Run the script

Run the **AminoAcidPredictor.ipynb** with Jupyter Notebook.

Steps of ML model training

1. Import the necessary libraries.
2. Load the data from the 2 TSV files and store them in a single dataframe **a_domains_df**.
3. Data Preparation. The input set (**X**) stores the proteinogenic sequence and the output set (**y**) store the amino acid **three_letter_code** column that will bind to the sequence.
   The **CountVectorizer** is used to convert the input sequences (proteinogenic sequences) into a numerical representation that can be used by the machine learning model. The **analyzer='char'** parameter specifies that we want to treat each character in the sequence as a separate feature.
   We then use the **fit_transform ()** method of the vectorizer to transform the input sequences (**X_train, X_test**) into a sparse matrix (**X_train_vec, X_test_vec**). The sparse matrix represents the count of each character in the sequences.
   The **LabelEncoder** is used to encode the recruited amino acid labels (**y**) into numerical values. Each unique label is assigned a unique integer value.
   We then use the **fit_transform ()** method of the label encoder to transform the recruited amino acid labels (**y_train, y_test**) into encoded numerical values (**y_train_enc, y_test_vec**). This step is necessary because machine learning models typically require numerical inputs for the target variable.
   By the end of this step, the input features and the encoded target variable is prepared to be used for training the machine learning model.

4. Choose a model. ML models used in the project are:
   a. RandomForestClassifier (Decision Trees)
   b. SVC (Support Vector Machines)
   c. MLPClassifier (Multi-Layer Perceptron neural network with 100 neurons in the first hidden layer and 50 neurons in the second hidden layer.)
   d. XGBClassifier: The **XGBClassifier** is a gradient boosting model that uses an ensemble of decision trees for classification tasks. It combines the predictions of multiple weak models (decision trees) to make more accurate predictions.

5. Evaluate the model. The test data is passed through the model to get the output predictions. The predicted labels are decoded using the inverse transformation from the label encoder. Accuracy is calculated by comparing the predicted labels with the true labels, and the classification report is printed to provide a more detailed evaluation.

Neural network using PyTorch:

a. The **AminoAcidClassifier** class defines a neural network model. It consists of an embedding layer (**nn.Embedding**) that maps each input value to a dense vector representation, followed by a fully connected linear layer (**nn.Linear**) that produces the final output. The **forward ()** method specifies how the input flows through the layers to generate the output.
b. The vectorized input sequences and encoded target variable are converted to PyTorch tensors using the **torch.tensor()** function.
c. The loss function is defined as **nn.CrossEntropyLoss()**, which is suitable for multi-class classification problems. The optimizer is set to **optim.Adam()**, which implements the Adam optimization algorithm. The model parameters are passed to the optimizer using **model.parameters()**.
d. Train the model. For each batch, the optimizer is zeroed (**optimizer.zero_grad()**), the input and target tensors for the batch are extracted, the model is called with the batch input to get the output, the loss is calculated using the criterion, and then backpropagation is performed (**loss.backward()**) followed by an optimizer step (**optimizer.step()**) to update the model's parameters based on the computed gradients.
e. Evaluate the model. The test data is passed through the model to get the output predictions. The predicted labels are decoded using the inverse transformation from the label encoder. Accuracy is calculated by comparing the predicted labels with the true labels, and the classification report is printed to provide a more detailed evaluation.

Results of all models:

ML model	Accuracy
RandomForestClassifier	66%
SVC	55.5%
MLPClassifier	61.8%
XGBClassifier	63.2%
PyTorch Neural Network	22.2%