

ReadMe

ASSIGNMENT 1: Submission by Group_72

AIM : Classify protein sequence into positive and negative based on the label provided in training data.

DATA PROVIDED : Train, Test and Sample Data

Approach: The protein sequence consists of text-based data, which has to be processed or converted into numerical value before applying any model. Further, it would be trained on training data, and predictions would be checked afterwards.

■ IMPORTING THE REQUIRED LIBRARIES

Import pandas as pd: This imports the Pandas library, commonly used for data manipulation and analysis. The alias "pd" is often used for brevity when referring to Pandas functions and objects.

from sklearn.preprocessing import LabelEncoder: This imports the LabelEncoder class from the scikit-learn (sklearn) library. It's used for encoding categorical variables into numerical values.

import numpy as np: This imports the NumPy library and assigns it the alias "np." NumPy is essential for numerical operations in Python, especially when working with arrays and matrices.

from sklearn.model_selection import train_test_split: This imports the train_test_split function from scikit-learn, which splits a dataset into training and testing subsets.

from sklearn.ensemble import RandomForestClassifier: This imports the RandomForestClassifier class from scikit-learn. Random Forest is a popular ensemble machine-learning algorithm used for classification tasks.

from sklearn.metrics import accuracy_score: This imports the accuracy_score function from scikit-learn. It's commonly used to calculate the accuracy of a machine learning model's predictions.

from imblearn.over_sampling import SMOTE: This imports the Synthetic Minority Over-sampling Technique (SMOTE) from the imbalanced-learn library. SMOTE is used to handle class imbalance by generating synthetic samples of the minority class.

import collections: This imports Python's built-in "collections" module, which provides specialised container datatypes.

import csv: This imports Python's built-in "csv" module for working with CSV (Comma Separated Values) files.

from sklearn.svm import SVC: This imports the Support Vector Classifier (SVC) class from scikit-learn. SVC is a machine learning algorithm for classification tasks based on support vector machines.

■ ENCODING AMINO ACIDS

We begin by declaring a custom encoder for amino acids using a LabelEncoder and an AA_sequence_loop dictionary. This encoder maps each amino acid character to a numerical index.

We load training data from a CSV file named "train.csv" into a pandas DataFrame called train_data.

We define a function encode_sequence(sequence) that takes an amino acid sequence as input and encodes it using the custom encoder, producing a list of numerical representations.

We apply the custom encoding to the 'amino_acid_sequence' column of Wer DataFrame using the encode_sequence function, creating a new column called 'encoded_sequence' to store the encoded representations.

We initialise a TF-IDF vectoriser using scikit-learn's TfidfVectorizer class. The vectoriser is configured to treat individual characters (amino acids) as features and consider n-grams of length 1 to 3 for TF-IDF encoding.

We fit and transform the encoded sequences from the 'encoded_sequence' column using the TF-IDF vectoriser, resulting in a tfidf_matrix that contains the TF-IDF-encoded features for Wer amino acid sequences.

■ LOOPING TRAINING DATASET

It starts by iterating through each amino acid sequence in the 'Sequence' column of the train_data DataFrame using a for loop.

Calculating Amino Acid Sequence Characteristics:

For each sequence in the loop, it calculates two things:

AA_characters: The sequence's total number of characters (amino acids).

AA_occurrence: A dictionary that counts the occurrences of each unique amino acid in the sequence using the collections.Counter() function.

■ CREAING TF-IDF-LIKE SCORES

After calculating the amino acid characteristics, it creates a dictionary named `tf_idf_scores` that represents TF-IDF-like scores for the amino acids in the sequence. However, in this code, the scores are based on the sequence's raw counts of amino acids, not the traditional TF-IDF formula.

Appending Score Dictionary to List:

Finally, it appends the `tf_idf_scores` dictionary (representing the amino acid counts) to the `tfidf_list`. This list will contain one dictionary for each sequence in the training data, with each dictionary holding the counts of amino acids.

Finalising the Encoding Sequence Based on TF-IDF-like Scores:

This code snippet is designed to finalise the encoding of amino acid sequences based on the TF-IDF-like scores calculated earlier.

Initialisation of `final_list`: It starts by creating an empty list called `final_list`, which will hold the final encoded representations of the amino acid sequences.

Looping Through TF-IDF-like Score Dictionaries:

The code enters a for loop to iterate through the dictionaries in `tfidf_list`. Each dictionary represents the TF-IDF-like scores (or, in this case, raw counts) of amino acids for a specific sequence.

Creating a New List with 21 Zeroes:

Inside the loop, it initialises a new list called `temp_list` containing 21 zeroes. This list is used to represent the encoding of the current sequence.

Populating `temp_list` with Amino Acid Counts:

The code then enters a nested loop, iterating through the items (amino acid counts) in the current TF-IDF-like score dictionary (IDF).

Each amino acid in the dictionary determines its corresponding index in the `temp_list` based on the `AA_sequence_loop` dictionary and assigns the amino acid count to that index in `temp_list`.

Appending `temp_list` to `final_list`:

After populating `temp_list` with amino acid counts, it appends `temp_list` to the `final_list`. This results in the final list containing a list of lists, where each inner list represents the encoding of one amino acid sequence.

Converting Sequences and Labels to Numpy Arrays:

Sequences are created as a NumPy array and contain the final encoded representations of amino acid sequences from final_list. Labels are created as a NumPy array and contain the labels (or target values) associated with each sequence, which are extracted from the 'Label' column of the train_data DataFrame.

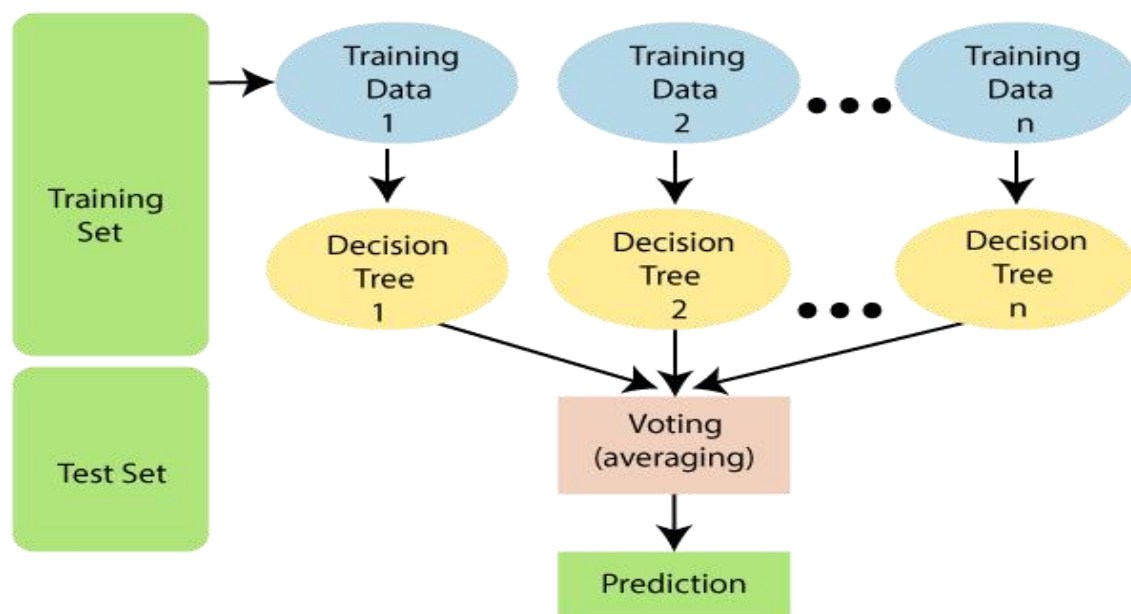
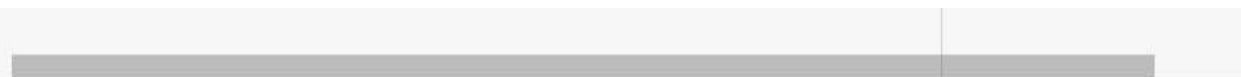
■ APPLYING SMOTE (Synthetic Minority Over-sampling Technique):

The SMOTE (Synthetic Minority Over-sampling Technique) resampling technique with a specified random seed. SMOTE is used for dealing with class imbalance in machine learning datasets. It generates synthetic examples of the minority class to balance the class distribution.

The X_sample and y_resample variables store the resampled data. X_sample contains the resampled sequences (features), and y_resample contains the corresponding resampled labels.

■ WORKFLOW FOR TRAINING AND EVALUATING ML MODEL

- The data is split into training and testing sets.
- A random forest classifier is initialised and trained on the training data.
- The trained model is used to make predictions on the testing data.
- The accuracy of the model's predictions is calculated and displayed as a percentage, assessing the model's performance on the test data.



■ PROCEEDING SIMILARLY WITH TEST DATASET

Reading the Original Test Dataset and applying the TF-IDF on Test Data.

Reads the test dataset, processes the amino acid sequences, and calculates TF-IDF-like scores (though it's essentially counting amino acid occurrences) for each sequence in the test dataset. The results are stored in the `test_tfidf_list`, which can be used for further analysis or predictions.

Converts TF-IDF-like scores into encoded test sequences based on a predefined mapping.

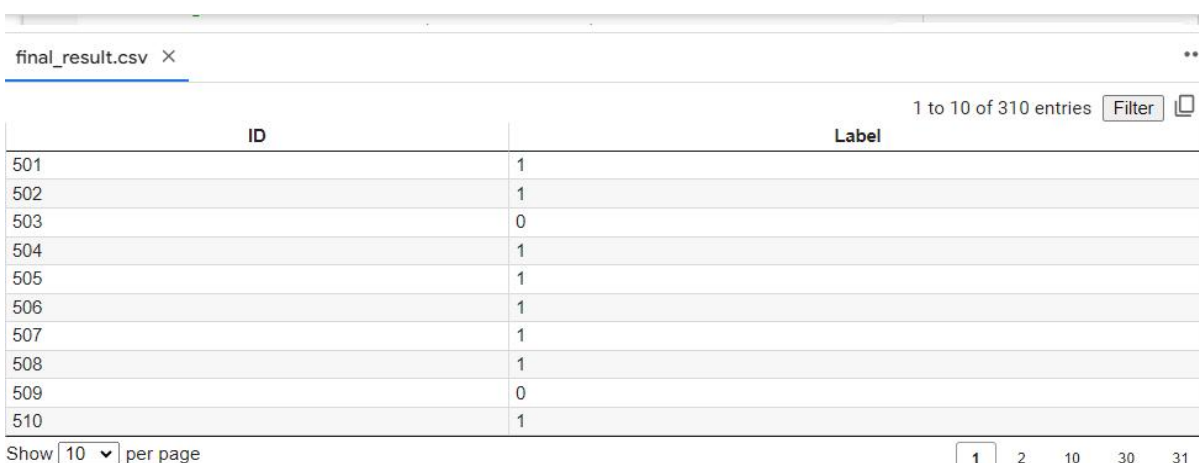
A TF-IDF-like score dictionary from `test_tfidf_list` translates the counts of amino acids into a fixed-length encoding (represented by `temp_list`) and appends this encoding to `test_final_list`. The `test_final_list` will contain a set of encoded representations for each amino acid sequence in the test data, similar to what was done previously for the training data.

■ MODEL PREDICTION

Now, at last, a trained **Random Forest classifier predicts labels for the test data and then saves the results to a CSV file.**

Using the Trained Random Forest Classifier for Predictions and Saving the Results to a CSV File.

The trained Random Forest classifier makes predictions on the test data and then stores these predictions in a CSV file for further analysis or submission. The resulting CSV file will have 'ID' and 'Label' columns.



ID	Label
501	1
502	1
503	0
504	1
505	1
506	1
507	1
508	1
509	0
510	1

This is the final_result.csv generated.