Project title:

Mice Protein Expression

By INeuron

Team Members:

* Abhishek M Ganjigatti
* Manoj Kendri

Date: 19-01-2024

**Project link GitHub:** [**https://github.com/abhishek-ganjigatti/Mice-protein-expression-ml-project**](https://github.com/abhishek-ganjigatti/Mice-protein-expression-ml-project)

Introduction

The Low-Level Documentation (LLD) for the Mice Protein Expression Classification project serves as a comprehensive guide for developers, collaborators, and stakeholders involved in the project. This document delves into the intricacies of the project's implementation, providing detailed insights into the code structure, data processing steps, model training, evaluation metrics, and deployment strategies.

Purpose of the Document:

The primary purpose of this document is to offer a detailed breakdown of the low-level aspects of the Mice Protein Expression Classification project. It aims to:

1. Provide a deep understanding of the code implementation for developers, ensuring clarity in coding practices and structure.
2. Facilitate collaboration among team members by offering a standardized reference for coding conventions, modular functions, and dependencies.
3. Serve as a guide for future developers or maintainers, allowing them to comprehend the codebase efficiently.

Scope and Objectives:

The scope of this document encompasses various facets of the project, including but not limited to:

1. Code Structure and Modularity:

* Understanding the organization of the code, modular functions, and the main script that orchestrates the entire workflow.

1. Data Processing Steps:

* Detailed insights into how the dataset is loaded, relevant features are extracted, missing values are handled, and categorical variables are encoded.

1. Model Training and Hyperparameter Tuning:

* Explanation of how the Support Vector Machine (SVM) model is initialized, how the dataset is split for training and testing, and the hyperparameter tuning process.

1. Evaluation Metrics:

* Discussion on the metrics used to evaluate the model's performance, including accuracy, precision, recall, and F1 score.

1. Code Documentation and Logging:

* Overview of how the codebase is documented, adherence to PEP 8 standards, and the implementation of the Python logging library for action tracking.

1. Database Integration:

* Details on how the project interacts with the Cassandra database, including data retrieval and storage processes.

1. Cloud Deployment and API/User Interface:

* Information on the deployment strategies, cloud platform selection, and the provision of an API or user interface for testing and interaction.

1. Ops Pipeline and Continuous Integration:

* Discussion on the integration tools used, such as Jenkins or Circle CI, and the implementation of AI Ops pipeline components like MLflow or DVC.

1. System Architecture and Latency Measurement:

* Insights into the high-level design overview, components interaction, and strategies for measuring model response time.

1. Optimization Techniques and Testing:

* Explanation of optimization strategies at both code and architecture levels, along with the testing procedures and test case examples.

1. Conclusion and Future Recommendations:

A summarized conclusion of the low-level aspects, key findings, and recommendations for future enhancements.

By addressing these aspects, the document aims to provide a comprehensive reference for anyone involved in the development, maintenance, or understanding of the Mice Protein Expression

Project Components:

* **Data Loading**

Load the dataset from the provided Cassandra database link to initiate data processing.

# Load the dataset

df = pd.read\_csv("Mice Protein Expression data.csv")

### Feature Extraction and Imputation:

### Extract relevant protein expression columns and handle missing values through imputation.

# Extract the relevant columns

t\_colname\_I = df[['DYRK1A\_N', 'ITSN1\_N', 'BDNF\_N', ...]].values

# Impute missing values with median

imputer = SimpleImputer(missing\_values=np.NaN, strategy='median')

t\_colname\_I = imputer.fit\_transform(t\_colname\_I)

### Label Encoding:

### Convert categorical variables in target columns into numerical format for model compatibility.

# Extract relevant columns for target variable encoding

t\_colname\_D = df[['Genotype', 'Treatment', 'Behavior', 'class']].values

# Label encode the categorical variables

LE\_t\_colname\_D = LabelEncoder()

for i in range(4):

t\_colname\_D[:, i] = LE\_t\_colname\_D.fit\_transform(t\_colname\_D[:, i])

### Model Training:

### Train a Support Vector Machine (SVM) model with a linear kernel using the processed data.

# Model training using Support Vector Machine (SVM) algorithm

model = SVC(kernel='linear')

model.fit(I\_train, D2\_train)

### Evaluation Metrics:

Assess the model's performance using accuracy and additional metrics (precision, recall, F1 score).

# To predict the output "Behavior"

df\_fopt1 = model.predict(I\_test)

# Evaluate the model

accc = model.score(I\_test, D2\_test)

print(accc)

### Logging and Documentation:

### Implement logging for tracking actions and adhere to documentation standards for maintainability.

# Logging is a must for every action performed by your code

import logging

# Configure logging

### logging.basicConfig(filename='project\_log.log', level=logging.INFO)

These components collectively form the foundation of the Mice Protein Expression Classification project, encompassing data loading, preprocessing, model training, and evaluation.

**Code Structure and Modularity:**

* Main Script

Orchestrate the workflow by calling modular functions and managing the overall execution of the project.

# Main script structure

def main():

# Call modular functions

load\_data()

preprocess\_data()

train\_model()

evaluate\_model()

# Execute the main script

if \_\_name\_\_ == "\_\_main\_\_":

main()

* Modular Functions

Organize functionality into modular units for better readability, reusability, and maintainability.

def load\_data():

# Code for loading the dataset

df = pd.read\_csv("Mice Protein Expression data.csv")

logging.info("Dataset loaded successfully.")

def preprocess\_data():

# Code for feature extraction, imputation, and label encoding

# ...

def train\_model():

# Code for training the Support Vector Machine (SVM) model

# ...

def evaluate\_model():

# Code for evaluating the model and printing performance metrics

# ...

### Libraries and Dependencies:

Specify external libraries and dependencies required for the project.

# Libraries and dependencies

import numpy as np

import pandas as pd

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

import logging

* **Logging and Documentation**

Implement logging for tracking actions and adhere to documentation standards for maintainability.

# Configure logging

logging.basicConfig(filename='project\_log.log', level=logging.INFO)

By structuring the code in this modular fashion, each function handles a specific aspect of the workflow, making the codebase more readable, maintainable, and facilitating collaboration among team members. The main script serves as the entry point, orchestrating the flow through these modular functions. External libraries and dependencies are explicitly stated for transparency and ease of understanding.

**Data Processing Steps:**

**1. Loading the Dataset:**

Load the dataset from the provided Cassandra database link.

# Load the dataset

df = pd.read\_csv("Mice Protein Expression data.csv")

**2. Feature Selection:**

Select relevant protein expression columns for further analysis.

# Extract the relevant columns

t\_colname\_I = df[['DYRK1A\_N', 'ITSN1\_N', 'BDNF\_N', ...]].values

**3. Handling Missing Values:**

Impute missing values in protein expression data using the median.

# Impute missing values with median

imputer = SimpleImputer(missing\_values=np.NaN, strategy='median')

t\_colname\_I = imputer.fit\_transform(t\_colname\_I)

**4. Label Encoding:**

Convert categorical variables in target columns to numerical format for model compatibility.

# Extract relevant columns for target variable encoding

t\_colname\_D = df[['Genotype', 'Treatment', 'Behavior', 'class']].values

# Label encode the categorical variables

LE\_t\_colname\_D = LabelEncoder()

for i in range(4):

t\_colname\_D[:, i] = LE\_t\_colname\_D.fit\_transform(t\_colname\_D[:, i])

**Model Training and Evaluation**

**1. SVM Model Initialization**:

Initialize the Support Vector Machine (SVM) model with a linear kernel.

# Model training using Support Vector Machine (SVM) algorithm

model = SVC(kernel='linear')

**2. Splitting the Dataset:**

Split the dataset into training and testing sets for model evaluation.

# Split the dataset for dependent variable "Behavior"

I\_train, I\_test, D2\_train, D2\_test = train\_test\_split(df\_1\_I, df\_3\_D, test\_size=0.2, random\_state=0)

**3. Accuracy Calculation**

Calculate the accuracy of the trained model on the test set.

# Evaluate the model

accc = model.score(I\_test, D2\_test)

print(accc)

Testing and Test Cases

* Loading the Data model :

I have used

import pickle

filename='trained\_model3.sav'

pickle.dump(model,open(filename,'wb'))

loaded\_model=pickle.load(open('trained\_model3.sav','rb'))

input\_data=(0.447506385,  0.62817583  ,0.36738809 ,2.38593897,  4.807635435,  0.218577766,  0.176233365 ,2.14128243 ,0.195187525  ,1.442398172  ,0.566339562  ,0.289823901  ,0.363892996  ,0.26683694,  0.85912085, 0.521306627 ,1.538244388  ,1.968275306, 0.495899987 ,0.672402205, 0.36940449  ,0.357171663, 0.179728458,  1.227449926 ,2.956983466  ,1.447909665  ,0.250840167, 0.284043554,  0.704395752,  0.156875924 ,0.391047184  ,2.467132679  ,0.327597795, 0.404489851,  0.296276381 ,0.674418605  ,0.539723081, 0.354214276 ,0.51431644 ,0.347224089  ,0.303132141, 0.412824304 ,0.382578304  ,0.162330317, 0.77969457, 0.186792986 ,1.634615385, 0.28803733, 0.332367081 ,1.12344457 ,0.175692873  ,0.150593891  ,0.183823529, 0.106476244 ,0.13956448,  0.174844457 ,0.130514706  ,0.115243213  ,0.236849548  ,0.13645362 ,0.478577489  ,0.244485294  ,1.507777149, 2.003535068 ,0.120687217, 0.920178167 ,0.843679299, 0.190469457,  0.131575226 , 0.106476244 ,0.109445701, 0.439833145 ,0.11665724,  0.140766403 ,0.14218043,  1.816388575,  0,0,0,0)

input\_data\_as\_numpy\_array=np.asarray(input\_data)

input\_data\_reshaped=input\_data\_as\_numpy\_array.reshape(1,-1)

prediction =model.predict(input\_data\_reshaped)

print(prediction)

Result: ['c-CS-m']

**Documentation Links**

**High-Level Document:**

<https://github.com/abhishek-ganjigatti/Mice-protein-expression-ml-project/blob/decb989fd1cf984c5972f976dab6ef19a14e9e25/High%20Level%20Documentation.docx>

**Project code:**

[**https://github.com/abhishek-ganjigatti/Mice-protein-expression-ml-project/blob/decb989fd1cf984c5972f976dab6ef19a14e9e25/CODE\_mice\_protein\_expression.md**](https://github.com/abhishek-ganjigatti/Mice-protein-expression-ml-project/blob/decb989fd1cf984c5972f976dab6ef19a14e9e25/CODE_mice_protein_expression.md)

**Conclusion:**

The Mice Protein Expression Classification project successfully addresses the challenge of identifying discriminant protein subsets between control and Down syndrome mice exposed to context fear conditioning. Leveraging machine learning techniques, specifically Support Vector Machine (SVM), the project achieved commendable results in protein expression classification.

**Summary of Key Points:**

* **Objective Achievement:**

Successfully classified protein expression levels in mice.

Balanced trade-off between accuracy and efficiency in analysis validation.

* **Dataset Characteristics:**

77 proteins measured in the cerebral cortex of control and Down syndrome mice.

Dataset comprises 72 mice, with 38 control and 34 trisomic mice.

* **Data Processing:**

Robust data loading, feature selection, and imputation strategies.

Effective label encoding for categorical variables.

* **Model Training:**

Utilized Support Vector Machine (SVM) with a linear kernel.

Achieved satisfactory accuracy in predicting mouse behavior.

* **Documentation and Logging:**

Adhered to PEP 8 coding standards.

Implemented Python logging for tracking actions.

**References:**

* Siddhardhan (YouTube channel) -

<https://youtu.be/WLwjvWq0GWA?si=cDnYtO46e0cR-xtt>