Configuration Manual

MSc Research Project
MSc Data Analytics

Anjali Augestin
Student ID: X23155086

School of Computing
National College of Ireland

Supervisor: Hamilton Niculescu

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Anjali Augestin
Student ID:	X23155086
Programme:	MSc Data Analytics
Year:	2024
Module:	MSc Research Project
Supervisor:	Hamilton Niculescu
Submission Due Date:	12/12/2024
Project Title:	Configuration Manual
Word Count:	922
Page Count:	7

Signature:	Anjali Augestin
Date:	29th January 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

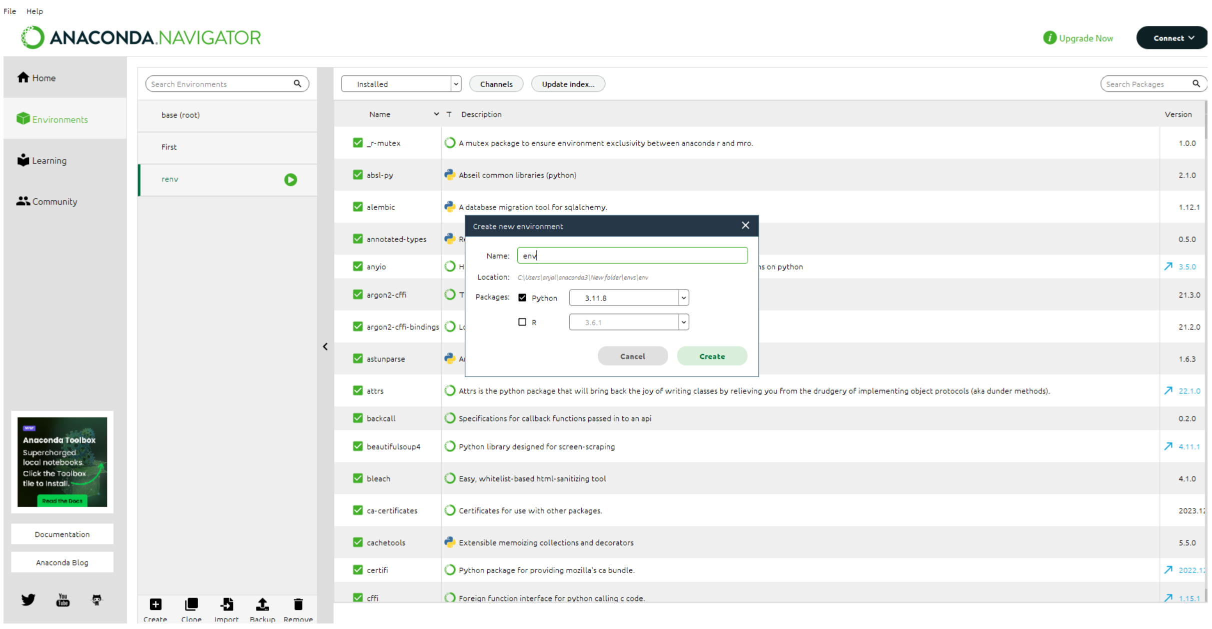
Configuration Manual

Anjali Augestin
X23155086

1 Experimental Setup and Environment

The research is implemented in python programming language and executed utilizing the Jupyter Notebook environment, running on an Anaconda framework.

- Download and install anaconda from `https://www.anaconda.com/download`.
- Open anaconda and click on 'Environments' and next click on 'create' at the bottom left corner.
- Tick the python packages option and give a name, then click create, as shown in fig 1.



Environments: base (root), First, renv

Create new environment

Name: env

Location: C:\Users\anjal\anaconda3\New folder\envs\env

Packages: [✓ Python] 3.11.8; R 3.6.1

Cancel | Create

Name	Description	Version
_r-mutex	A mutex package to ensure environment exclusivity between anaconda r and mro.	1.0.0
absl-py	Abseil common libraries (python)	2.1.0
alembic	A database migration tool for sqlalchemy.	1.12.1
annotated-types	R	0.5.0
anyio	H ns on python	3.5.0
argon2-cffi	T	21.3.0
argon2-cffi-bindings	L	21.2.0
astunparse	A	1.6.3
attrs	Attrs is the python package that will bring back the joy of writing classes by relieving you from the drudgery of implementing object protocols (aka dunder methods).	22.1.0
backcall	Specifications for callback functions passed in to an api	0.2.0
beautifulsoup4	Python library designed for screen-scraping	4.11.1
bleach	Easy, whitelist-based html-sanitizing tool	4.1.0
ca-certificates	Certificates for use with other packages.	2023.1
cachetools	Extensible memoizing collections and decorators	5.5.0
certifi	Python package for providing mozilla's ca bundle.	2022.1
cffi	Foreign function interface for python calling c code.	1.15.1

Figure 1: Anaconda environment creation

- After creating go to home page and launch jupyter notebook and open.
- Navigate to desired location in the system and click 'New' at top right, then select 'python kernel' and start writing the code.

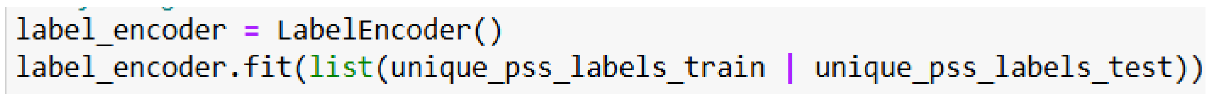
2 Data Source

The datasets used for the research are CullPDB6133_filtered and CB513, can be downloaded from : `https://zenodo.org/records/7764556#ZByi1ezMJvI`. The datasets downloaded are numpy format, then the amino acid sequence and secondary structure labels are extracted from the dataset using the algorithm given in the github repository of its original source, Zhou and Troyanskaya (2014) and saved in text format.

3 Data pre-preprocessing

The data pre-processing steps are implemented using the 'pandas' and 'numpy' python libraries. The libraries are imported to support the steps mentioned in the report for processing the data. For label encoding the 'LabelEncoder' package in python 'scikit learn' library. The label encoded target variables are converted to on-hot labels by using 'to_categorical' package, imported from 'tensorflow'.
In the data preprocessing phase:

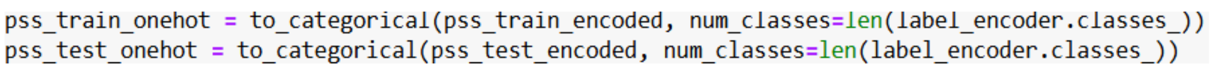
- Analyze the data for null values.
- Check the data for white space and remove the white spaces.
- Label encode the secondary structure labels as shown in fig 2.



```
label_encoder = LabelEncoder()
label_encoder.fit(list(unique_pss_labels_train | unique_pss_labels_test))
```

Figure 2: Label encoding

- Convert the label encoded variables to one-hot labels as shown fig 3.



```
pss_train_onehot = to_categorical(pss_train_encoded, num_classes=len(label_encoder.classes_))
pss_test_onehot = to_categorical(pss_test_encoded, num_classes=len(label_encoder.classes_))
```

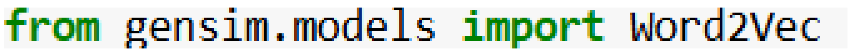
Figure 3: Converting to one-hot labels

4 Feature Engineering

Three main NLP (Natural Language Processing) methods are used - Word2vec, Glove, ESM(Evolutionary Scaling Modeling).

4.1 Word2vec

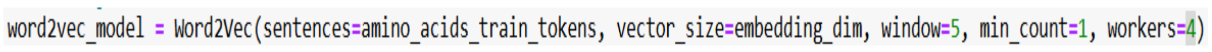
- For the successful extraction of embeddings from amino acid strings using word2vec, the 'gensim' library from python is installed and imported the 'word2vec' package from the library as shown in fig 4.



```python
from gensim.models import Word2Vec
```

Figure 4: Importing word2vec

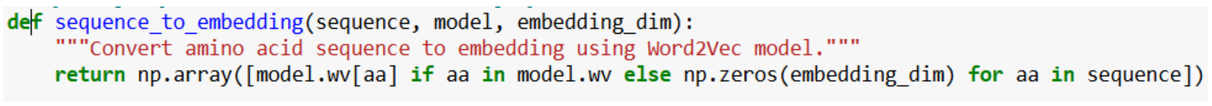
For the Word2vec embedding extraction:
Install 'gensim', define parameters a vector size of 50, Minimum Count (min count) defined to 1, and workers as 4. Then train a word2vec model on the amino acid sequence, shown in fig 5.



```python
word2vec_model = Word2Vec(sentences=amino_acids_train_tokens, vector_size=embedding_dim, window=5, min_count=1, workers=4)
```

Figure 5: Training word2vec model

Then create a function as given in fig 6, to retrieve the embeddings. Extract the embeddings by calling the function and save as 'train_embeddings' and 'test_embeddings'.



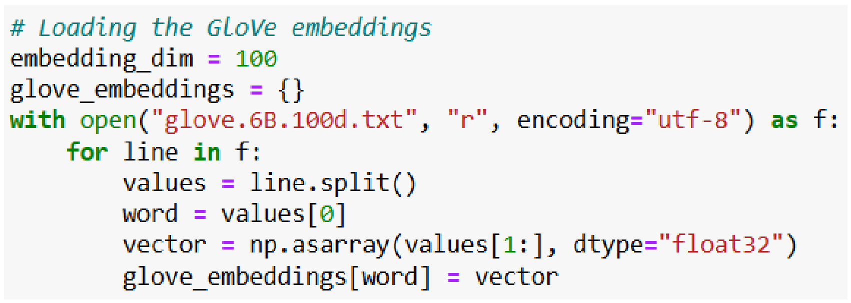
```python
def sequence_to_embedding(sequence, model, embedding_dim):
    """Convert amino acid sequence to embedding using Word2Vec model."""
    return np.array([model.wv[aa] if aa in model.wv else np.zeros(embedding_dim) for aa in sequence])
```

Figure 6: Function to extract word2vec embeddings

4.2 Glove

- For the extraction using Glove (Pennington et al. (2014)), the "glove.6B.100d.txt" embeddings file is downloaded from its official site: `https://nlp.stanford.edu/projects/glove/`.

For embedding extraction using Glove:
Create an embedding matrix to load the Glove embeddings which maps each word to its pre-trained vector according to the standard vector representation, according to the code snippet in fig 7. Then generate an embedding matrix which retrieves the vectors for each amino acid sequence.

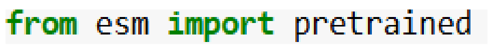


```python
# Loading the GloVe embeddings
embedding_dim = 100
glove_embeddings = {}
with open("glove.6B.100d.txt", "r", encoding="utf-8") as f:
    for line in f:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], dtype="float32")
        glove_embeddings[word] = vector
```

Figure 7: Loading Glove vector representation

4.3 ESM

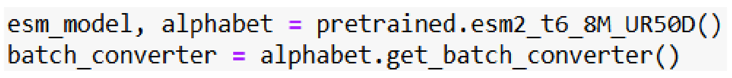
- The ESM embeddings (Lin et al. (2022)) are utilized by installing the ”fair-esm” library in python and imported the package ‘pretrained’, shown in fig 8.



```python
from esm import pretrained
```

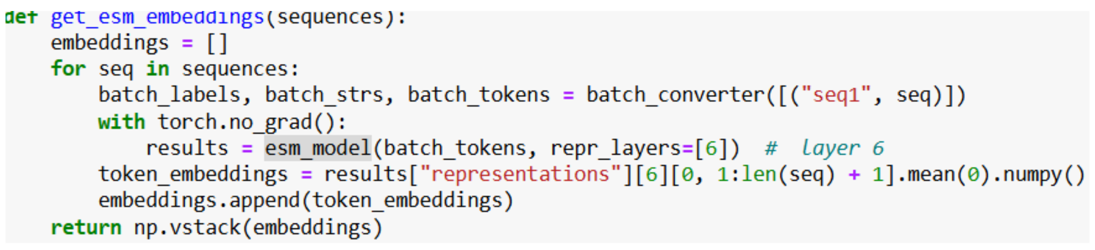
Figure 8: Importing ‘pretrained’ package

For embedding extraction using ESM: Load the latest version of pre-Trained ESM model, ‘ESM2-t6-8M UR50D’, shown in fig 9. Next, create a function which extracts the embeddings of amino acid according to the latest ESM version, fig 10. Then, pass the amino acid sequences to the function and save the resultant embeddings.



```python
esm_model, alphabet = pretrained.esm2_t6_8M_UR50D()
batch_converter = alphabet.get_batch_converter()
```

Figure 9: Loading the latest version of ESM

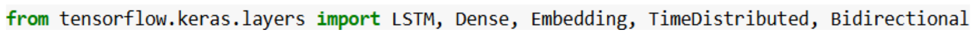


```python
def get_esm_embeddings(sequences):
    embeddings = []
    for seq in sequences:
        batch_labels, batch_strs, batch_tokens = batch_converter([("seq1", seq)])
        with torch.no_grad():
            results = esm_model(batch_tokens, repr_layers=[6])  #  layer 6
        token_embeddings = results["representations"][6][0, 1:len(seq) + 1].mean(0).numpy()
        embeddings.append(token_embeddings)
    return np.vstack(embeddings)
```

Figure 10: Function to extract ESM embeddings

5 Model Training

The LSTM model and the BiLSTM models are implemented utilizing the ‘keras’ package from ’tensorflow’ library, as it is widely used for deep learning (Srushti et al. (2023)). The packages such as ‘LSTM’, ‘Dense’, ‘Embedding’, ‘Dropout’, ‘TimeDistributed’, ‘Bidirectional’ etc are imported for the enabling all sufficient layers for the model as shown in fig 11.

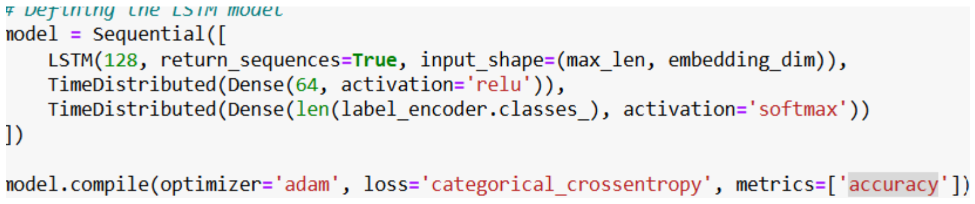


```python
from tensorflow.keras.layers import LSTM, Dense, Embedding, TimeDistributed, Bidirectional
```

Figure 11: Importing libraries for model training

5.1 LSTM Model:

First, define an embedding layer which inputs the embeddings, which is folliowed by an LSTM layer of 128 units. Then add a TimeDistributed dense layer with ’ReLU’ activation, which is followed by another dense layer with 'softmax' activation. Finally add the output layer. The complete model definition is shown in 12.



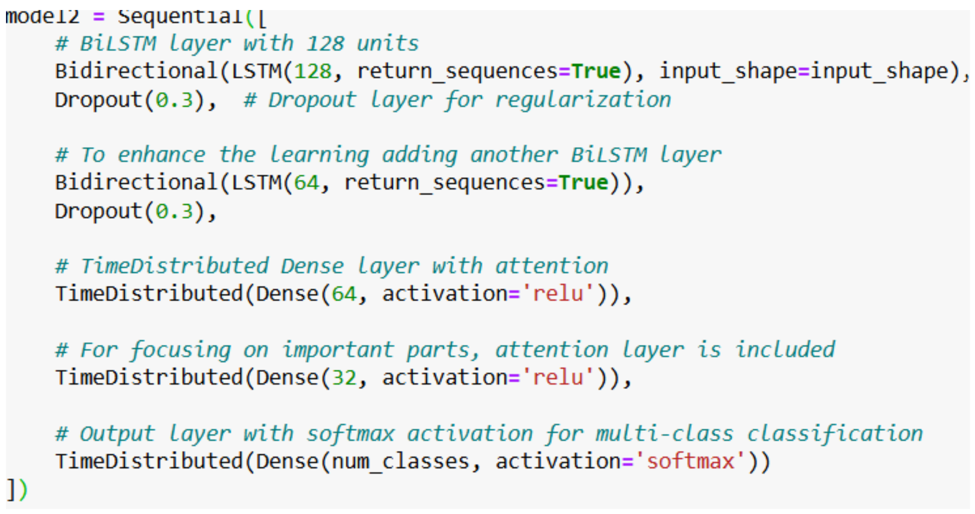
```
# Defining the LSTM model
model = Sequential([
    LSTM(128, return_sequences=True, input_shape=(max_len, embedding_dim)),
    TimeDistributed(Dense(64, activation='relu')),
    TimeDistributed(Dense(len(label_encoder.classes_), activation='softmax'))
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Figure 12: LSTM model

Compile the model using 'Adam' optimizer, then train each model for 10 epocs and batch size as 32.

5.2 BiLSTM with attention mechanism

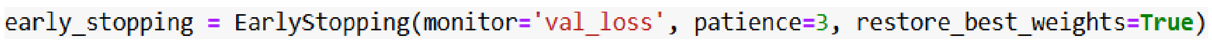
For hyperparametric tuning, first define a model with a BiLSTM layer contains 128 units, followed by the second BiLSTM layer with 64 units. Then, add a TimeDistributed dense layer with 64 units with 'ReLU' activation function and then add the second TimeDistributed dense layer with 32 units. Finally, add the output layer with 'Softmax' activation function. The model definition is shown in fig 13.



```
model2 = Sequential([
    # BiLSTM layer with 128 units
    Bidirectional(LSTM(128, return_sequences=True), input_shape=input_shape),
    Dropout(0.3),  # Dropout layer for regularization

    # To enhance the learning adding another BiLSTM layer
    Bidirectional(LSTM(64, return_sequences=True)),
    Dropout(0.3),

    # TimeDistributed Dense layer with attention
    TimeDistributed(Dense(64, activation='relu')),

    # For focusing on important parts, attention layer is included
    TimeDistributed(Dense(32, activation='relu')),

    # Output layer with softmax activation for multi-class classification
    TimeDistributed(Dense(num_classes, activation='softmax'))
])
```

Figure 13: BiLSTM model

After the model defining, define Early Stopping with a patience level of 3 and the parameter 'restore best weights' is assigned as True. Define the early stopping function as fig 14.



```
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
```

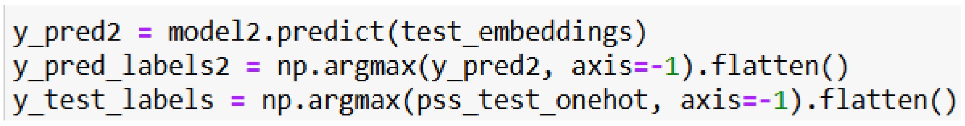
Figure 14: Early stopping regularization

Then, compile the model using 'Adam' optimizer and train each model for 20 epocs.

6 Evaluation

The models are evaluated by metrics like accuracy, precision , f1-score and recall. These metrics are utilized by importing packages ‘classification_report’, ‘accuracy_score’ from python ‘scikit-learn’ library (Ghosh and Shill (2021)).

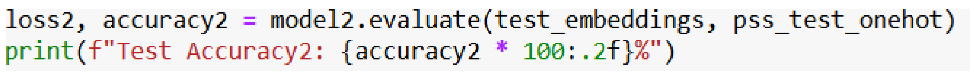
- Predict the values for the test using predict() function, refer fig 15.



```
y_pred2 = model2.predict(test_embeddings)
y_pred_labels2 = np.argmax(y_pred2, axis=-1).flatten()
y_test_labels = np.argmax(pss_test_onehot, axis=-1).flatten()
```

Figure 15: Prediction of secondary structure

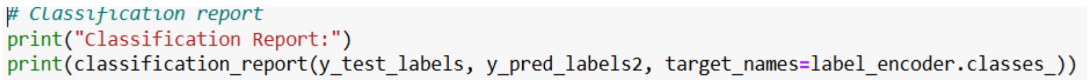
- Calculate the overall accuracy using the evaluate() function as shown in fig 16.



```
loss2, accuracy2 = model2.evaluate(test_embeddings, pss_test_onehot)
print(f"Test Accuracy2: {accuracy2 * 100:.2f}%")
```

Figure 16: Evaluation of model

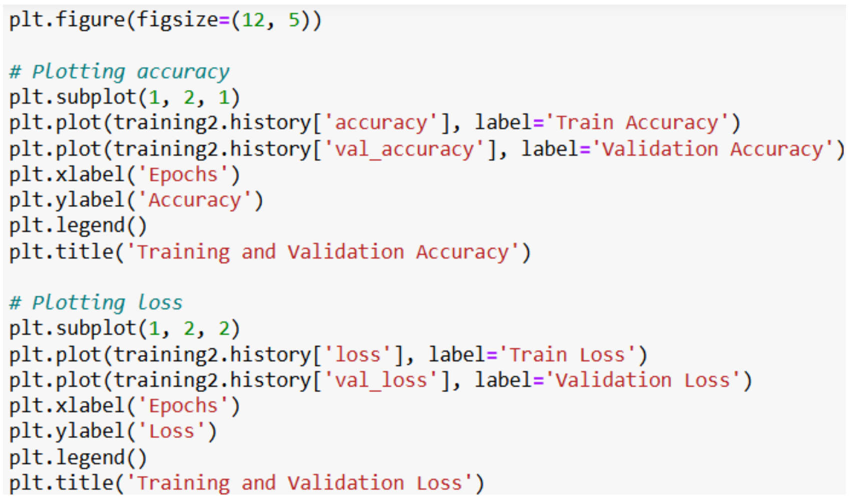
- Print the classification report, refer fig 17.



```
# Classification report
print("Classification Report:")
print(classification_report(y_test_labels, y_pred_labels2, target_names=label_encoder.classes_))
```

Figure 17: Printing classification report

- Generate accuracy and loss plots for the model before and after tuning, refer 18.



```
plt.figure(figsize=(12, 5))

# Plotting accuracy
plt.subplot(1, 2, 1)
plt.plot(training2.history['accuracy'], label='Train Accuracy')
plt.plot(training2.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')

# Plotting loss
plt.subplot(1, 2, 2)
plt.plot(training2.history['loss'], label='Train Loss')
plt.plot(training2.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')
```

Figure 18: Plotting accuracy and loss plots

- Compare the accuracy, precision and plots of all models.

References

Ghosh, S. and Shill, P. C. (2021). Protein secondary structure detection without alignment by recurrent neural network with lstm, *2021 5th International Conference on Electrical Information and Communication Technology (EICT)*, pp. 1–6.

Lin, Z., Akin, H., Rao, R., Hie, B., Zhu, Z., Lu, W., Smetanin, N., Verkuil, R., Kabeli, O., Shmueli, Y., dos Santos Costa, A., Fazel-Zarandi, M., Sercu, T., Candido, S. et al. (2022). Language models of protein sequences at the scale of evolution enable accurate structure prediction, *bioRxiv* .

Pennington, J., Socher, R. and Manning, C. D. (2014). Glove: Global vectors for word representation, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Association for Computational Linguistics, pp. 1532–1543.
URL: *https://www.aclweb.org/anthology/D14-1162*

Srushti, C., Prathibhavani, P. and Venugopal, K. (2023). Eight-state accuracy prediction of protein secondary structure using ensembled model, *2023 International Conference for Advancement in Technology (ICONAT)*, IEEE, pp. 1–6.

Zhou, J. and Troyanskaya, O. G. (2014). Deep supervised and convolutional generative stochastic network for protein secondary structure prediction, *Proceedings of the 31st International Conference on Machine Learning (ICML)*, Vol. 32, JMLR: W&CP, Beijing, China.
URL: *https://proceedings.mlr.press/v32/zhou14.pdf*