NATIONAL UNIVERSITY OF SINGAPORE
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

ACADEMIC YEAR 2025-2026
SEMESTER 1

EE2213: Introduction to AI

PROJECT – Multi-Class Classification: L7-L9

Project (20% of total grade)
Submission Deadline: 27 October 2025 (Monday), 23:59

Project Overview:

You will implement a machine learning task for multi-class classification using the **Vehicle** dataset from OpenML (https://www.openml.org/search?type=data&status=active&id=54). This project involves building a classification system from scratch to predict vehicle types based on their features. You must work with the provided project_template.ipynb file to import packages and load the dataset.

Dataset Information:

- 846 samples, 18 features
- Target labels (4 classes): ['bus' 'opel' 'saab' 'van']
- Need to convert to class [0 1 2 3]
- Number of samples for each class: [218 212 217 199]

The project consists of four main components implemented sequentially.

PART 1: Dataset Partition and One-hot Encoding (2%)

Requirements:

1. Split the dataset generated from PART 0 into 3 sets: 60% of samples for training, 20% of samples for validation, and 20% of samples for testing. Please use “from sklearn.model_selection import train_test_split” with “random_state=xxx”, **where “xxx” is the last 3 digits of your student matriculation number**, e.g., xxx=567 for A1234567B.
2. Generate target output using one-hot encoding for training set, validation set and test set.

Instructions:

Please use the corresponding cell template provided to you. Do not comment out or delete any lines. It should contain a function “dataset_partition_encoding” that takes the following inputs and returns the following outputs in the following order:

Python function inputs:

- X: numpy feature matrix with dimensions (number_of_samples × number_of_features)

- y: numpy target output array of length number_of_samples.

Python function outputs:

- X_train: training numpy feature matrix with dimensions (number_of_training_samples × number_of_features)
- X_val: validation numpy feature matrix with dimensions (number_of_validation_samples × number_of_features)
- X_test: test numpy feature matrix with dimensions (number_of_test_samples × number_of_features)
- Ytr_onehot: one-hot encoded training target numpy matrix with dimension (number_of_training_samples × number_of_classes)
- Yval_onehot: one-hot encoded validation target numpy matrix with dimension (number_of_validation_samples × number_of_classes)
- Yts_onehot: one-hot encoded test target numpy matrix with dimension (number_of_test_samples × number_of_classes)

PART 2: Feature Selection using Pearson Correlation (4%)

Implement feature selection to remove redundant features for classification.

Requirements:

1. The first feature ("COMPACTNESS") is selected by default. Each new feature is added only if it is not highly correlated (i.e., the absolute Pearson correlation coefficient |r| <= 0.8) with all already selected features.
2. Use **training data only** for the feature selection
3. Apply the selected feature indices to the training, validation and test sets.

Instructions:

Please use the corresponding cell template provided to you. Do not comment out or delete any lines. It should contain a function "feature_selection" that takes the following inputs and returns the following outputs in the following order:

Python function inputs:

- X_train: training numpy feature matrix with dimensions (number_of_training_samples × number_of_features)
- X_val: validation numpy feature matrix with dimensions (number_of_validation_samples × number_of_features)
- X_test: test numpy feature matrix with dimensions (number_of_test_samples × number_of_features)

- feature_names: a list of feature names, where each name is a string. The length should be 18.
- threshold: the threshold value for absolute Pearson correlation coefficient. It is fixed to 0.8.

Python function outputs:

- selected_features: a list of selected feature names, where each name is a string.
- FS_X_train: training numpy feature matrix after applying feature selection. The dimension should be (number_of_training_samples × number_of_selected_features)
- FS_X_val: validation numpy feature matrix after applying feature selection. The dimension should be (number_of_validation_samples × number_of_selected_features)
- FS_X_test: test numpy feature matrix after applying feature selection. The dimension should be (number_of_test_samples × number_of_selected_features)

PART 3: Polynomial Regression for Classification (6%)

Implement polynomial feature transformation and classification.

Requirements:

1. Using the training, validation, and test sets generated from PART 2, perform a polynomial regression (utilizing "from sklearn.preprocessing import PolynomialFeatures") from orders 1 to 3 for classification. If $P^T P$ is non-invertible, adopt L2 regularization with regularization factor $\lambda = 0.001$.
2. For each polynomial order, compute training accuracy and validation accuracy.
3. Select the best polynomial order based on validation accuracy and compute the test accuracy for the best polynomial order.

Instructions:

Please use the corresponding cell template provided to you. Do not comment out or delete any lines. It should contain a function "polynomial_for_classification" that takes the following inputs and returns the following outputs in the following order:

Python function inputs:

- FS_X_train: training numpy feature matrix after applying feature selection. The dimension should be (number_of_training_samples × number_of_selected_features)
- FS_X_val: validation numpy feature matrix after applying feature selection. The dimension should be (number_of_validation_samples × number_of_selected_features)
- FS_X_test: test numpy feature matrix after applying feature selection. The dimension should be (number_of_test_samples × number_of_selected_features)
- Ytr_onehot: one-hot encoded training target numpy matrix with dimension

(number_of_training_samples × number_of_classes)

- Yval_onehot: one-hot encoded validation target numpy matrix with dimension (number_of_validation_samples × number_of_classes)

- Yts_onehot: one-hot encoded test target numpy matrix with dimension (number_of_test_samples × number_of_classes)

- max_order: an interger number specifies the maximum polynomial order to consider. It is fixed to 3.

- lamda: a float number specifies regularization strength λ. It is fixed to 0.001.

Python function outputs:

- acc_train_list: a list of training accuracies for different polynomial orders. The length should be 3.

- acc_val_list: a list of validation accuracies for different polynomial orders. The length should be 3.

- best_order: an interger number specifies the best polynomial order based on the validation accuracy.

- acc_test: a float number specifies the test accuracy for the best polynomial order.

PART 4: Multinomial Logistic Regression (8%)

Implement multinomial logistic regression using gradient descent with categorical cross-entropy loss $L_{CCE}(\mathbf{f}_{\mathbf{W}}(\mathbf{x}_i), \mathbf{y}_i)$.

Requirements:

1. Using the training, validation, and test sets generated from PART 2, perform multinomial logistic regression using gradient descent (the offset/bias term should be added) for 4 different learning rates: [0.0001, 0.001, 0.01, 0.1]. Use "np.random.seed(xxx)" and "np.random.normal(0, 0.1, (n_features, num_classes))" for weight matrix initialization, **where "xxx" should be the last 3 digits of your student matriculation number**. The number of gradient descent iterations for each learning rate is fixed to 20000. For each learning rate, compute final training accuracy and final validation accuracy. Then, select the best learning rate based on validation accuracy and compute the test accuracy for the best learning rate. Plot the mean categorical cross-entropy loss $J(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N} L_{CCE}(\mathbf{f}_{\mathbf{W}}(\mathbf{x}_i), \mathbf{y}_i)$ versus number of iterations for each learning rate on the same axes (i.e., 4 separate curves in one plot) . Use different colors for each learning rate.

2. Using the training, validation, and test sets generated from PART 2, perform z-score standardization (utilizing "from sklearn.preprocessing import StandardScaler"). Note that the mean and standard deviation should only be calculated based on training data. The same mean and standard deviation should be applied to the normalization of validation and test sets. Then, add offset/bias term after normalization and perform multinomial logistic

regression using gradient descent for the same 4 different learning rates: [0.0001, 0.001, 0.01, 0.1]. Use “np.random.seed(xxx)” and “np.random.normal(0, 0.1, (n_features, num_classes))” for weight matrix initialization as well, **where “xxx” should be the last 3 digits of your student matriculation number**. The number of gradient descent iterations for each learning rate is also fixed to 20000. For each learning rate, compute final training accuracy and final validation accuracy. Then, select the best learning rate based on validation accuracy and compute the test accuracy for the best learning rate. Plot the mean categorical cross-entropy loss $J(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N} L_{CCE}(\mathbf{f}_{\mathbf{W}}(\mathbf{x}_i), \mathbf{y}_i)$ versus number of iterations for each learning rate on the same axes (i.e., 4 separate curves in one plot). Use different colors for each learning rate.

3. Compare the two figures. Analyze the effect of normalization based on the two figures.

Note: When computing softmax output, compute $\sigma(\mathbf{z})_k = \frac{e^{z_k - \max(\mathbf{z})}}{\sum_{j=1}^{K} e^{z_j - \max(\mathbf{z})}}$ instead to prevent overflow. Then, clip the output utilizing “np.clip(output, eps, 1-eps)”, where “eps” is set to 1e-15. This is to prevent extreme probabilities of exactly 0 or 1, which would cause issues during cross-entropy computation.

Instructions:

Please use the corresponding cell template provided to you. Do not comment out or delete any lines. It should contain two functions “MLR_select_lr” and “cost_vs_iter_curve”. “MLR_select_lr” aims to perform multinomial logistic regression and the select the best learning rate for both input without normalization and input with z-score standardization. “cost_vs_iter_curve” aims to plot two figures showing cost values versus number of iterations for the 4 learning rates (one figure for input without normalization, the other figure for input with z-score standardization). The inputs and outputs of the two functions are shown as follows:

Python function “MLR_select_lr” inputs:

- FS_X_train: training numpy feature matrix after applying feature selection. The dimension should be (number_of_training_samples × number_of_selected_features)
- FS_X_val: validation numpy feature matrix after applying feature selection. The dimension should be (number_of_validation_samples × number_of_selected_features)
- FS_X_test: test numpy feature matrix after applying feature selection. The dimension should be (number_of_test_samples × number_of_selected_features)
- Ytr_onehot: one-hot encoded training target numpy matrix with dimension (number_of_training_samples × number_of_classes)
- Yval_onehot: one-hot encoded validation target numpy matrix with dimension (number_of_validation_samples × number_of_classes)
- Yts_onehot: one-hot encoded test target numpy matrix with dimension (number_of_test_samples × number_of_classes)
- lr_list: a list of learning rates to test. It is fixed to [0.0001, 0.001, 0.01, 0.1].
- num_iters: an integer number specifies number of iterations for gradient descent. It is fixed

to 20000.

Python function "MLR_select_lr" outputs:

- cost_dict: a dictionary of cost values for each learning rate for input without normalization. Each key is the learning rate. The corresponding value should be a numpy array of cost values with a length of (number of iterations + 1) for the corresponding learning rate. Initial cost before performing gradient descent should be stored in the first element of the numpy array.
- acc_train_list_Log: a list of training accuracies for different learning rates for input without normalization. The length should be 4.
- acc_val_list_Log: a list of validation accuracies for different learning rates for input without normalization. The length should be 4.
- best_lr: a float number specifies the best learning rate based on the validation accuracy for input without normalization.
- test_acc_Log: a float number specifies the test accuracy for the best learning rate for input without normalization.
- cost_dict_norm: a dictionary of cost values for each learning rate for input with z-score standardization. Each key is the learning rate. The corresponding value should be a numpy array of cost values with a length of (number of iterations + 1) for the corresponding learning rate. Initial cost before performing gradient descent should be stored in the first element of the numpy array.
- acc_train_list_Log_norm: a list of training accuracies for different learning rates for input with z-score standardization. The length should be 4.
- acc_val_list_Log_norm: a list of validation accuracies for different learning rates for input with z-score standardization. The length should be 4.
- best_lr_norm: a float number specifies the best learning rate based on the validation accuracy for input with z-score standardization.
- test_acc_Log_norm: a float number specifies the test accuracy for the best learning rate for input with z-score standardization.

Python function "cost_vs_iter_curve" inputs:

- cost_dict: a dictionary of cost values for each learning rate for input without normalization. Each key is the learning rate. The corresponding value should be a numpy array of cost values with a length of (number of iterations + 1) for the corresponding learning rate. Initial cost before performing gradient descent should be stored in the first element of the numpy array.
- cost_dict_norm: a dictionary of cost values for each learning rate for input with z-score standardization. Each key is the learning rate. The corresponding value should be a numpy array of cost values with a length of (number of iterations + 1) for the corresponding learning rate. Initial cost before performing gradient descent should be stored in the first

element of the numpy array.

Put your analysis of effect of normalization in the Markdown cell right after the Markdown cell "Analysis of Effect of Normalization Based on Your Results"

Data leakage Prevention:

Prevent data leakage by using training data **ONLY** for:

- Pearson correlation calculation and feature selection
- Normalization statistics (mean, std)
- All hyperparameter selection (polynomial order, learning rate) must use validation set only

Restrictions:

You are **NOT** allowed to use the following libraires:

- *sklearn.linear_model.LogisticRegression*
- *sklearn.linear_model.LinearRegression*
- *sklearn.feature_selection*

You can **ONLY** use libraries already imported in the template:

- import pandas as pd
- import matplotlib.pyplot as plt
- import numpy as np
- import sympy as sp
- from sklearn.datasets import fetch_openml
- from sklearn.model_selection import train_test_split
- from sklearn.metrics import accuracy_score
- from sklearn.preprocessing import PolynomialFeatures, StandardScaler, OneHotEncoder

No additional library imports are permitted.

Submission Requirements:

1. Submit a file named **Project_StudentMatriculationNumber.ipynb** (replace "StudentMatriculationNumber" with your own student matriculation number). Please start with the provided template Project_template.ipynb and rename it with your matriculation number.
2. The notebook must be able to run from the first cell to the last cell **without restarting the**

kernel.

3. Make sure all cells execute successfully and **outputs are visible when submitted**.
4. Your notebook must include:

   a) All implemented functions.

   b) Complete execution showing all results with clear output for each part
5. Please do NOT zip/compress your file.

Late Submission Policy:

- Up to 12 hours late: 80% of your grade
- 1 day late: 50% of your grade
- 2 days late: 30% of your grade
- More than 2 days late: No marks (0%)