Instructor: Alina Vereshchaka

Assignment 1

Classification and Regression Methods

Checkpoint Due: Sep 25, Thu, 11:59 pm

Final Due Date: Oct 7, Tue, 11:59 pm

Our Assignment 1 is focused on learning how to build classification and regression models from scratch. In the first part you will perform data preprocessing. In the second part, you will implement logistic regression and test it on a penguin dataset. In the third part, you will implement linear regression using ordinary least squares (OLS) and in the last part, you will extend it to ridge regression. There are also bonus tasks available that you can consider completing.

The goal of this assignment is to give you hands-on experience with building classification and regression models and apply them to solve tasks based on the real-world datasets.

Note: For this assignment, *scikit-learn* or any other libraries with in-built functions that help to implement ML methods cannot be used. Submissions with used ML libraries (e.g. scikit-learn) will not be evaluated.

Part I: Data Analysis & Preprocessing [10 points]

Learning objectives:

- Build and reuse a preprocessing pipeline on multiple real datasets.
- Detect/handle missing values, string formats, and outliers.
- Produce informative visualizations.

Datasets:

Choose three datasets from the 'noisy_datasets' folder provided at UBlearns:

- Penguins dataset. This dataset is slightly different from the one provided in A0.
- Two more datasets: select from the 'noisy_datasets' folder

Step 1: Preprocessing on Penguin [4 points]

1. Import libraries (e.g. pandas, numpy, matplotlib.pyplot (optionally seaborn). Not allowed: *scikit-learn* or other libraries with in-built functions that help to implement ML methods).
2. Load Penguins dataset. Show head(), shape, dtypes. Analyze the dataset, e.g., return the main statistics. Provide a brief description (2-3 sentences) of the dataset: What does it represent / key features.

3. Return basic statistics using describe(). Return missing counts per column table. Which columns have missing values or suspicious values?
4. Handle missing entries. Possible solutions:
   - Drop rows with missing entries. If you have a large dataset and only a few missing features, it may be acceptable to drop the rows containing missing values.
   - Impute missing data. Replace the missing entries with the mean/median/mode of the feature. You can use the k-nearest neighbor algorithm to find the matching sample.
5. Handle mismatched string formats.

   For example, in the penguins dataset, the "Species" feature might appear as "Adelie" or "adelie," both of which refer to the same penguin species. These variations should be standardized to a consistent format such as "Adelie" or "adelie".
6. Handle outliers. Detect and manage outliers within the dataset.

   For example, in the penguins dataset, while flipper lengths typically fall within the range of [180 – 210], certain entries might show values like [10-30]. These can be considered outliers. Possible solutions:
   - Remove outliers. If there are just a few outliers, you may eliminate the rows containing these outliers.
   - Impute outliers. Replace the outliers with the mean/median/mode of the feature.
7. Using any data visualization library (e.g. matplotlib, seaborn, plotly), provide at least 5 visualization graphs related to your dataset. You can utilize any columns or a combination of columns in your dataset to generate graphs. E.g. correlation matrix, features vs. the target, counts of categorical features vs. the target.
8. Identify uncorrelated or unrelated features.

   Note: Unrelated or uncorrelated features can introduce confusion to your model and negatively impact its performance. You can compute the correlation matrix between the features and the target variable. Features with a low correlation coefficient should be identified and subsequently dropped from the dataset to enhance model performance.

   To do this, you first choose a suitable binary target for the dataset. E.g.: you can predict the gender of a penguin (female or male). In this case, the "gender" column would be your target (Y).

   As another example, you can predict whether a penguin's location is Torgersen Island or not, using the "island" column as your target (Y).

   To compute the correlation matrix, load your dataset using Pandas and use the pandas.DataFrame.corr method. You can refer to the documentation here:

   https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html

   You'll need to decide which features to drop based on their correlation with the target.
9. Convert features with string datatype to categorical (e.g., species, island). Possible ways:
   - One-hot encoding, creating binary columns for each category, denoting their presence or absence. E.g., in the "Species" feature, "Adelie," "Chinstrap," and "Gentoo" become binary columns with "1" for presence and "0" for absence.

- Label encoding assigns unique integers to distinct feature values, useful for ordinal relationships among categories. E.g., "Small" as 0, "Medium" as 1, and "Large" as 2 can represent a "Size" feature. However, it may introduce unintended patterns.

10. Normalize non-categorical features (e.g. **bill_length_mm, bill_depth_mm,** flipper_length_mm, body_mass_g).
    a. Find the min and max values for each column.
    b. Rescale dataset columns to the range from 0 to 1

    Why do we do this? Normalization is to transform features to be on a similar scale. This improves the performance and training stability of the model.

    Note: normalize() is not allowed as it is a part of *scikit-learn* library.

Step 2: Refactor into a Reusable Preprocessing Method [3 points]

Now that you have completed all the preprocessing steps for the Penguins dataset, let's refactor your work so you don't have to repeat the same code for every dataset.

1. Create a reusable structure:
   - This can be a class (e.g., PreprocessPipeline) or a clean set of functions.
   - Your code should include:
     - Handling missing values
     - Cleaning string formats
     - Outlier detection and treatment
     - Encoding categorical variables
     - Scaling numeric features
2. Fit and transform on Penguins
   - Run your reusable code on the Penguins dataset.
   - Confirm that the output matches the cleaned dataset you created in Step 1.
   - Print out a short "settings summary" (e.g., how missing values were handled, which features were encoded, min/max for scaling).
3. Save the dataset again as penguins_preprocessed.csv, produced by your reusable code.

Step 3: Apply your reusable code to two other datasets [3 points]

Apply your created Dataset Preprocessing Method to two additional datasets from the noisy_datasets folder.

1. Load the dataset, show head(), shape, and dtypes. Write 1-2 sentences describing what kind of data it is.
2. Select one binary target column (similar to Penguins). Show class counts and note if the dataset is imbalanced.
3. Preprocess using your reusable method. Run your Step-2 code (fit + transform) on the dataset. Show a preview of the transformed dataset (first 5 rows, column names).

4. Visualizations (at least 5). Create five plots to help understand this dataset (e.g., histograms, boxplots, scatterplots, correlation heatmaps). Add a 1-2 sentence insight under each plot.
5. Save the cleaned dataset as DATASETNAME_preprocessed.csv.
6. Print a short summary:
   - Which missingness strategy you used.
   - How you handled outliers.
   - Which columns were encoded.
   - Which numeric features were scaled.
7. Repeat the same for another dataset from noisy_datasets folder.
8. References. Include details on all the resources used to complete this part, e.g. links to datasets, research papers or articles, code examples or tutorials you referred.

FAQs:

- Can I use KNN imputation?

  Yes, but you must implement it yourself. You may use numpy or pandas, but not scikit-learn. Clearly explain how your KNN works.

- How do I know when to drop features?

  Low correlation with a binary target does not automatically mean a feature is useless. Keep features if you think they could contribute in nonlinear or interaction terms. If you drop a feature, justify your reasoning in one sentence.

- How should I handle outliers?

  Start with a simple method such as the IQR rule (value < Q1 − 1.5·IQR or > Q3 + 1.5·IQR). Document your threshold. You can also use domain-specific knowledge (e.g., penguin body mass cannot be 10 g).

- My plots look wrong after scaling/encoding.

  That's normal: use original (pre-scaled) values for interpretability plots, and scaled values for training-ready versions.

- Do I have to use a class for Step 2?

  Not required, but strongly recommended. A clean function-based approach is also acceptable. The key is that you can apply the same code to multiple datasets without rewriting everything.

- Should I fit on train/test separately?

  In Part I, you may use the full dataset. In later parts, you must design your code so it can fit on train and transform test (no leakage).

- Do I need to explain every choice?

  Yes, but briefly. A single clear sentence ("I dropped X because it had 95% missing values") is enough.

Part II: Logistic Regression using Gradient Descent [35 points]

In this part, we will work on logistic regression and will use a logistic function to model a binomial (Binary / Bernoulli) output variable. The logistic regression model predicts that the observation belongs to a particular category. To generate these probabilities, logistic regression uses the sigmoid function that maps a real number to a value between 0 and 1.

Learning objectives

- Implement logistic regression from scratch with gradient descent.
- Diagnose training using loss curves and handle issues like NaNs/divergence.
- Evaluate using metrics: accuracy, precision, recall, F1-score, and a confusion matrix.
- Understand how hyperparameters (learning rate, iterations, initialization) affect performance.

DATASET

Penguins Dataset: Use the preprocessed dataset from Part I.

STEPS:

1. Import required libraries (not allowed: *scikit-learn* or other libraries with in-built functions that help to implement ML methods).
2. Choose target variable Y. For this dataset, there are several options:
   - Predict which gender a penguin belongs to (female or male). In this case, column 'gender' can be used as Y (target)
   - Predict if a penguin's location is Torgersen Island or not. In this case, column 'island' can be used as Y (target)
   - Show class counts for your chosen target.
3. Create the data matrices for X (input) and Y (target) in a shape $\mathrm{X} = N \times d$ and $\mathrm{Y} = N \times 1$, were N is a number of data samples and d has a number of features.
4. Split into train/test sets
   - Use an 80% / 20% split.
   - Print the shape of X_train, y_train, X_test, y_test.
   - (Optional but recommended: set a fixed random seed for reproducibility.)
5. Recommended structure of your code to define logistic regression:

```
class LogitRegression()

  def __init__()
     # Takes as an input hyperparameters: learning rate and the number
     of iterations.
```

```
def sigmoid():
    # Define a sigmoid function as
```

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

```
def cost():
    # Loss function for Logistic Regression can be defined as
```

$$\mathrm{h} = \sigma(w^T x + b) = \left(\frac{1}{1+e^{-(w^T x+b)}}\right)$$

$$J(w) = \frac{1}{N}(-y * \log(h) - (1-y) * \log(1-h)$$

```
def gradient_descent():
    # Define current prediction y_hat for logistic regression as
```

$$\hat{y} = \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{X} + \mathrm{b})$$

```
    # Gradient descent is just the derivative of the loss function
    with respect to its weights. Thus:
```

$$\frac{\partial J(w)}{\partial w} = \frac{1}{N} X^T(\sigma(w^T x + b) - y)$$

```
    To implement this formula, you can use intermediate variables,
    e.g.
        pred = σ(w^T x + b)
        delta = pred – y_train
        dW = (X^T*delta)/N

    Update rule: w = w − α ∇J(w), where α is a learning rate


def fit():
    # This method performs the training.
    # Initialize weights
    # For a number of iterations
        # Call gradient_descent function
        # Call cost function and keep it in an array, e.g.
        loss.append()

def predict(self, X):
    # Return the predicted result in the binary form
```

$$\hat{y} = \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{X} + \mathrm{b}) = \begin{cases} +1 & \text{if } \sigma(z) \geq 0.5 \\ 0 & \text{if } \sigma(z) < 0.5 \end{cases}$$

Hints:

- Initialize weights randomly (e.g., uniform [0,1]) or with small zeros.
- Clip extreme values if the sigmoid overflows (e.g., when z is very large).
- Store loss at each iteration for plotting.

6. Train the model:
   - Define a model by calling LogitRegression class and passing hyperparameters, e.g.

   ```
   model = LogitRegression(learning_rate, iterations)
   ```

   - Train the model, by calling fit function and passing your training dataset, e.g.

   ```
   model.fit(X_train, y_train)
   ```

   - Try at least THREE various hyperparameters. You can try different learning rates and number of iterations to improve your accuracy (accuracy of greater than 64% is expected). Suggested hyperparameters:

   ```
   learning_rate=1e-3

   iterations=100000

   weights = np.random.uniform(0, 1)
   ```

7. Evaluate performance
   - On the test dataset, calculate:
     - Accuracy. **Accepted accuracy rate based on the test dataset for penguin dataset is above 64%**
     - Precision, Recall, F1-score (print all values)
     - Confusion matrix (2×2, nicely formatted)
   - Compare metrics across your three hyperparameter setups.
8. Plot and analyze loss curves
   - Plot loss vs. iteration for each hyperparameter setup.
   - Add a short description under each plot: does the model converge, diverge, plateau?
   - How did learning rate and iterations influence convergence and accuracy?
9. Save the weights of the model, that returns the highest accuracy as a1_part_2_weights_TEAMMATE1_ TEAMMATE2.pkl or .pickle. You will need to submit it along with your other files. Check more details about Pickle
10. Discussion (write in markdown cell)
    - Report your best-performing hyperparameters and explain why they worked best.
    - Summarize the advantages and drawbacks of logistic regression (e.g., simplicity, interpretability vs. linear decision boundary, sensitivity to scaling).
    - Mention any issues you faced (NaNs, slow convergence) and how you fixed them.
11. References. Include details on all the resources used to complete this part, e.g. links to datasets, research papers or articles, code examples or tutorials you referred.

FAQs:

- What if my loss becomes NaN?
  - Lower your learning rate (e.g., from 1e-2 to 1e-4).
  - Clip input to sigmoid between [−500, +500] to avoid overflow.
- What if accuracy is below 64%?

  - Check: (1) Did you normalize features? (2) Did you encode categorical variables correctly?
  - Try smaller/larger learning rates and more iterations.

- Can I add regularization?
  Not required here, but if you experiment, explain your method and show its effect.
- Should I randomize weight initialization?
  Yes, but set a random seed for reproducibility. Different seeds may affect convergence speed.
- How do I format the confusion matrix?
  A simple 2×2 table is fine. Label rows = actual, cols = predicted.

Part III: Linear & Ridge Regressions using OLS [25 points]

Implement linear regression using the ordinary least squares (OLS) method to perform direct minimization of the squared loss function.

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - w^T x_i)^2$$

In matrix-vector notation, the loss function can be written as:

$$J(\mathbf{w}) = (\mathbf{y} - \mathbf{X}\mathbf{w})^\top (\mathbf{y} - \mathbf{X}\mathbf{w})$$

where $\boldsymbol{X}$ is the input data matrix, $\boldsymbol{y}$ is the target vector, and $\boldsymbol{w}$ is the weight vector for regression.

Learning objectives

- Implement closed-form OLS and ridge regression (numerical stability).
- Compare models via MSE and R^2 and diagnostic plots.
- Understand how regularization trades variance for bias.

DATASET

Use any dataset from the noisy_datasets folder, excluding Penguins.

- You may reuse a dataset you already preprocessed in Part I, or choose a new one.
- Note: For the Wine dataset, you must combine the two CSV files into one.

STEPS:

1. Import required libraries (not allowed: *scikit-learn* or other libraries with in-built functions that help to implement ML methods).
2. Data analysis & preprocessing
   - Reuse your preprocessing code from Part I (missing values, outliers, encoding, scaling).

- Ensure that scaling is done after splitting into train/test to avoid leakage.
- Describe the dataset (domain, number of samples, features, target).

3. Data preparation:
    a. Define the target variable Y
    b. Create matrices for input X and target Y with shape $\mathrm{X} = N \times d$ and $\mathrm{Y} = N \times 1$, where N is a number of data samples and d is a number of features.
    c. Split into training (80%) and testing (20%) sets.
    d. Print the shapes of X_train, y_train, X_test, y_test.
4. Linear Regression using OLS:
    a. Calculate the weights with the OLS equation:

$$\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

    b. Predict on training and testing sets.
    c. Compute evaluation metrics:
        - Mean Squared Error (MSE)
        - Coefficient of determination (R^2)
    d. Plot predicted vs actual values:
        - For univariate target, scatter plot predicted vs actual with a reference line y=x.
        - If using more than 2 features, show predictions on y-axis and samples on x-axis.
5. Ridge Regression using OLS:
    a. Implement Ridge regression by minimizing the regularized squared loss:

$$J(\boldsymbol{w}) = \frac{1}{2N}\sum_{i=1}^{N}(y_i - w^T x_i)^2 + \frac{1}{2}\lambda \boldsymbol{w}^T\boldsymbol{w}$$

    b. Compute the weights using the ridge regression OLS equation:

$$\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X} + \boldsymbol{\lambda I})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

    c. Use at least three different λ values (e.g., 0.01, 0.1, 1, 10).
    d. Predict on training and testing sets for each λ.
    e. Compute metrics (MSE and R^2).
    f. Plot predictions vs actual values for each λ.
    g. Plot a line graph of test MSE vs λ to show the effect of regularization.
    h. Save weights that returns the best results as a1_part_3_weights_TEAMMATE1_TEAMMATE2.pkl or .pickle
6. Discussion (write in the markdown cell):
    - What are the benefits and drawbacks of using closed-form OLS?
    - What are the strengths and weaknesses of linear regression in general?
    - Why does ridge regression help? What bias/variance tradeoff did you observe?

- For your dataset, did ridge regression improve generalization compared to plain OLS?

7. Save:
   - Save your best weights as TEAMMATE1_TEAMMATE2_assignment1_part2.pickle.
   - Keep your notebook (assignment1_part_2.ipynb) with outputs and plots visible.

FAQs:

- What if $X^T X$ is singular?
  - Use np.linalg.pinv (pseudo-inverse) instead of np.linalg.inv.
  - Ridge regression naturally fixes this by adding λI.
- How should I choose λ?
  - Start with small values (0.01, 0.1) and increase (1, 10).
  - Too large λ will underfit; too small may behave like plain OLS.
- Why do we compute R^2 in addition to MSE?

  MSE shows average squared error. R^2 shows how much variance is explained by the model (a more interpretable measure).
- Should I evaluate on both train and test?

  Yes, compare train vs test to detect overfitting. Ridge should help reduce overfitting.
- What if my dataset is categorical-heavy?

  Ensure encoding is done before applying OLS. One-hot encoding may expand feature space, which makes ridge regularization more useful.

Part IV: Elastic Net Regression using Gradient Descent [30 points]

In this part, you will implement Elastic Net Regularization using gradient descent to solve a regression problem. You will explore different weight initialization techniques and stopping criteria to evaluate their impact on model performance.

Learning objectives:

- Implement Elastic Net (L1+L2) regression using gradient descent.
- Explore weight initialization strategies and their impact on convergence.
- Compare stopping criteria (fixed iterations vs. gradient threshold).
- Evaluate how different regularization strengths affect performance.

DATASET

Use any dataset from the noisy_datasets folder, excluding Penguins and the dataset used in Part III.

- Datasets for Part II, III, and IV must be different.
- Note: For the Wine dataset, combine the two CSV files into one.

STEPS

1. Import required libraries (not allowed: *scikit-learn* or other libraries with in-built functions that help to implement ML methods).
2. Data analysis & preprocessing
   - Reuse your preprocessing code from Part I (missing values, outliers, encoding, scaling).
   - Ensure that scaling is done after splitting into train/test to avoid leakage.
   - Describe the dataset (domain, number of samples, features, target).
3. Data preparation
   a. Define the target variable Y
   b. Create matrices for input X and target Y with shape $\mathrm{X} = N \times d$ and $\mathrm{Y} = N \times 1$, where N is a number of data samples and d is a number of features.
   c. Split the dataset into training (80%) and testing (20%) sets.
   d. Print the shapes of X_train, y_train, X_test, y_test.
4. Implement Elastic Net Regularization
   Elastic Net combines L1 (Lasso) and L2 (Ridge) regularization.
   The regularized loss function is defined as:

$$J(\boldsymbol{w}) = \frac{1}{2N}\sum_{i=1}^{N}(y_i - w^T x_i)^2 + \frac{1}{2}\lambda_1 \left|\left|w\right|\right|_2^2 + \lambda_2 \left|\left|w\right|\right|_1$$

   Where λ_1 and λ_2 are the regularization parameters for L2 and L1 penalties.
5. Solve the problem using Gradient Descent. Implement gradient descent from scratch to minimize the Elastic Net regularized loss function.
   Update the weights iteratively using:

$$w = w - \alpha \nabla J(w)$$

   where α is the learning rate and $\nabla J(w)$ is the gradient of the loss function.
6. Experiment with at least three different methods for weights initializing. Some ideas:
   a. **Random**: initialize weights randomly
   b. **Zero**: initialize all weights to zero
   c. **Xavier (Glorot) Initialization:** initialize weights in such a way that helps to maintain the variance of activations and gradients throughout the layers. E.g.

```
def xavier_initialization(input_dim, output_dim):
    # Calculate the limit for uniform distribution
    limit = sqrt(6 / (input_dim + output_dim))

    # Initialize weights with uniform distribution in the range
[-limit, limit]
    weights = random.uniform(-limit, limit, size=(input_dim,
output_dim))

    return weights
```

   d. Discuss and analyze. Which method converged faster? Which was more stable? How initialization affects:
      - Speed of convergence
      - Final loss
      - Stability of training

7. Experiment with training stopping criteria (≥2 methods):
   a. Run gradient descent for a predefined number of iterations, e.g., 10,000 or 100,000.
   b. Stop the algorithm when the gradient falls within a small threshold, e.g., $-0.01<\text{gradient}<0.01$.
   c. Compare the performance and convergence behavior of these stopping criteria.
   d. Discuss: Did early stopping prevent overfitting or save computation time?
   e. Save weights that returns the best results as a1_part_4_weights_TEAMMATE1_ TEAMMATE2.pkl or .pickle
8. Evaluation
   - Report metrics:
     - **MSE** (Mean Squared Error) on train and test
     - **R^2** on train and test
   - Create plots:
     - Predictions vs. actuals (scatter with y=x line)
     - Convergence plots (loss vs iterations)
     - Test loss/MSE vs λ values (if you experiment with different regularization strengths)
9. Analysis & reflection (markdown cells)
   - Elastic Net vs. Ridge/Lasso: Compare advantages. Did Elastic Net improve results for your dataset?
   - What are the benefits and limitations of Elastic Net + GD?
10. Save:
    - Save your best model weights as:

      assignment1_part4.pickle.

- Save your notebook (assignment1_part_4.ipynb) with all outputs visible.

FAQs:

- How should I handle the L1 gradient?

  Use subgradients: sign(w) for each coefficient. At w=0, any value in [−1,1] is valid — you may use 0.

- What if training diverges?
  - Lower the learning rate.
  - Clip gradients or standardize inputs if needed.
- How to choose λ_1 and λ_2?
  - Start small: $\lambda_1 = 0.01$, $\lambda_2 = 0.001$.
  - Try increasing values (0.1, 1, 10) and observe tradeoffs.
- Why multiple initialization strategies?
  - To see how starting values impact convergence.
  - Random init may help escape poor basins, but Xavier often converges faster.
- When should I stop training?
  - If using fixed iterations, check loss curve to ensure it converges.
  - If using gradient threshold, confirm that early stopping doesn't cut training too soon.
- Should I compare Elastic Net to Ridge/Lasso?

  Yes, mention in your reflection how Elastic Net balances feature selection (L1) and shrinkage (L2).

Bonus points [max 10 points]

The bonus section is optional but strongly encouraged if you want to challenge yourself further or explore interesting datasets. You may attempt one or both parts.

Buffalo-related Dataset [8 points]

A key part of any research is identifying a suitable dataset for your problem.

Steps:

1. Find a dataset
   - Choose a Buffalo-related dataset from the Buffalo Open Data Portal.
   - Minimum size: >1,000 entries.
   - Example domains: housing, transportation, environment, public services.
2. Perform data analysis as in Part I (missing values, outliers, encoding, scaling, visualizations).
3. Apply at least one method
   - Apply a method you implemented in this assignment (Logistic Regression,

OLS/Ridge, Elastic Net).

- You may try multiple methods if you want.

4. Train and evaluate
   - Report accuracy if classification, or MSE/R^2 if regression.
   - Target goal: **Accuracy ≥70% (or equivalent regression performance).**
   - If results are lower, justify why (e.g., dataset imbalance, limited features, noise).
5. Visualizations. Include at least 5 plots that help understand the dataset and your model's performance.
6. Save model weights (pickle) and cleaned dataset (CSV).

Improved Accuracy on Penguin dataset [2 points]

Steps:

1. Experiment with improvements
   - Try feature engineering (combine features, create interaction terms).
   - Adjust hyperparameters (learning rate, iterations, initialization).
   - Add regularization terms (L1, L2).
   - Explore smarter preprocessing (e.g., handling class imbalance, scaling variations).
2. **Target goal: achieve testing accuracy >85%.**
3. Save best weights as pickle.

Bonus part submission:

- Create a separate Jupyter Notebook (.ipynb) named as TEAMMATE1_TEAMMATE2 _assignment1_bonus.ipynb
  e.g., avereshc_ pinazmoh_ assignment1_bonus.ipynb
- You can duplicate code from your other parts if needed
- Submit Jupyter Notebook (.ipynb) with saved outputs
- A file with saved weights that generate the best results for your model (.pickle).
- Report is not required; you can include all the analysis as part of your Jupyter Notebook.
- Submit your Bonus related files as part of your Final Submission

ASSIGNMENT STEPS

1. Register your team (September 16)

- You may work individually or in a team of up to 2 people.

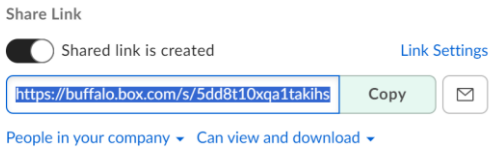
- The evaluation will be the same regardless of team size.
- Register your team at UBLearns > Groups. If you joined the wrong group, make a private post on Piazza.

2. Submit checkpoint: Part I, Part II (September 25)

- Complete Part I and Part II of the assignment.
- Include all the references at the end of each part.
- Your Jupyter notebook should be saved with the outputs.
- If you are working in a team, we expect equal contribution for the assignment. Each team member is expected to make a code-related contribution. Provide a contribution summary by each team member in the form of a table below. If the contribution is highly skewed, then the scores of the team members may be scaled w.r.t the contribution.

Team Member	Assignment Part	Contribution (%)

- Datasets:
  - Go to UBbox > CSE 574-D Assignment 1 by TEAMMATE 1 & TEAMMATE 2
  - Upload your chosen datasets into this folder that you used for this assignment.
  - Copy a shared link to the UBbox folder, so it can be viewed by people in your company & it can be viewed and downloaded.

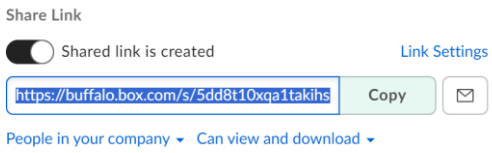


  - Add the link to the txt file, named as a1_datasets_TEAMMATE1_ TEAMMATE2.txt
  - Submit this txt file as part of your submission on UBlearns
- Saved weights that generate the best results for your model, named as a1_part#_TEAMMATE1_ TEAMMATE2.pkl or .pickle
- e.g. a1_part1_avereshc_ manasira.pickle
- Combine all files in a single zip folder that will be submitted on UBlearns, named as assignmen1_checkpoint_YOUR_UBIT.zip
- Submit to UBLearns > Assignments
- Suggested file structure:

assignment_1_checkpoint_TEAMMATE1_ TEAMMATE2.zip
- a1_part_1_ TEAMMATE1_ TEAMMATE2.ipynb
- a1_part_2_ TEAMMATE1_ TEAMMATE2.ipynb
- a1_part_2_weights_TEAMMATE1_ TEAMMATE2.pkl
- a1_datasets_TEAMMATE1_ TEAMMATE2.txt

3. Submit final results (October 7)

- Complete all parts of the assignment (I–IV, plus Bonus if attempted).
- Add all your assignment files in a zip folder including ipynb files for Part I, Part II, Part III, Part IV, Part V & Bonus part (optional) and txt file with a link to UBbox with links to weights and datasets.
- Jupyter Notebooks must contain saved outputs and plots visible.
- Include all references at the end of each part.
- You may make unlimited submissions — only the latest will be evaluated.
- Datasets:
  - Go to UBbox > CSE 574-D Assignment 1 by TEAMMATE 1 & TEAMMATE 2
  - Upload your chosen datasets into this folder that you used for this assignment.
  - Copy a shared link to the UBbox folder, so it can be viewed by people in your company & it can be viewed and downloaded.



  - Add the link to the txt file, named as a1_datasets_TEAMMATE1_ TEAMMATE2.txt
  - Submit this txt file as part of your submission on UBlearns
- Name zip folder with all the files as assignment1_final_ TEAMMATE1_ TEAMMATE2.zip

e.g. assignment1_final_avereshc_manasira.zip

- Submit to UBLearns > Assignments
- Your Jupyter notebook should be saved with the outputs.
- Include all the references at the end of each part that have been used to complete that part.
- You can make unlimited number of submissions and only the latest will be evaluated
- If you are working in a team, we expect equal contribution for the assignment. Each team member is expected to make a code-related contribution. Provide a contribution summary by each team member in the form of a table below. If the contribution is highly skewed, then the scores of the team members may be scaled w.r.t the contribution.

Team Member	Assignment Part	Contribution (%)

- Suggested file structure (bonus part is optional):

  assignment_1_final_TEAMMATE1_ TEAMMATE1.zip
  - a1_part_1_ TEAMMATE1_ TEAMMATE2.ipynb
  - a1_part_2_ TEAMMATE1_ TEAMMATE2.ipynb
  - a1_part_3_ TEAMMATE1_ TEAMMATE2.ipynb
  - a1_part_4_ TEAMMATE1_ TEAMMATE2.ipynb
  - a1_part_2_weights_TEAMMATE1_ TEAMMATE2.pkl
  - a1_part_3_weights_TEAMMATE1_ TEAMMATE2.pkl
  - a1_part_4_weights_TEAMMATE1_ TEAMMATE2.pkl
  - a1_datasets_TEAMMATE1_ TEAMMATE2.txt
  - a1_bonus_ buffalo_TEAMMATE1_ TEAMMATE2.ipynb
  - a1_bonus_ improved_TEAMMATE1_ TEAMMATE2.ipynb

Notes:

- Ensure your code is well-organized and includes comments explaining key functions and attributes. Also ensure that all the section headings for your solutions corresponding to this assignment are displayed. After running the Jupyter Notebooks, all results and plots used in your report should be generated and clearly displayed.
- You can submit multiple files, but they all need to be labeled with a clear name.
- Recheck the submitted files, e.g. download and open them, once submitted and verify that they open correctly
- Large files: any individual file, except .ipynb file, that is larger then 10MB (e.g. your model weights or datasets) should be uploaded to UBbox and you have to provide a link to them. A penalty of -10pts will be applied towards the assignment if there is a file submitted that is bigger than 10MB.
- It's ok if your final combined zip folder is larger then 10MB.
- The zip folder should include all relevant files, clearly named as specified.
- Only files uploaded on UBlearns and a submitted link to UBbox for saved datasets are considered for evaluation.

Academic Integrity

The standing policy of the Department is that all students involved in any academic integrity violation (e.g., plagiarism in any way, shape, or form) will receive an F grade for the course.

The catalog describes plagiarism as “Copying or receiving material from any source and submitting that material as one’s own, without acknowledging and citing the particular debts to the source, or in any other manner representing the work of another as one’s own.”. Refer to the Office of Academic Integrity for more details.

Important Information

This assignment can be done individually or in a team of up to two students. The grading rubrics are the same for a team of any size.

- No collaboration, cheating, and plagiarism is allowed in assignments, quizzes, the midterms or final project.
- All the submissions will be checked using MOSS as well as other tools. MOSS is based on the submitted works for the past semesters as well the current submissions. We can see all the sources, so you don't need to worry if there is a high similarity with your Checkpoint submission.
- The submissions should include all the references. Kindly note that referencing the source does not mean you can copy/paste it fully and submit as your original work. Updating the hyperparameters or modifying the existing code is a subject to plagiarism. Your work has to be original. If you have any doubts, send a private post on piazza to confirm.
- All parties involved in any suspicious cases will be officially reported using the Academic Dishonesty Report form. Please refer to the Academic Integrity Policy for more details.

Late Days Policy

You can use up to 7 late days throughout the course that can be applied to any assignment-related due dates. You do not have to inform the instructor, as the late submission will be tracked in UBLearns.

If you work in teams, the late days used will be subtracted from both partners. In other words, you have 4 late days, and your partner has 3 late days left. If you submit one day after the due date, you will have 3 days, and your partner will have 2 late days left.

Important Dates

Sep 25, Thursday - Checkpoint is Due

Oct 7, Tuesday - Final Submission is Due