be loaded.
- **distances_plot_output** : The file location where the cosine distance plot will be saved.
- **tsne_plot_output** : The file location where the tsne plost will be saved.

2.1 Loading annotations

If the corresponding configuration is set, the **AnnotationData** will be loaded from a file. When the program runs, a file is created after fetching the data from the APIs. The process of fetching annotations is quite time-consuming since they are retrieved sequentially. It is recommended to load the data from a file once it has been fetched.

2.2 Fetching annotations

For each **RefSeq identifier** in the FASTA file, it is converted to a **UniProtKB ID**. The annotations are then retrieved from **UniProt** and **Pfam** via their respective APIs, and a list of **AnnotationData** objects is created.

These objects store the relevant information for the nucleotide sequences, including:

- **UniProt ID**
- **RefSeq ID**
- **Pfam ID**
- Annotations from **UniProt** and **Pfam**
- **BioBERT embeddings** (added later in the process)

3. Evalutate Data

The annotations are evaluated by analyzing their **average length**, **maximum**, **minimum**, and **missing annotations**. The results are then saved in the location specified by the **data_eval_output** setting.

3.5 Clean Data

For our purpose, only proteins that have both **Pfam** and **UniProt** annotations can be used. Additionally, annotations must not exceed **512 words**, as the **BioBERT** model would truncate any longer annotations, resulting in incomplete data.

For these reasons, all **AnnotationData** objects that do not meet these criteria are removed in this step.

4. Generate Encodings

For each annotation, an embedding is generated by passing the data through the model and then saving the embedding in the respective **AnnotationData** object.

```
inputs = tokenizer(textInput, return_tensors="pt", padding=True, truncation=True, max_length=512)

# Forward pass through the model
with torch.no_grad():
    **outputs = model(inputs)
```

```
# Extract embeddings (CLS token representations)
embeddings = outputs.last_hidden_state[:, 0, :]
```

This data is then saved to the path specified in **annotation_embedding_file_output**. Since this file is too large to upload to GitHub, it has to be created locally. It can be reused after being created once, allowing the first four steps of the program to be skipped.

5. Calculate the distance between embeddings

For each protein, the distance between the embeddings of the **UniProt** and **Pfam** annotations is computed using **cosine similarity**. Since embeddings can be represented as high-dimensional vectors, cosine similarity measures how similar the "direction" of the two vectors is. A plot of a histogram and the min, max and average distance is created.
The resulting plot is saved to the location specified in the **biobert.ini** file.

The value for distance ranges from 0 to 2. A distance of 0 means that the vectors are identical, 1 means orthogonal and 2 means the vectors are opposite.

6. Visualization of Embeddings

To better understand the relationships between protein annotations, **t-Distributed Stochastic Neighbor Embedding (t-SNE)** is applied to visualize the high-dimensional embeddings in a two-dimensional space. The program allows visualization of either **Pfam** or **UniProt** embeddings. By default, **Pfam embeddings** are selected, but this can be changed in the code. t-SNE reduces the dimensionality of data while preserving local relationships between points. The transformation is applied with the following parameters:

- **n_components = 2** (reducing the embeddings to two dimensions)
- **perplexity = 30** (balances local and global aspects of the data)
- **learning_rate = 200**
- **max_iter = 5000**
- **random_state = 42** (ensures reproducibility)

After the transformation, a **DataFrame** is created, storing the computed t-SNE coordinates along with protein names and annotation labels.
The visualization is performed using two methods: **Matplotlib** and **Plotly**. A static **scatter plot** is generated with **Matplotlib**, where each point represents a protein annotation in the transformed space. Labels and colors can be customized to enhance interpretability.
Additionally, an **interactive visualization** is created using **Plotly**, allowing users to explore the data by hovering over points to reveal protein names and annotation details.
For further analysis, **K-Means clustering** can be applied to group proteins with similar annotation embeddings. The number of clusters is configurable (default: 5). The clustered results are visualized in an **interactive Plotly scatter plot**, where each cluster is represented by a distinct color.

These visualizations provide insight into the similarities and differences between protein annotations and can assist in understanding how UniProt and Pfam embeddings relate to each other.

Developer Documentation

This documentation was created with pydoc. The generated html files are included in the zip file and they accessible in the github repository (https://github.com/WyruSeppo/BioBert_Annotations) as well.

program.py

Modules

Bio.SeqIO
configparser
logging
numpy
os
pandas
requests
time
torch
matplotlib.pyplot
plotly.express
sklearn.manifold.TSNE
sklearn.cluster.KMeans

Functions:

main()
Executes the BioBERT annotation similarity workflow.

This includes:

- Reading and validating the configuration.
- Loading or fetching annotation data.
- Cleaning and evaluating the data.
- Generating embeddings.
- Calculating cosine similarity between embeddings.
- Saving results to output files.
- Performing t-SNE dimensionality reduction.
- Generating static (Matplotlib) and interactive (Plotly) visualizations.
- Applying K-Means clustering to analyze annotation similarities.

---

classes.py

#the class AnnotationData bundles all the data we are handling for each protein

Classes

class AnnotationData(builtins.object)

AnnotationData(id: int, pfam_id: str, uniprot_id: str, pfam_embedding, uniprot_embedding, refSeqAccession, entry, entryName, proteinNames, geneNames, organism, pfam_description, uniprot_function, embedding_distance)

Represents annotation data for a protein, including Pfam and UniProt annotations.

Attributes:

id (int): Unique identifier for the annotation.
pfam_id (str): Pfam identifier for the protein.
uniprot_id (str): UniProt identifier for the protein.
pfam_embedding: Embedding representation of the Pfam annotation.
uniprot_embedding: Embedding representation of the UniProt annotation.
refSeqAccession (str): RefSeq accession number.
entry (str): Entry name from UniProt.
entry_name (str): Alternative entry name.
protein_names (str): Names of the protein.
gene_names (str): Names of the associated genes.
organism (str): Organism from which the protein originates.
pfam_description (str): Description of the Pfam annotation.
uniprot_function (str): Functional description from UniProt.
embedding_distance: Distance metric between Pfam and UniProt embeddings.

Methods defined here:

__init__(self, id: int, pfam_id: str, uniprot_id: str, pfam_embedding, uniprot_embedding, refSeqAccession, entry, entryName, proteinNames, geneNames, organism, pfam_description, uniprot_function, embedding_distance)

Initialize self. See help(type(self)) for accurate signature.

__repr__(self)

Returns a string representation of the AnnotationData object.

cleanAnnotations(self)

Cleans annotation descriptions by removing excess whitespace.

to_dict(self)

Converts the AnnotationData object into a dictionary.

Data descriptors defined here:

__dict__
dictionary for instance variables

__weakref__
list of weak references to the object

class EvaluatedData(builtins.object)
Stores statistical evaluations of annotation data, such as counts and lengths of annotations.

Attributes:
no_sequences (int): Number of sequences analyzed.
uniprot_annotation_amount (int): Count of UniProt annotations.
uniprot_annotation_length_min (int): Minimum length of UniProt annotations.
uniprot_annotation_length_max (int): Maximum length of UniProt annotations.
uniprot_annotation_length_avg (int): Average length of UniProt annotations.
uniprot_annotation_no_words (int): Number of words in UniProt annotations.
uniprot_annotation_missing_amount (int): Number of missing UniProt annotations.
uniprot_annotation_missing_percent (float): Percentage of missing UniProt annotations.
pfam_annotation_amount (int): Count of Pfam annotations.
pfam_annotation_length_min (int): Minimum length of Pfam annotations.
pfam_annotation_length_max (int): Maximum length of Pfam annotations.
pfam_annotation_length_avg (int): Average length of Pfam annotations.
pfam_annotation_no_words (int): Number of words in Pfam annotations.
pfam_annotation_missing_amount (int): Number of missing Pfam annotations.
pfam_annotation_missing_percent (float): Percentage of missing Pfam annotations.

Methods defined here:

__init__(self)
Initialize self. See help(type(self)) for accurate signature.

__repr__(self)
Return repr(self).

generateString(self)
Returns a formatted string summary of the evaluation data.

print_data(self)
Prints a formatted summary of the evaluation data.

Data descriptors defined here:

__dict__
dictionary for instance variables

__weakref__
list of weak references to the object

---

apiMethods.py

Modules

Bio.SeqIO
configparser
logging
os
pandas
requests
time

Functions:

annotate_data(annotationData, showProgress=False)
Annotates data with UniProt and Pfam information.

Args:
annotationData (list): List of annotation objects.
showProgress (bool, optional): Whether to show progress. Defaults to False.

Returns:
list: Annotated data.

getUniProtConversion(ffrom, to, refseqIds)
Converts identifiers using UniProt ID mapping service.

Args:
ffrom (str): Source database.
to (str): Target database.
refseqIds (list): List of RefSeq IDs.

Returns:

list: Mapped annotation data.

get_pfam_annotation(protein_id)
Fetches Pfam annotation for a given protein ID.

Args:
protein_id (str): The Pfam ID of the protein.

Returns:
str: The Pfam description, or None if retrieval fails.

get_uniprot_annotation(protein_id)
Fetches UniProt annotation for a given protein ID.

Args:
protein_id (str): The UniProt ID of the protein.

Returns:
tuple: (protein_name, function, pfam_id), or (None, None, None) if retrieval fails.

---

bertMethods.py

Modules

logging
torch

Functions:

getEmbeddings(annotationData, modelName='dmis-lab/biobert-base-cased-v1.1', mode='pfam')
Generates BioBERT embeddings for annotation data.

Args:
annotationData (list): List of annotationData objects.
modelName (str, optional): The BioBERT model to use. Defaults to "dmis-lab/biobert-base-cased-v1.1".
mode (str, optional): Annotation mode, either "pfam" or "uniprot". Defaults to "pfam".

Returns:
list: annotationData with added embeddings.

---

dataMethods.py

Modules

Bio.SeqIO
configparser
logging
os
pandas
requests
time

Functions:

can_save_file(file_path)

Checks whether a file can be saved in the specified directory.

Parameters:
file_path (str): Path to the file.

Returns:
bool: True if the file can be saved, False otherwise.

configIsValid(config)

Validates the configuration by checking if files can be written to specified paths.

Parameters:
config (dict): Dictionary containing file paths from the configuration.

Returns:
bool: True if all paths are writable, False otherwise.

evaluateData(annotationData)

Evaluates annotation data to compute statistics on Pfam and UniProt annotations.

Parameters:
annotationData (list[AnnotationData]): A list of annotation data objects.

Returns:
EvaluatedData: An object containing statistical analysis of the annotation data.

getAnnotations(annotationData, outputFilePath)

Annotates data using an external method and saves the results to a TSV file.

Parameters:
annotationData (list[AnnotationData]): A list of annotation data objects.
outputFilePath (str): Path to save the annotated data.

Returns:
list[AnnotationData]: The updated annotation data list.

loadAnnotations(filePath)

Loads annotation data from a TSV file and converts it into a list of AnnotationData objects.

Parameters:
filePath (str): Path to the TSV file containing annotation data.

Returns:
list[AnnotationData]: A list of AnnotationData objects parsed from the file.

load_ids_fasta(fasta_file)

Loads sequence IDs from a FASTA file.

Parameters:
fasta_file (str): Path to the FASTA file.

Returns:
list[str]: A list of sequence IDs from the FASTA file.

read_config(filePath='biobert.ini')

Reads a configuration file and extracts relevant settings.

Parameters:
filePath (str, optional): Path to the configuration file. Defaults to 'biobert.ini'.

Returns:
dict: A dictionary containing configuration values.

read_tsv(file_path)

Reads a TSV file and returns its contents as a Pandas DataFrame.

Parameters:
file_path (str): Path to the TSV file.

Returns:
pd.DataFrame or None: The DataFrame containing file data, or None if an error occurs.

saveAnnotations(annotationData, outputFilePath)

Saves annotation data to a TSV file.

Parameters:
annotationData (list[AnnotationData]): A list of annotation data objects.
outputFilePath (str): Path to save the data.

writeToFile(input, outputfile)

Writes a string to a file.

Parameters:
input (str): The content to be written.
outputfile (str): Path to the output file.