Toxin Data Preparation

Available dataset:

- protein_train1002.csv
- protein_test1002.csv
- peptide.csv

1. protein_train1002.csv:

- Total no. of sequences: 10084

Non-toxic	5671 (56.2%)
Toxic	4413 (43.8%)

- Now, we extracted peptides from protein_train1002.csv:

Number of Peptide sequences with length=1755

Non-toxic	656 (37.4%)
Toxic	1099 (62.6%)

- Now, we extracted protein from protein_train1002.csv:

Number of protein sequences with length= 8329

Non-toxic	5015 (60.2%)
Toxic	3314 (39.8%)

2. protein_test1002.csv:

- Total no. of sequences: 729

Non-toxic	670 (91.9%)
Toxic	59 (8.1%)

- Now we extracted peptides from protein_test1002.csv:

Number of Peptide sequences with length=2

Non-toxic	0 (0%)
Toxic	2 (100%)

- Now we extracted protein from protein_test1002.csv:

Number of Protien sequences with length= 727

Non-toxic	670 (92.2%)
Toxic	57 (7.8%)

3. Peptide.csv

- Total no. of peptide sequences: 3864

Non-toxic	1932(50%)
Toxic	1932(50%)

Now, we further combined the data to be used to ML models:

- Total number of Peptide sequences with length : 5621 (3864+2+1755)

Non-toxic	2588 (46.0%)
Toxic	3033(54.0%)

- Total number of Protein sequences with length : 9056 (727+8329)

Non-toxic	5685 (62.8%)
Toxic	3371(37.2%)

Experiment 1:(AAC + DPC)

1.Protein Prediction:

```
# Define the feature extraction functions
def calculate_aac(sequence):
    amino_acids = 'ACDEFGHIKLMNPQRSTVWY'
    sequence_length = len(sequence)
    aa_count = Counter(sequence)
    aac = [aa_count[aa] / sequence_length for aa in amino_acids]
    return aac

def calculate_dpc(sequence):
    amino_acids = 'ACDEFGHIKLMNPQRSTVWY'
    dipeptides = [''.join(pair) for pair in itertools.product(amino_acids, repeat=2)]
    dipeptide_count = Counter([sequence[i:i+2] for i in range(len(sequence)-1)])
    sequence_length = len(sequence) - 1
    dpc = [dipeptide_count[dp] / sequence_length for dp in dipeptides]
    return dpc

# Extract features for a given dataset with renamed columns to avoid overlap
def extract_features(data):
    aac_features = data['sequence'].apply(calculate_aac)
    dpc_features = data['sequence'].apply(calculate_dpc)

    aac_df = pd.DataFrame(aac_features.tolist(), columns=[f'aac_{i}' for i in range(20)])
    dpc_df = pd.DataFrame(dpc_features.tolist(), columns=[f'dpc_{i}' for i in range(400)])

    features = aac_df.join(dpc_df)

    return features
```

This code processes a dataset containing protein sequences to extract amino acid composition (AAC) and dipeptide composition (DPC) features. It calculates AAC and DPC for each sequence, constructs a comprehensive feature set, and saves the features along with the corresponding labels to a CSV file for further analysis.

The extracted features from the protein sequences were loaded from a CSV file and split into training (60%), validation (20%), and testing (20%) sets. The LazyPredict library was utilized to evaluate multiple classification models' performance.

Below mentioned are the top 10 results:

```
                               Accuracy  Balanced Accuracy  ROC AUC  F1 Score  \
Model
LGBMClassifier                     0.95               0.95     0.95      0.95
XGBClassifier                      0.95               0.95     0.95      0.95
SVC                                0.95               0.95     0.95      0.95
ExtraTreesClassifier               0.95               0.94     0.94      0.95
RandomForestClassifier             0.94               0.94     0.94      0.94
BaggingClassifier                  0.93               0.92     0.92      0.93
LinearDiscriminantAnalysis         0.93               0.92     0.92      0.93
RidgeClassifierCV                  0.93               0.91     0.91      0.93
RidgeClassifier                    0.93               0.91     0.91      0.93
LogisticRegression                 0.92               0.91     0.91      0.92
```

```
                            Time Taken
Model
LGBMClassifier                    8.65
XGBClassifier                     7.14
SVC                               5.08
ExtraTreesClassifier              2.06
RandomForestClassifier            6.91
BaggingClassifier                19.52
LinearDiscriminantAnalysis        1.07
RidgeClassifierCV                 1.14
RidgeClassifier                   0.44
LogisticRegression                0.92
```

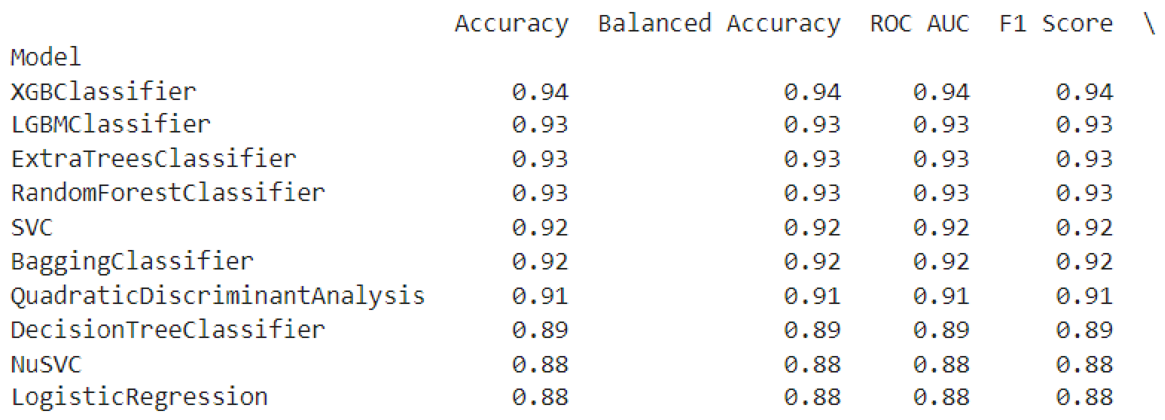
The top-performing model from LazyPredict was selected, its performance was then evaluated on both the validation and test sets, with accuracy results printed for each set.

Best Model	LGBMClassifier=0.95
Validation Accuracy	0.9531
Test Accuracy	0.9570

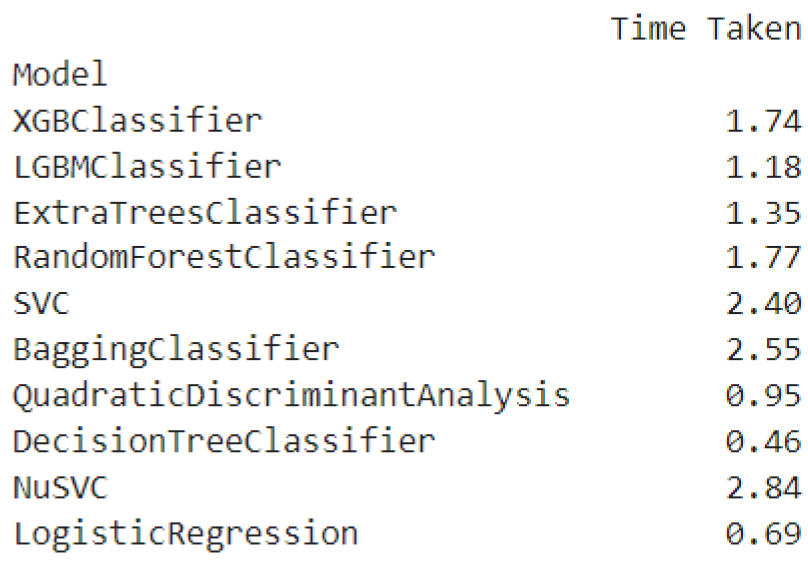
2. Peptide prediction:

Same like Protein we now, extracts amino acid composition (AAC) and dipeptide composition (DPC) features from peptide sequences, saves them to a CSV file, and splits the data into training (60%), validation (20%), and testing (20%) sets and using LazyPredict, multiple classification models were evaluated.

Below mentioned are the top 10 results:



```
                               Accuracy  Balanced Accuracy  ROC AUC  F1 Score  \
Model
XGBClassifier                      0.94               0.94     0.94      0.94
LGBMClassifier                     0.93               0.93     0.93      0.93
ExtraTreesClassifier               0.93               0.93     0.93      0.93
RandomForestClassifier             0.93               0.93     0.93      0.93
SVC                                0.92               0.92     0.92      0.92
BaggingClassifier                  0.92               0.92     0.92      0.92
QuadraticDiscriminantAnalysis      0.91               0.91     0.91      0.91
DecisionTreeClassifier             0.89               0.89     0.89      0.89
NuSVC                              0.88               0.88     0.88      0.88
LogisticRegression                 0.88               0.88     0.88      0.88
```



```
                               Time Taken
Model
XGBClassifier                        1.74
LGBMClassifier                       1.18
ExtraTreesClassifier                 1.35
RandomForestClassifier               1.77
SVC                                  2.40
BaggingClassifier                    2.55
QuadraticDiscriminantAnalysis        0.95
DecisionTreeClassifier               0.46
NuSVC                                2.84
LogisticRegression                   0.69
```

The top-performing model from LazyPredict was selected, its performance was then evaluated on both the validation and test sets, with accuracy results printed for each set.

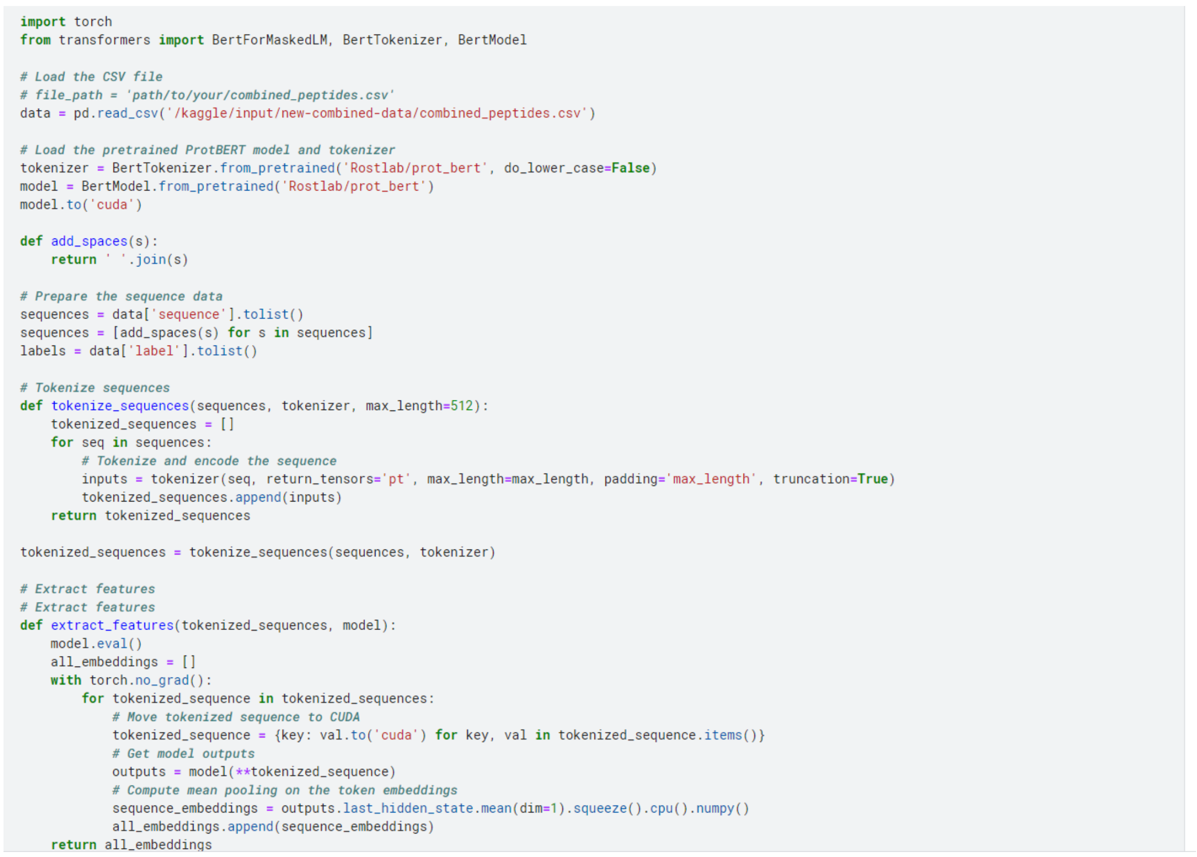
Best Model	XGBClassifier=0.94
Validation Accuracy	0.9359
Test Accuracy	0.9298

Experiment – 2: (ProtBert)

1: peptide prediction

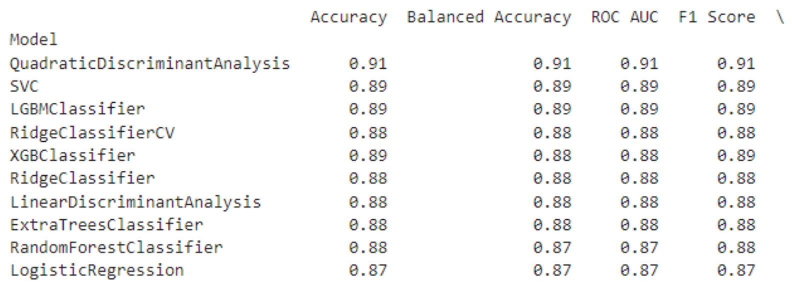
In experiment 2, we have used ProtBERT to extract features of peptides. ProtBERT is a protein language model inspired by the BERT architecture, designed to understand and generate protein sequences by learning the patterns and relationships between amino acids. These models, trained on extensive protein sequence datasets, can extract features that encapsulate the biological and chemical properties of the sequences.

The extracted feature embeddings are then saved in a CSV file. Subsequently, the data is split into training, testing, and validation sets in the following proportions: 60% for training, 20% for testing, and 20% for validation.



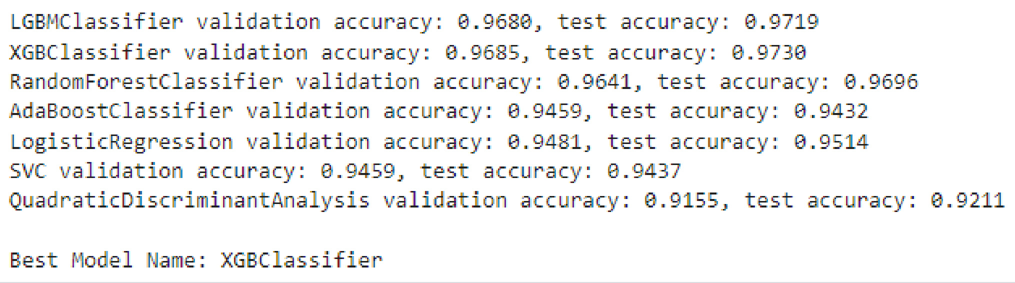
```python
import torch
from transformers import BertForMaskedLM, BertTokenizer, BertModel

# Load the CSV file
# file_path = 'path/to/your/combined_peptides.csv'
data = pd.read_csv('/kaggle/input/new-combined-data/combined_peptides.csv')

# Load the pretrained ProtBERT model and tokenizer
tokenizer = BertTokenizer.from_pretrained('Rostlab/prot_bert', do_lower_case=False)
model = BertModel.from_pretrained('Rostlab/prot_bert')
model.to('cuda')

def add_spaces(s):
    return ' '.join(s)

# Prepare the sequence data
sequences = data['sequence'].tolist()
sequences = [add_spaces(s) for s in sequences]
labels = data['label'].tolist()

# Tokenize sequences
def tokenize_sequences(sequences, tokenizer, max_length=512):
    tokenized_sequences = []
    for seq in sequences:
        # Tokenize and encode the sequence
        inputs = tokenizer(seq, return_tensors='pt', max_length=max_length, padding='max_length', truncation=True)
        tokenized_sequences.append(inputs)
    return tokenized_sequences

tokenized_sequences = tokenize_sequences(sequences, tokenizer)

# Extract features
# Extract features
def extract_features(tokenized_sequences, model):
    model.eval()
    all_embeddings = []
    with torch.no_grad():
        for tokenized_sequence in tokenized_sequences:
            # Move tokenized sequence to CUDA
            tokenized_sequence = {key: val.to('cuda') for key, val in tokenized_sequence.items()}
            # Get model outputs
            outputs = model(**tokenized_sequence)
            # Compute mean pooling on the token embeddings
            sequence_embeddings = outputs.last_hidden_state.mean(dim=1).squeeze().cpu().numpy()
            all_embeddings.append(sequence_embeddings)
    return all_embeddings
```

This is the result of top 10 models by using LazyPredict:



```
                              Accuracy  Balanced Accuracy  ROC AUC  F1 Score  \
Model
QuadraticDiscriminantAnalysis     0.91               0.91     0.91      0.91
SVC                               0.89               0.89     0.89      0.89
LGBMClassifier                    0.89               0.89     0.89      0.89
RidgeClassifierCV                 0.88               0.88     0.88      0.88
XGBClassifier                     0.89               0.88     0.88      0.89
RidgeClassifier                   0.88               0.88     0.88      0.88
LinearDiscriminantAnalysis        0.88               0.88     0.88      0.88
ExtraTreesClassifier              0.88               0.88     0.88      0.88
RandomForestClassifier            0.88               0.87     0.87      0.88
LogisticRegression                0.87               0.87     0.87      0.87
```

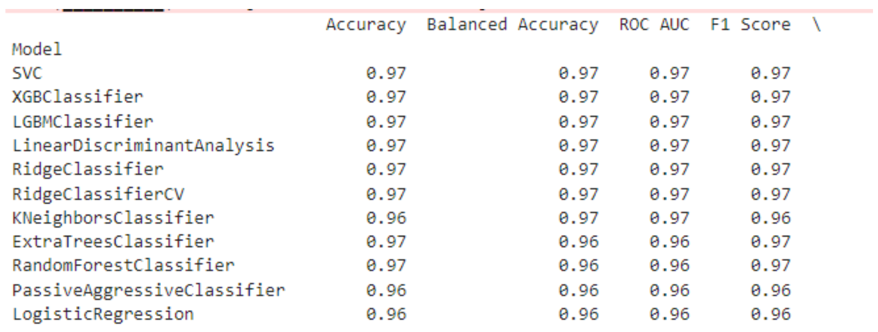
And here below is our Validation and Testing Accuracy:



```
LGBMClassifier validation accuracy: 0.9680, test accuracy: 0.9719
XGBClassifier validation accuracy: 0.9685, test accuracy: 0.9730
RandomForestClassifier validation accuracy: 0.9641, test accuracy: 0.9696
AdaBoostClassifier validation accuracy: 0.9459, test accuracy: 0.9432
LogisticRegression validation accuracy: 0.9481, test accuracy: 0.9514
SVC validation accuracy: 0.9459, test accuracy: 0.9437
QuadraticDiscriminantAnalysis validation accuracy: 0.9155, test accuracy: 0.9211

Best Model Name: XGBClassifier
```

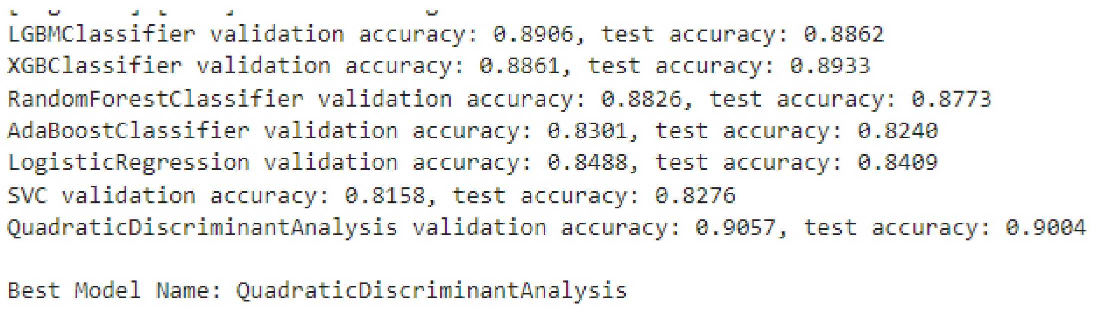
2 : **Protein**

This is the result of top 10 models by using LazyPredict after using the same ProtBert Embedding:



```
                               Accuracy  Balanced Accuracy  ROC AUC  F1 Score  \
Model
SVC                                0.97               0.97     0.97      0.97
XGBClassifier                      0.97               0.97     0.97      0.97
LGBMClassifier                     0.97               0.97     0.97      0.97
LinearDiscriminantAnalysis         0.97               0.97     0.97      0.97
RidgeClassifier                    0.97               0.97     0.97      0.97
RidgeClassifierCV                  0.97               0.97     0.97      0.97
KNeighborsClassifier               0.96               0.97     0.97      0.96
ExtraTreesClassifier               0.97               0.96     0.96      0.97
RandomForestClassifier             0.97               0.96     0.96      0.97
PassiveAggressiveClassifier        0.96               0.96     0.96      0.96
LogisticRegression                 0.96               0.96     0.96      0.96
```

This is Testing and Validation accuracy on Protein Data:



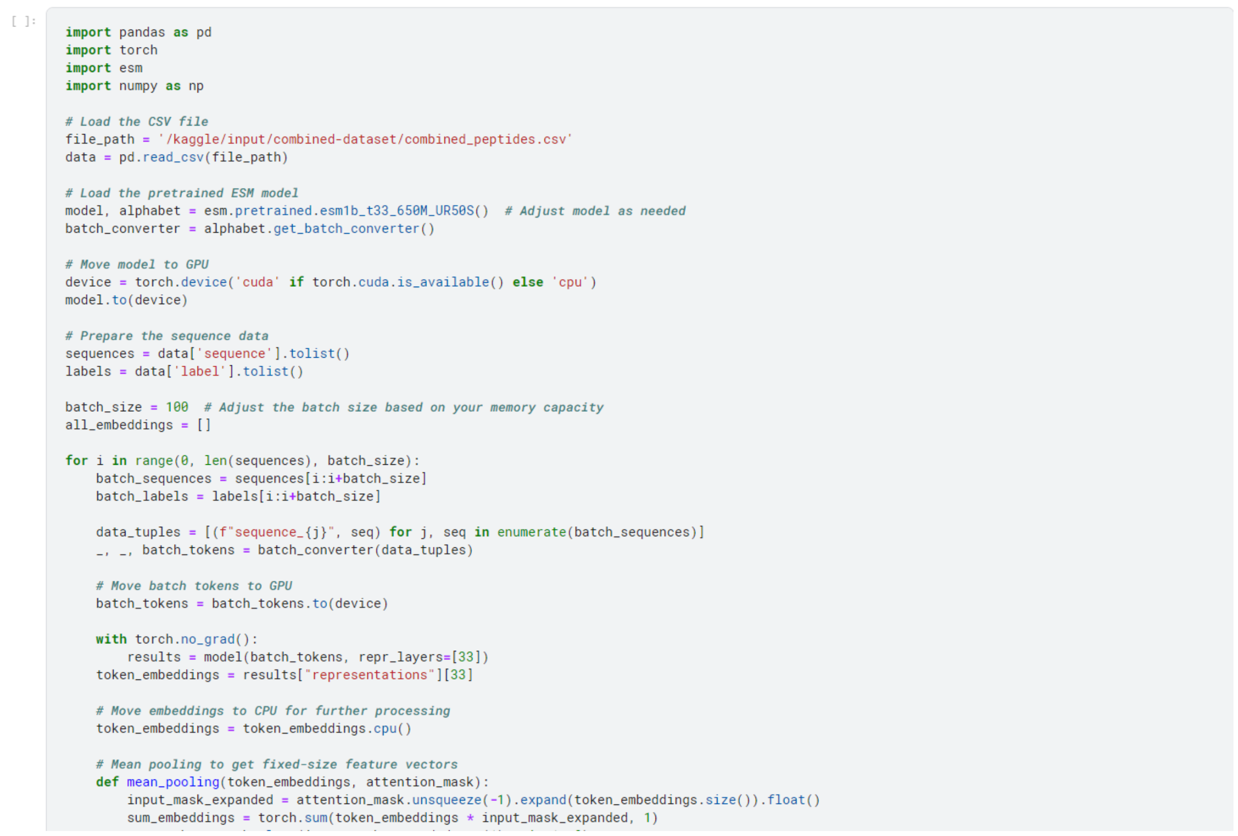
```
LGBMClassifier validation accuracy: 0.8906, test accuracy: 0.8862
XGBClassifier validation accuracy: 0.8861, test accuracy: 0.8933
RandomForestClassifier validation accuracy: 0.8826, test accuracy: 0.8773
AdaBoostClassifier validation accuracy: 0.8301, test accuracy: 0.8240
LogisticRegression validation accuracy: 0.8488, test accuracy: 0.8409
SVC validation accuracy: 0.8158, test accuracy: 0.8276
QuadraticDiscriminantAnalysis validation accuracy: 0.9057, test accuracy: 0.9004

Best Model Name: QuadraticDiscriminantAnalysis
```

Experiment – 3 (Evolutionary Scale Modeling)

1: peptide prediction

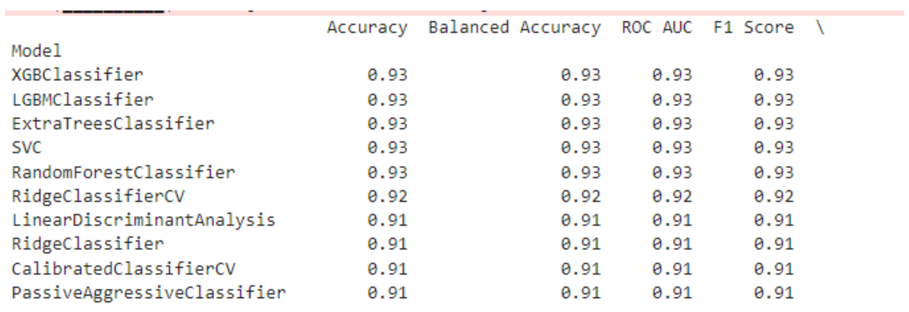
In experiment 3 , we have used Evolutionary scale modelling to extract feature of peptide. ESM is a technique based on protein language models, which are designed to understand and generate protein sequences by learning patterns and relationships between amino acids. These models, like those based on transformer architectures, are trained on large-scale protein sequence datasets. They can extract features that encapsulate the biological and chemical properties of the sequences.

The extracted feature embeddings are then saved in a CSV file. Subsequently, the data is split into training, testing, and validation sets in the following proportions: 60% for training, 20% for testing, and 20% for validation.



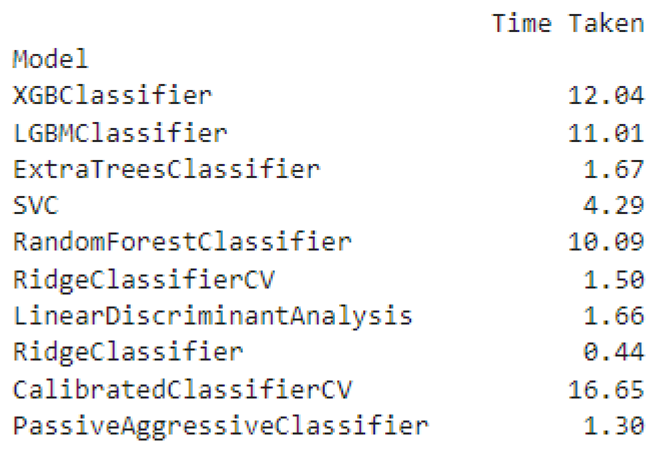
```
[ ]:
import pandas as pd
import torch
import esm
import numpy as np

# Load the CSV file
file_path = '/kaggle/input/combined-dataset/combined_peptides.csv'
data = pd.read_csv(file_path)

# Load the pretrained ESM model
model, alphabet = esm.pretrained.esm1b_t33_650M_UR50S()  # Adjust model as needed
batch_converter = alphabet.get_batch_converter()

# Move model to GPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

# Prepare the sequence data
sequences = data['sequence'].tolist()
labels = data['label'].tolist()

batch_size = 100  # Adjust the batch size based on your memory capacity
all_embeddings = []

for i in range(0, len(sequences), batch_size):
    batch_sequences = sequences[i:i+batch_size]
    batch_labels = labels[i:i+batch_size]

    data_tuples = [(f"sequence_{j}", seq) for j, seq in enumerate(batch_sequences)]
    _, _, batch_tokens = batch_converter(data_tuples)

    # Move batch tokens to GPU
    batch_tokens = batch_tokens.to(device)

    with torch.no_grad():
        results = model(batch_tokens, repr_layers=[33])
    token_embeddings = results["representations"][33]

    # Move embeddings to CPU for further processing
    token_embeddings = token_embeddings.cpu()

    # Mean pooling to get fixed-size feature vectors
    def mean_pooling(token_embeddings, attention_mask):
        input_mask_expanded = attention_mask.unsqueeze(-1).expand(token_embeddings.size()).float()
        sum_embeddings = torch.sum(token_embeddings * input_mask_expanded, 1)
```

This is the Result on LazyPredict:



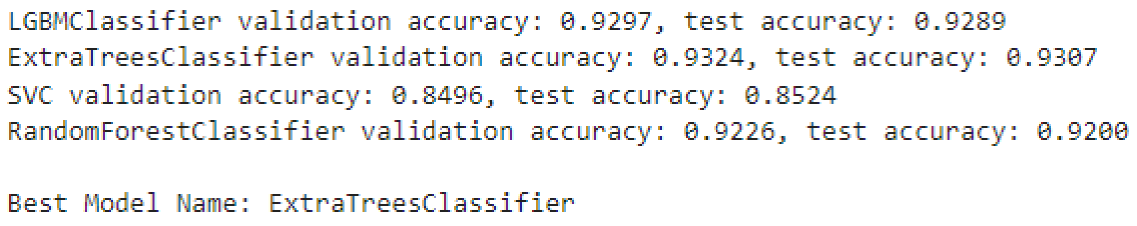
```
                              Accuracy  Balanced Accuracy  ROC AUC  F1 Score  \
Model
XGBClassifier                     0.93               0.93     0.93      0.93
LGBMClassifier                    0.93               0.93     0.93      0.93
ExtraTreesClassifier              0.93               0.93     0.93      0.93
SVC                               0.93               0.93     0.93      0.93
RandomForestClassifier            0.93               0.93     0.93      0.93
RidgeClassifierCV                 0.92               0.92     0.92      0.92
LinearDiscriminantAnalysis        0.91               0.91     0.91      0.91
RidgeClassifier                   0.91               0.91     0.91      0.91
CalibratedClassifierCV            0.91               0.91     0.91      0.91
PassiveAggressiveClassifier       0.91               0.91     0.91      0.91
```

This the time taken by lazyPredict Model:



```
                              Time Taken
Model
XGBClassifier                      12.04
LGBMClassifier                     11.01
ExtraTreesClassifier                1.67
SVC                                 4.29
RandomForestClassifier             10.09
RidgeClassifierCV                   1.50
LinearDiscriminantAnalysis          1.66
RidgeClassifier                     0.44
CalibratedClassifierCV             16.65
PassiveAggressiveClassifier         1.30
```

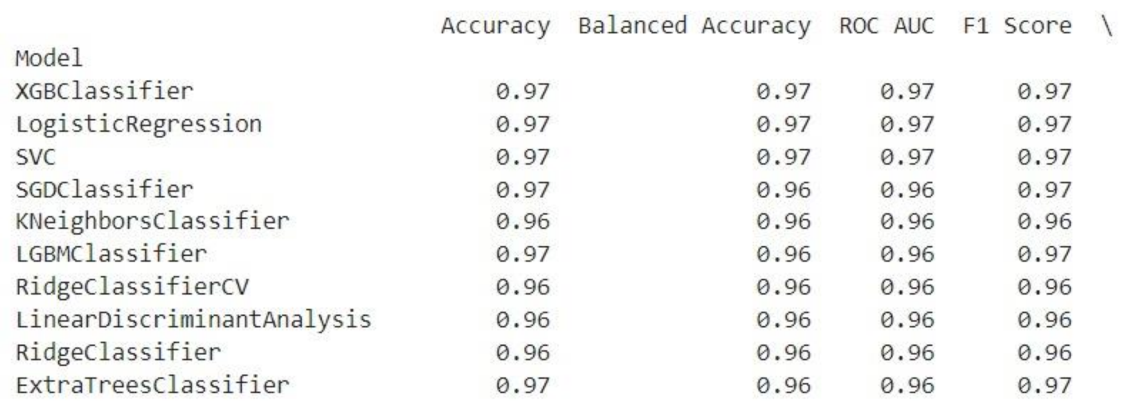
This is Testing and Validation accuracy on Peptide Data:



```
LGBMClassifier validation accuracy: 0.9297, test accuracy: 0.9289
ExtraTreesClassifier validation accuracy: 0.9324, test accuracy: 0.9307
SVC validation accuracy: 0.8496, test accuracy: 0.8524
RandomForestClassifier validation accuracy: 0.9226, test accuracy: 0.9200

Best Model Name: ExtraTreesClassifier
```

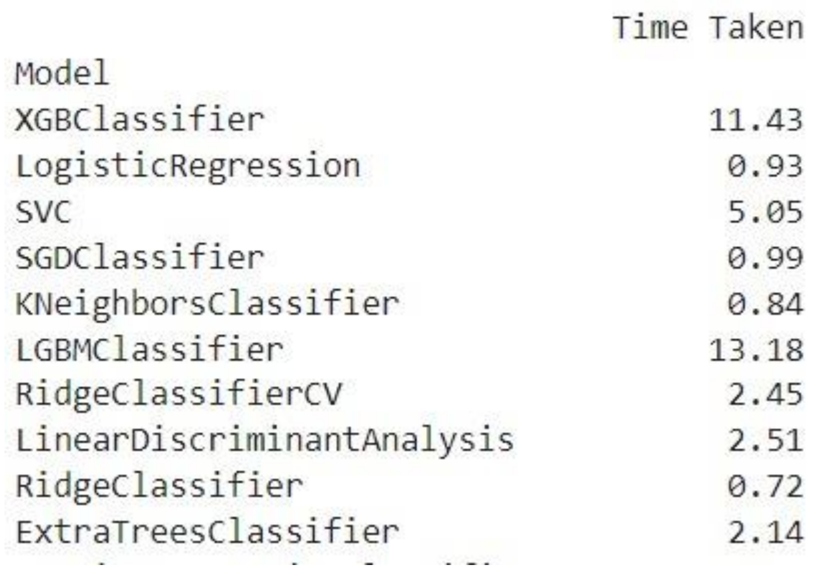
2: Protein

This is the result of top 10 models by using LazyPredict after using the same ESM Embedding:



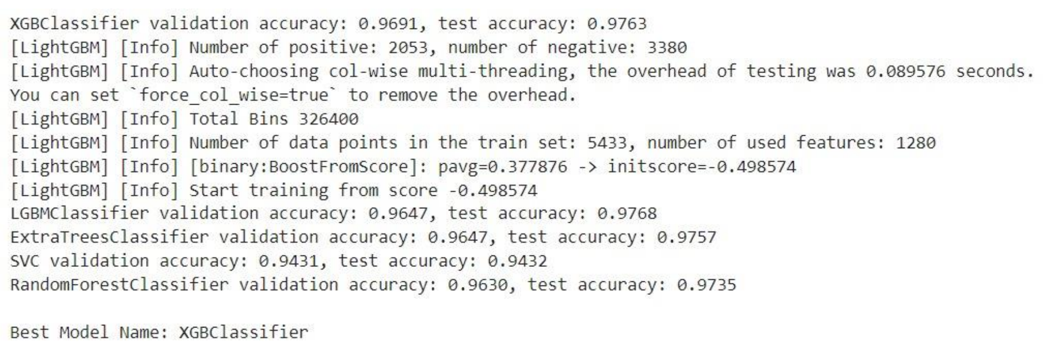
```
                              Accuracy  Balanced Accuracy  ROC AUC  F1 Score  \
Model
XGBClassifier                     0.97               0.97     0.97      0.97
LogisticRegression                0.97               0.97     0.97      0.97
SVC                               0.97               0.97     0.97      0.97
SGDClassifier                     0.97               0.96     0.96      0.97
KNeighborsClassifier              0.96               0.96     0.96      0.96
LGBMClassifier                    0.97               0.96     0.96      0.97
RidgeClassifierCV                 0.96               0.96     0.96      0.96
LinearDiscriminantAnalysis        0.96               0.96     0.96      0.96
RidgeClassifier                   0.96               0.96     0.96      0.96
ExtraTreesClassifier              0.97               0.96     0.96      0.97
```

Time taken by respective models:



```
                              Time Taken
Model
XGBClassifier                      11.43
LogisticRegression                  0.93
SVC                                 5.05
SGDClassifier                       0.99
KNeighborsClassifier                0.84
LGBMClassifier                     13.18
RidgeClassifierCV                   2.45
LinearDiscriminantAnalysis          2.51
RidgeClassifier                     0.72
ExtraTreesClassifier                2.14
```

Results on Testing and validation :



```
XGBClassifier validation accuracy: 0.9691, test accuracy: 0.9763
[LightGBM] [Info] Number of positive: 2053, number of negative: 3380
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.089576 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 326400
[LightGBM] [Info] Number of data points in the train set: 5433, number of used features: 1280
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.377876 -> initscore=-0.498574
[LightGBM] [Info] Start training from score -0.498574
LGBMClassifier validation accuracy: 0.9647, test accuracy: 0.9768
ExtraTreesClassifier validation accuracy: 0.9647, test accuracy: 0.9757
SVC validation accuracy: 0.9431, test accuracy: 0.9432
RandomForestClassifier validation accuracy: 0.9630, test accuracy: 0.9735

Best Model Name: XGBClassifier
```

References:

ToxIBTL: prediction of peptide toxicity based on information bottleneck and transfer learning