National College of Ireland

Configuration Manual: Protein Sequence Classification using Machine Learning and Deep Learning

MSc Research Project
Data Analytics

Shravanee
Shekhar Siddha

Student ID: x18180949

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Shravanee Shekhar Siddha
Student ID:	x18180949
Programme:	Data Analytics
Year:	2019-2020
Module:	M.Sc. Research Project
Supervisor:	Dr.Catherine Mulwa
Submission Due Date:	17/08/2020
Project Title:	Configuration manual: Protein Sequence Classification using Machine Learning and Deep Learning
Word Count:	1152
Page Count:	13

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Shravanee Shekhar Siddha
Date:	17/08/2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	Q
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	Q
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.	Q

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

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

Configuration Manual

Shravanee Shekhar Siddha
x18180949

1 Introduction

The configuration manual demonstrates the implementation phases of the project "Protein Sequence Classification using machine learning and deep learning". The main objective of this project is to provide an efficient protein sequence classification system. In order to build the model, a combination of techniques like Natural Language Processing for feature extraction like TF-IDF along with machine learning algorithms such as Decision Tree and Random Forest and Word Embedding using keras with deep learning models like Convolutional Neural Network and Long Short-Term Memory were implemented This configuration manual contains the required project specifications for hardware and software to implement the project in Chapter 2. Chapter 3 explains the Data Preparation followed by Chapter 4 which describes the implementation steps in detail and the output generated.

2 Hardware Specification

Operating System	Windows 10 Home Single Language
Processor	Intel(R) Core (TM) i5-8250U
Installed Memory (RAM)	8.00GB
System type	64-bit Operating System

3 Software Specification

- Installation of Anaconda and Python 3 version

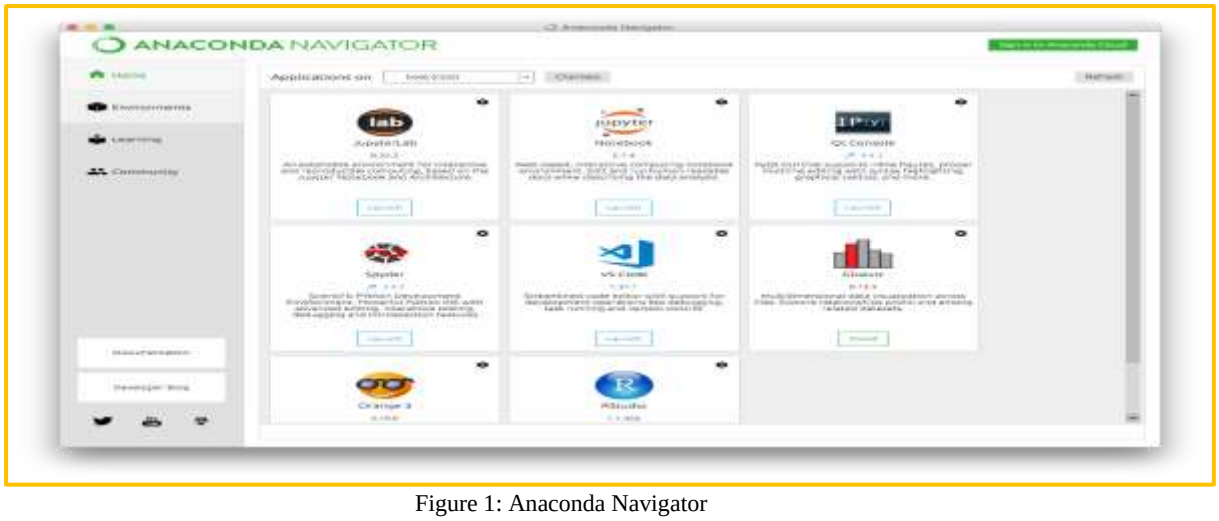

Figure 1: Anaconda Navigator

4 Data Preparation

The dataset is collected from Kaggle and was available in .csv format. The following link is to the dataset:

https://www.kaggle.com/shahir/protein-data-set

Installing Tensorflow and importing Keras for using Deep Learning.

Figure 2: Installation of Tensorflow and importing Keras

Necessary libraries were imported before performing data pre-processing and EDA.

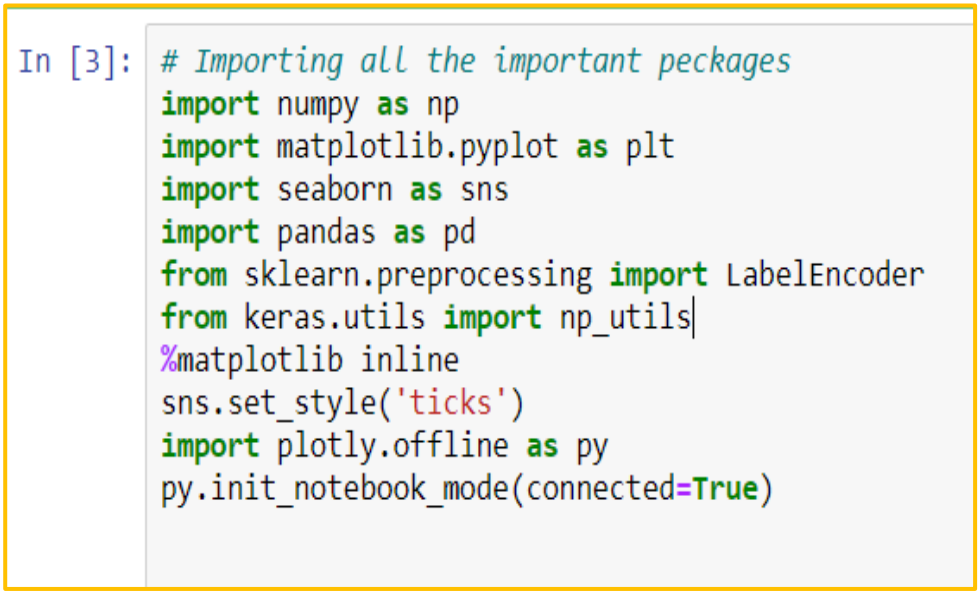

Figure 3: Importing libraries for EDA

The data was in the form of csv. Initially, it was loaded in Pandas Data Frame using the read csv function. The irrelevant columns and missing values were then removed and the required column labels were made appropriate. The following figure 4 shows the data after cleaning:

	structureId	classification	chainId	sequence	seq_length
67	117E	HYDROLASE	A	TYTTRQIGAKNTLEYKVYIEKDGKPVSAFHDIPLYADKENNIFNMV...	286
68	117E	HYDROLASE	B	TYTTRQIGAKNTLEYKVYIEKDGKPVSAFHDIPLYADKENNIFNMV...	286
74	11AS	LIGASE	A	MKTAYIAKQRQISFVKSHFSRQLEERLGLIEVQAPILSRVGDGTQD...	330
75	11AS	LIGASE	B	MKTAYIAKQRQISFVKSHFSRQLEERLGLIEVQAPILSRVGDGTQD...	330
76	11BA	HYDROLASE	A	KESAAAKFERQHMDSGNSPSSSSNYCNLMMCCRKMTQGKCKPVNTF...	124

Figure 4: Cleaned data

The dataset was used for Exploratory Data Analysis (EDA), to get better understanding of the data by using simple, creative visualizations.

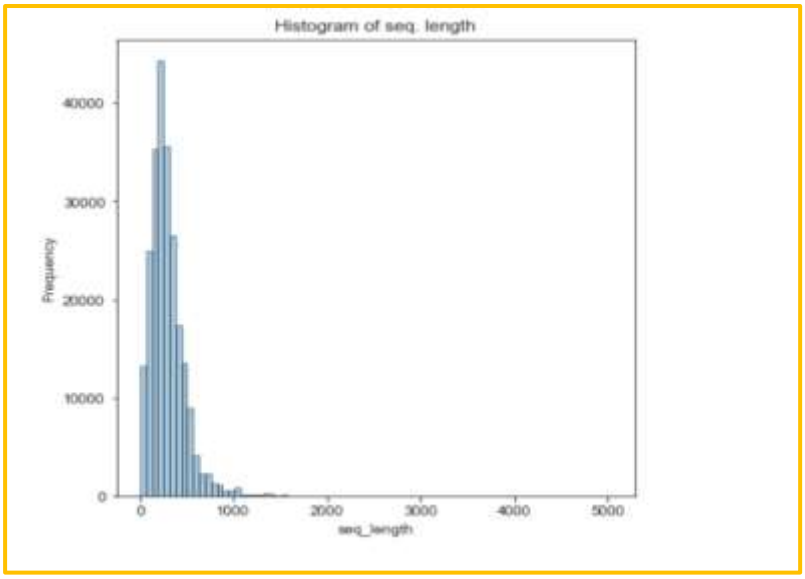

Figure 5: Length of protein sequences

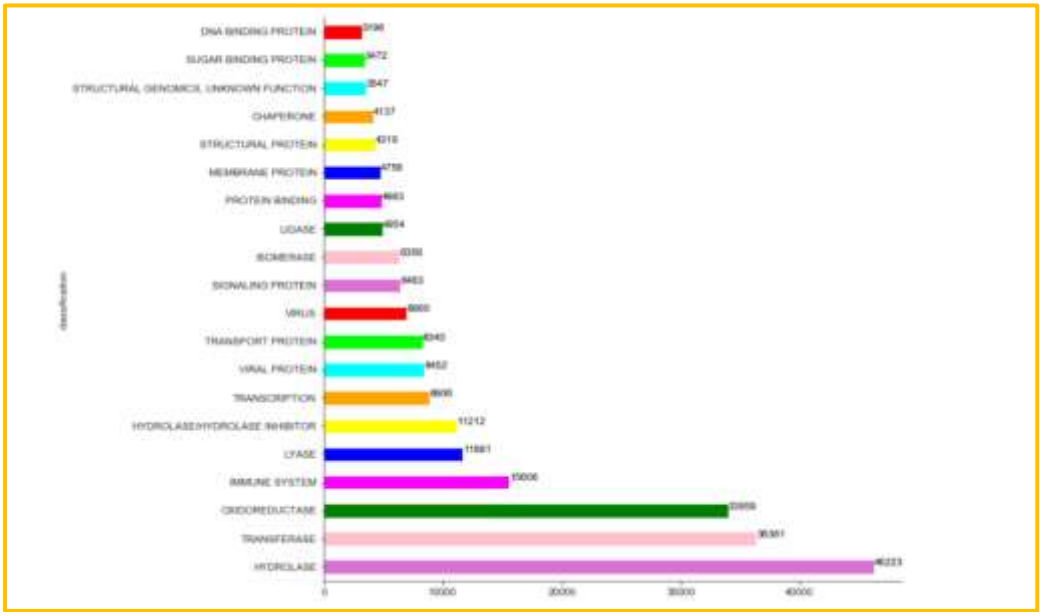

Figure 6: Bar plot with top 20 frequently occurring classes

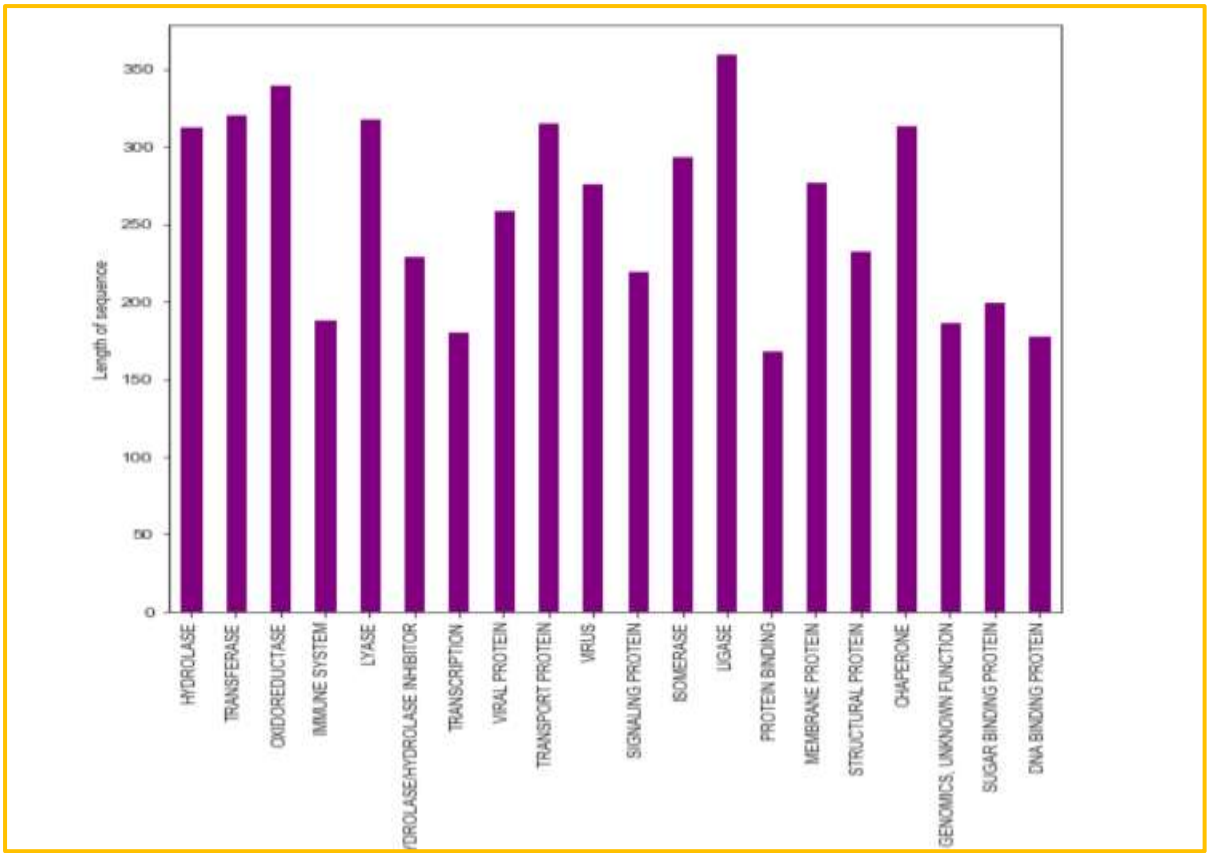

Figure 7: Imbalanced data showing variations in the length of sequences

5 Implementation, Evaluation and Results of Protein Sequence Classification Models

```
from sklearn.feature_extraction.text import TfidfVectorizer
from keras.preprocessing import text, sequence
from keras.preprocessing.text import Tokenizer

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers



from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

from imblearn.under_sampling import RandomUnderSampler

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

from keras.models import Sequential
from keras.layers import Dense, Conv1D, MaxPooling1D, Flatten
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from sklearn.metrics import accuracy_score, classification_report
```

Figure 8: Importing required libraries

Importing the libraries required for feature extraction, feature selection, data balancing and implementation of machine learning and deep learning models.

5.1 Transforming labels into numeric representations using Label Encoder

The labels/classes were transformed using LabelEncoder() function to categorical values.

```
#Using labelencoder() to lables into numeric labels
lb = LabelEncoder()
lb_fit = lb.fit(df_protein.classification)
Y = lb_fit.transform(df_protein.classification)
Y = np_utils.to_categorical(Y)
```

Figure 9: Using Label Encoder

5.2 Transform Sequences into Numeric Vectors

Using TF-IDFVeectorizer() function with n-gram range (4,4) for converting sequences into numeric representations. Transforming the features into array by using toarray() function.

```
#Using TF-IDFVectorizer() with n-gram for conevrsion of protein sequences into vector
funct = TfidfVectorizer(analyzer = "char_wb", ngram_range= (4,4), max_features= 1000)
funct.fit(df_protein['sequence'])
vector = funct.transform(df_protein['sequence'])

# Conevrting numeric representation to array
vector = vector.toarray()
print(vector)
```

Figure 9: Using TFIDF

Using tokenizer() for representing the sequences into numeric vectors which is passed as an embedding layer using Keras. The pad_sequences is used to have fixed length

```
#create and fit tokenize and then represent input data as word rank number sequences
max_length = 1000
seqs = df_protein.sequence.values
tokenizer = Tokenizer(char_level=True)
tokenizer.fit_on_texts(seqs)
vector1 = tokenizer.texts_to_sequences(seqs)
vector1 = sequence.pad_sequences(vector1, maxlen=max_length)
print(vector1)
```

Figure 10: Using Tokenizer

5.3 Feature Selection using chi2 technique

Chi2 and SelectKBest were used for passing relevant features to the models.

```
# Using chi2 and selectKBest for feature selection
chi_select = SelectKBest(chi2, k = 500)
X = chi_select.fit_transform(vector, Y)
print (X)
```

Figure 11: Feature selection for machine learning models

```
# Using chi2 and selectKBest for feature selection
chi_select = SelectKBest(chi2, k = 500)
X = chi_select.fit_transform(vector1, Y)
print (X)
```

Figure 12: Feature selection for deep learning models

5.4 Using Random Under-sampling for over-represented classes

The dataset was imbalanced and hence, random under-sampling was used for to lessen the number of majority classes. The imblearn library offers the function RandomUnderSampler().

```
#Resampling the data using RandomUnderSampler()
undersample = RandomUnderSampler(sampling_strategy='majority')
X_over, y_over = undersample.fit_resample(X, Y)
```

Figure 13: Random Under-sampling for balancing the data

5.5 Implementation Evaluation and Results of Decision Tree

Decision Tree model was implemented and performed the best of all the models with an accuracy of 78.71%. The classification report was also developed.

```
# Implementing Decision tree and printing accuracy and classification report
dtree = DecisionTreeClassifier()
dtree.fit(X_train, y_train)

predictions = dtree.predict(X_test)
print(classification_report(y_test, predictions, target_names=lb.classes_))
DT_accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", DT_accuracy)
```

Figure 14: Decision Tree Classifier

```
                                   precision    recall  f1-score

                        CHAPERONE       0.93      0.82      0.87
                DNA BINDING PROTEIN       0.84      0.67      0.75
                        HYDROLASE       0.66      0.32      0.44
      HYDROLASE/HYDROLASE INHIBITOR       0.92      0.81      0.86
                    IMMUNE SYSTEM       0.94      0.77      0.84
                        ISOMERASE       0.92      0.82      0.87
                           LIGASE       0.89      0.78      0.83
                            LYASE       0.94      0.90      0.92
                 MEMBRANE PROTEIN       0.85      0.71      0.78
                   OXIDOREDUCTASE       0.95      0.88      0.91
                  PROTEIN BINDING       0.80      0.52      0.63
                SIGNALING PROTEIN       0.86      0.62      0.72
STRUCTURAL GENOMICS, UNKNOWN FUNCTION       0.80      0.68      0.73
               STRUCTURAL PROTEIN       0.88      0.66      0.76
            SUGAR BINDING PROTEIN       0.93      0.70      0.80
                     TRANSCRIPTION       0.85      0.57      0.68
                       TRANSFERASE       0.92      0.86      0.89
                TRANSPORT PROTEIN       0.91      0.76      0.83
                    VIRAL PROTEIN       0.91      0.78      0.84
                            VIRUS       0.97      0.90      0.93

                         micro avg       0.92      0.79      0.85
                         macro avg       0.88      0.73      0.79
                      weighted avg       0.91      0.79      0.84
                       samples avg       0.79      0.79      0.79

Accuracy: 0.7871615422924694
```

Figure 15: Classification Report for Decision Tree

5.6 Implementation, Evaluation and Results of Random Forest

Random Forest Classifier was implemented and it also, achieved a good accuracy of 77.24%. The RandomForestClassifier() was used for executing the model. The classification report was evaluated for the multi-class labels.

```
# Implementing Random Forest and printing accuracy and classification report
rf = RandomForestClassifier()
rf.fit(X_train, y_train)

predictions = rf.predict(X_test)
print(classification_report(y_test, predictions,target_names=lb.classes_))
RF_accuracy = accuracy_score(y_test, predictions)
print("Accuracy :",RF_accuracy)
```

Figure 16: Random Forest Classifier

```
                                        precision    recall  f1-score

                               CHAPERONE       0.96      0.80      0.87
                      DNA BINDING PROTEIN       0.88      0.64      0.74
                               HYDROLASE       0.77      0.30      0.43
             HYDROLASE/HYDROLASE INHIBITOR       0.93      0.81      0.87
                           IMMUNE SYSTEM       0.95      0.76      0.85
                               ISOMERASE       0.95      0.81      0.87
                                  LIGASE       0.93      0.75      0.83
                                   LYASE       0.96      0.89      0.92
                        MEMBRANE PROTEIN       0.85      0.71      0.77
                          OXIDOREDUCTASE       0.98      0.86      0.92
                         PROTEIN BINDING       0.85      0.49      0.62
                       SIGNALING PROTEIN       0.91      0.59      0.72
STRUCTURAL GENOMICS, UNKNOWN FUNCTION          0.89      0.62      0.73
                      STRUCTURAL PROTEIN       0.90      0.65      0.76
                  SUGAR BINDING PROTEIN       0.96      0.68      0.80
                           TRANSCRIPTION       0.87      0.56      0.68
                             TRANSFERASE       0.97      0.84      0.90
                       TRANSPORT PROTEIN       0.94      0.74      0.83
                           VIRAL PROTEIN       0.93      0.76      0.84
                                   VIRUS       0.97      0.90      0.93

                               micro avg       0.94      0.77      0.85
                               macro avg       0.92      0.71      0.79
                            weighted avg       0.94      0.77      0.84
                             samples avg       0.77      0.77      0.77

Accuracy : 0.7724568163350896
```

Figure 17: Classification Report for Random Forest

5.7 Implementation, Evaluation and Results of Convolutional Neural Network

Convolutional Neural Network was built by using embedding layer and giving the numeric representations as input. The model was compiled using categorical crossentropy and adam

optimizer. The model achieved an accuracy of 75%. The classification report for CNN was also used as an evaluation metric.

```
# Model building CNN and using embedding layer has been initailized
# Printing the summary of the model
embedding_dim = 8
top_classes = 20
model = Sequential()
model.add(Embedding(len(tokenizer.word_index)+1, embedding_dim, input_length= 500))
model.add(Conv1D(filters=64, kernel_size=6, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(top_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

Figure 18: Convolutional Neural Network using Embedding Layer

	precision	recall	f1-score
CHAPERONE	0.96	0.80	0.87
DNA BINDING PROTEIN	0.88	0.64	0.74
HYDROLASE	0.77	0.30	0.43
HYDROLASE/HYDROLASE INHIBITOR	0.93	0.81	0.87
IMMUNE SYSTEM	0.95	0.76	0.85
ISOMERASE	0.95	0.81	0.87
LIGASE	0.93	0.75	0.83
LYASE	0.96	0.89	0.92
MEMBRANE PROTEIN	0.85	0.71	0.77
OXIDOREDUCTASE	0.98	0.86	0.92
PROTEIN BINDING	0.85	0.49	0.62
SIGNALING PROTEIN	0.91	0.59	0.72
STRUCTURAL GENOMICS, UNKNOWN FUNCTION	0.89	0.62	0.73
STRUCTURAL PROTEIN	0.90	0.65	0.76
SUGAR BINDING PROTEIN	0.96	0.68	0.80
TRANSCRIPTION	0.87	0.56	0.68
TRANSFERASE	0.97	0.84	0.90
TRANSPORT PROTEIN	0.94	0.74	0.83
VIRAL PROTEIN	0.93	0.76	0.84
VIRUS	0.97	0.90	0.93
micro avg	0.94	0.77	0.85
macro avg	0.92	0.71	0.79
weighted avg	0.94	0.77	0.84
samples avg	0.77	0.77	0.77

Accuracy : 0.7724568163350896

Figure 19: Classification Report for Convolutional Neural Network

5.8 Implementation, Evaluation and Results of Long Short-Term Memory

The Long Short-Term Memory was implemented by using embedding layer. This model used a single LSTM layer and dense layer as output layer. The model did not perform well and showed poor accuracy of %. A classification report for LSTM was also generated to investigate values for multi-class labels.

```
# Model Building LSTM and using embedding layer has been initailized
#Compiling the model
# Printing the summary of the model
embedding_dim = 8
lstm_out = 128
batch_size = 128
top_classes = 20

model1 = Sequential()
model1.add(Embedding(len(tokenizer.word_index)+1, 8))
model1.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model1.add(Dense(top_classes,activation='softmax'))
model1.compile(loss = 'categorical_crossentropy', optimizer='adam',metrics = ['accuracy'])
print(model1.summary())
```

Figure 20: Long Short-Term Memory using Embedding Layer

	precision	recall	f1-score
CHAPERONE	0.63	0.48	0.55
DNA BINDING PROTEIN	0.51	0.14	0.22
HYDROLASE	0.50	0.01	0.03
HYDROLASE/HYDROLASE INHIBITOR	0.76	0.67	0.71
IMMUNE SYSTEM	0.61	0.79	0.69
ISOMERASE	0.59	0.18	0.28
LIGASE	0.64	0.10	0.17
LYASE	0.54	0.35	0.42
MEMBRANE PROTEIN	0.56	0.23	0.33
OXIDOREDUCTASE	0.50	0.72	0.59
PROTEIN BINDING	0.18	0.05	0.07
SIGNALING PROTEIN	0.28	0.10	0.15
STRUCTURAL GENOMICS, UNKNOWN FUNCTION	0.20	0.02	0.03
STRUCTURAL PROTEIN	0.61	0.20	0.30
SUGAR BINDING PROTEIN	0.52	0.52	0.52
TRANSCRIPTION	0.27	0.36	0.31
TRANSFERASE	0.46	0.70	0.55
TRANSPORT PROTEIN	0.49	0.27	0.35
VIRAL PROTEIN	0.57	0.52	0.54
VIRUS	0.83	0.75	0.79
accuracy			0.51
macro avg	0.51	0.36	0.38
weighted avg	0.52	0.51	0.48

Figure 21: Classification report for LSTM

6 Comparison of the machine learning and deep learning models

The comparison of the developed models is done by using visualization in python. It is clearly seen that Decision Tree and Random Forest performed better than Convolutional Neural Network and Long Short-Term Memory. Hence, machine learning models with TF-IDF are efficient than deep learning models with Word Embedding.

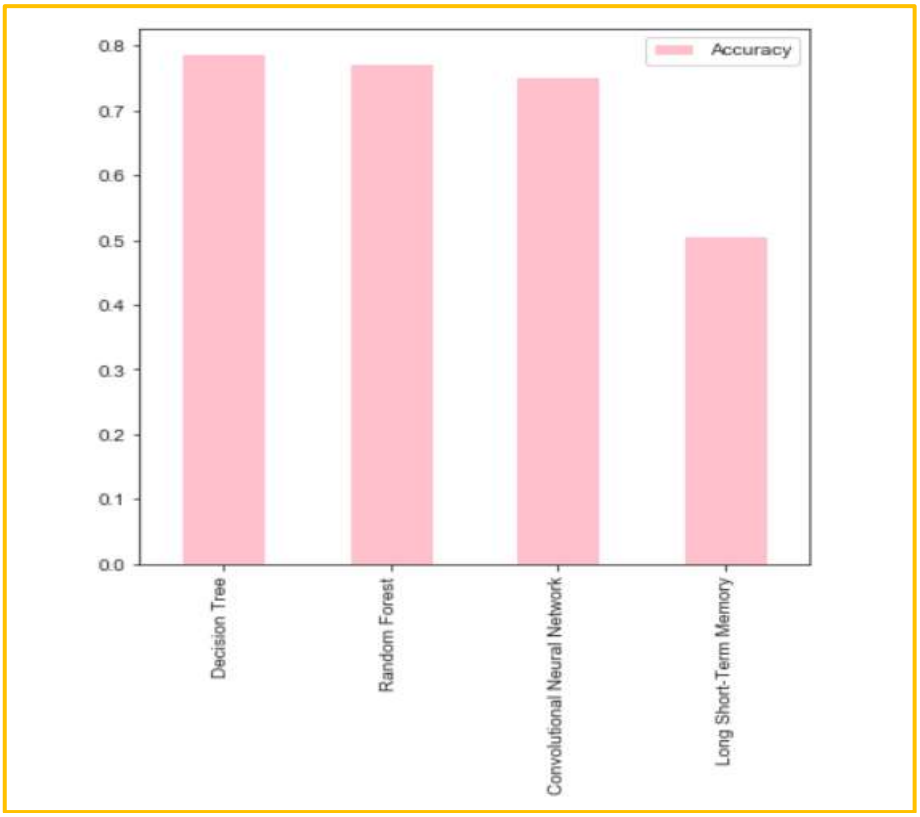

Figure 22: Comparison between the developed models

7 Conclusion

Both Machine Learning models performed well. But, Decision Tree out-performed all the models. Random Forest also achieved good performance. Convolutional Neural Network achieved accuracy slightly less than Random Forest and Long Short-Term Memory showed lowest accuracy of all the models.
Thus, Machine Learning models performed well with TF-IDF as feature extraction technique than the Deep Learning models.

References

Mullane, S., Chen, R., Vemulapalli, V., Draizen, E., Wang, K., Mura, C. and Bourne, P. (2019). Machine Learning for Classification of Protein Helix Capping Motifs, Systems and Information Engineering Design Symposium (SIEDS), DOI:10.1109/SIEDS.2019.8735646