CS233: Introduction to Machine Learning
Course Project

Important Information:

- This will be a group project and must be done in **groups of 3.**
- You will choose **one** of three datasets to build your project around: human poses, movie summaries or music. We will introduce all three datasets in this document.
- You have until **October 2nd** to choose a group and a dataset.
- You may post your questions about the project and search for group members on the "Student Forum" on Moodle.
- We have kept the exercise sessions of weeks 7-8 and 13-14 free so that you can make progress on your projects and ask your questions to the TAs.
- Here is a link to register your group on the Moodle page.
- **Important deadlines**:
  - Group registration and dataset selection: 2 October (23:59)
  - Milestone 1: 11 November (10:00)
  - Milestone 2: 23 December (10:00)
- This is the first time this course will have a course project. Please give us feedback so that we can continue to improve the course.

The Datasets

You will have the option to choose between three datasets to work on for the project:

- Human pose dataset: This dataset contains human pose sequences, represented as time series of 3D joint locations. We will be working on the tasks of classifying the action label of each sequence, and regressing the future poses of each sequence.
- Movie summaries dataset: This dataset contains feature vectors corresponding to movie summaries. We will be working on classifying the genre of the movie, and regressing the rating.

- Music dataset: This dataset contains features extracted from the audio files of songs. We will be working on classifying the song genres, and regressing the custom rating of each song.

Below we describe each dataset in detail.

Human Pose

- The human pose dataset is a subset of the popular Human 3.6M dataset.
- We have selected sequences from the following actions: walking, sitting down, directions, posing. These actions are performed by 6 subjects (but luckily for you, the data is already preprocessed so that the actor's skeletons are uniformized!).
- We will demonstrate two tasks on this dataset:
  - Action classification: By looking at sequences of 2 seconds, we will try to classify the sequences into one of the four action classes.
  - Future pose regression: By looking at sequences of 2 seconds, we will try to predict the poses in the next 1 second.
- The dataset is split into three parts: training, validation and testing.
- The `project/data.py` file contains the dataset class `H36M_Dataset`. This class already implements the functions to load the data split, do some processing on the joints and normalize the data.
- Here are some explanations about the format of the data:
  - We load the pose sequences (for example: `train_data.npy`) and action labels (`train_labels.npy`) using the np.load function.
  - When the data is loaded, it has the shape: (N, 75, 22, 3). N is the number of sequences (2589 for training data, 409 for validation data and 823 for test data). 75 is the number of frames in each sequence (25 frames is 1 second). 22 is the number of joints. 3 is the number of Cartesian coordinates (X, Y, and Z).
  - The sequences are later split into past and future sequences: (N, 50, 22, 3) and (N, 25, 22, 3). They are then reshaped so that they are of shape (N, 50*22*3) and (N, 25*22*3).

    - The action labels are of shape (N,) and contain integer labels in {0,1,2,3} (for four action classes).

Movie Summaries

- We use the [CMU movie summary corpus dataset]() for movie genre prediction and custom rating prediction. The dataset contains following features: movie ID, name, custom rating, box office revenue, runtime, languages, countries, plot summary, and genres. Here, we will only use the movie plot summary as input to our algorithms. We have preprocessed the raw data, which includes grouping and cleaning data, vectorizing words, etc.
- We will demonstrate two tasks on this dataset:
    - Genre classification: Given a feature vector which represents a movie plot summary, we will try to predict the corresponding genre (drama, comedy, documentary, romance, action) as a classification task.
    - Rating regression: We will try to predict the rating of a movie based on the feature vectors. We note that this is a feature that was generated by the TA team specifically for this project.
- The dataset is split into three parts: training, validation and testing.
- The `project/data.py` file contains the dataset class `Movie_Dataset`. A data loading function has been provided and can be used to load the training, validation, and testing data.
- Here are some explanations about the format of the data:
    - The numpy files contain the vectorized movie plot summary (for example `train_data.npy`) and the genre labels (for example `train_labels.npy`).
    - When the data is loaded, the shape is (N, 128). N is the number of movies. 128 is the length of a vectorized plot summary.
    - The genre labels are of shape (N,) and contain integer labels in {0,1,2,3,4} (for five genre classes).
    - The rating labels are of shape (N,)..

Music

- We are using a subset of the [FMA dataset]() for music analysis, with the data modified for our project. This dataset contains precomputed features for around 25.000 tracks. Each track is assigned with a genre and a rating.
- The two tasks we are interested in are as follows:
  - Genre classification: Given the precomputed features for each track, we will try to classify the track into one of the three genres.
  - Rating: We will try to predict the rating of a song based on the precomputed features. We note that this is a feature that was generated by the TA team specifically for this project.
- There are three splits for the dataset: training, validation and testing (%80 + %10 + %10).
- The FMA_Dataset class in project/data.py file already splits and preprocesses the data. Here is the format that you should expect from this pre-implemented data loader:
  - The data is stored in the 'tracks.csv' and 'features.csv' files. The 'tracks.csv' file contains all the metadata about the track, including the top genre and the custom rating, whereas 'features.csv' stores pre-computed audio features for each track.
  - In the load_data function, we load the two files, split the dataset and normalize the features.
  - The input features have the shape (N, 231), where N is the number of tracks and 231 is the number of features. These features are the statistics of all the features a music analysis library called librosa can compute. Specifically, intermediate features are computed with librosa over sliding windows on 30 second tracks. From these intermediate features, 7 statistics (mean, median, min, max, std, skew, kurtosis) are computed along the time axis.
  - The regression and classification targets have shape (N,1) and (N,) respectively (N=19922 for training set, i.e., ~%80 of 25000). The regression label is the custom rating, and the classification target is an integer between 0 and 2 (3 genres: Hip-Hop, Pop, Rock).
  - For more details, see [FMA: A Dataset For Music Analysis]()

Please register your groups and your dataset at this [link]() before October 2nd!

The Tasks and Metrics

For our project, we will be working on both a regression task and a classification task. As discussed above, each dataset has its own regression and classification tasks, but we can use common metrics to measure our success.

Classification Task

For the classification task, we will report the macro F1 score, which is an average of class-wise F1 scores. For a *class* = *i*, we can write its F1 score as

$$\mathrm{F1_i} = \frac{2 \times (\mathrm{P_i} \times \mathrm{R_i})}{\mathrm{P_i} + \mathrm{R_i}}$$

where $\mathrm{P_i}$ and $\mathrm{R_i}$ are the precision and recall values for the *class* = *i* . Such precision and recall values are computed as

$$\mathrm{P} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

$$\mathrm{R} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

For K classes, we can compute a macro F1 score

$$\mathrm{macro\,F1} = \frac{1}{K}\sum_{i=1}^{K} \mathrm{F1_i}$$

Higher F1 scores correspond to a better model and macro F1 reflects the model’s average performance on all the classes.

We will also report the average accuracy in percentage. We can write this as:

$$\text{accuracy} = \frac{\text{number of correct predictions}}{\text{total number of predictions}} \times 100$$

Note that for some of the datasets, we have class imbalance (meaning that the dataset is not divided evenly among the different class labels), and the F1 score is thus more meaningful than the accuracy metric.

Regression Task

We will report the mean-squared error (MSE) of the regression outputs. For a regression target with a single dimension, the MSE is computed as

$$\text{MSE} = \frac{1}{N_t}\sum_{i=1}^{N_t}(\hat{y}_i - y_i)^2$$

where N_t is the number of test samples, $\hat{y}_i$ is the prediction of sample i and y_i is the ground truth value of sample i.

When the regression target has multiple dimensions (i.e., the dimension of the vector is greater than 1), we calculate the mean of the features as well. This is expressed as

$$\text{MSE} = \frac{1}{C}\frac{1}{N_t}\sum_{i=1}^{N_t}\sum_{d=1}^{C}(\hat{\mathbf{y}}_i^{(d)} - \mathbf{y_i^{(d)}})^{\mathbf{2}}$$

where $\hat{\mathbf{y}}_i^{(d)}$ and $\mathbf{y}_i^{(d)}$ refer to feature d for sample i of the ground truth and prediction vectors respectively, and C is the total number of features.

Runtime Analysis

It is generally very useful to report how fast your algorithms can be trained. You can easily measure how fast a script runs in Python using the `time` module. Here is a quick example:

```
import time
s1 = time.time()
dummy_function()
s2 = time.time()
print("Dummy function takes", s2-s1, "seconds")
```

The Framework

- The framework is organized with the following folder structure:

```
<sciper1>_<sciper2>_<sciper3>_project

    __init__.py
    data.py
    main.py
    metrics.py
    utils.py
    <sciper1>_<sciper2>_<sciper3>_report.pdf
    methods
        __init__.py
        cross_validation.py
        deep_network.py
        dummy_methods.py
        knn.py
        linear_regression.py
        logistic_regression.py
        pca.py
```

- Please do not rename these files, nor move them around.
- Your report will also be a part of this folder structure (as <sciper1>_<sciper2>_<sciper3>_report.pdf).
  Please include it in your framework when you are preparing your Milestone 1 and Milestone 2 submissions. You will be submitting the zipped "project" folder.
- Add your dataset inside the project folder (you will download the datasets from moodle), but make sure you remove it for the submission (otherwise the file size will get very large!)
- Some parts of the code are provided to you to get you started. It will be your task to fill in the missing parts to get all the methods running on your data.

Test Script

- We also provide you with a test script: `test/test_ms1.py` or `test/test_ms2.py`, depending on the milestone. This script is for you to verify that your project (folder and code structure) is compatible with our grading system.

- While the methods are also tested on dummy data, this is not exhaustive and you are expected to verify the correctness of your code by your own means.
- The script can be launched as
    - `python test/test_ms1.py -p path_to_project_folder`

  where `path_to_project_folder` points to your (uncompressed!) directory: **`<sciper1>_<sciper2>_<sciper3>_project`**.

Running the Project

You will be running the Python scripts from your terminal. For example the main script can be launched from inside the project folder as:

```
python main.py --dataset="h36m" --path_to_data=<where you placed the
data folder> --method_name="dummy_classifier" --use_cross_validation
```

This will run the framework on the H36M dataset, using the dummy classifier and cross validation.

You can specify the method, the dataset, and many other arguments from the terminal. The arguments are defined at the bottom of the `main.py` script, we encourage you to check how they work, and what are their default values.

Milestone 1

- You need to submit your code and a 1 page project report (zipped together).
  The project report should be part of your folder structure (see above!)
  You should name the project folder <sciper1>_<sciper2>_<sciper3>_project, and the zipped file should have the name <sciper1>_<sciper2>_<sciper3>_project.zip. Please make sure that once you unzip the zip file, it has the name <sciper1>_<sciper2>_<sciper3>_project.
- What you are expected to have running by the end of Milestone 1:
    - Ridge/Linear Regression (they can be written in the same class, as linear regression is ridge regression when the regularization parameter is set to 0.)
    - We note that we will be implementing the closed-form solution.
    - Logistic Regression

    - Cross validation
- The project report should include a brief summary of the results you have achieved with the methods.
    - You should explain what hyperparameters you tried for these methods and what results you achieved. Adding simple visualizations of your validation losses with respect to the different hyperparameters would be a plus!
    - You should include a discussion on which methods you would prefer to use for your tasks, and why. You can justify this with the results you obtain, the simplicity of the model, and the training time of the models.
- The methods are all implemented in the form of Python classes. We have provided the implementation of some "dummy" methods as examples (they are in project/methods/dummy_methods.py). Study these classes well, you can use them as templates to code your own methods.
- In particular, dummy_classifier returns a random class as the result of classification. It contains some essential functions:
    - `__init__(self, *args, **kwargs)`
        - This function is used to initialize the class. Essentially, it calls its __set_arguments__ function to set some essential arguments.
    - `set_arguments(self, *args, **kwargs)`
        - Sets the hyperparameters of the particular model. `*args` is a list of hyperparameter values. `**kwargs` is a dictionary with keys and values which correspond to hyperparameters.
    - `fit(self, training_data, training_labels)`
        - This is the training or fitting step of the model. Training data with corresponding labels are used to estimate the parameters of the model. This method should be called before the `predict` function.
    - `predict(self, test_data)`
        - It generates predictions for the given data. It should be called after the `fit` function.
- You will need to complete the relevant parts of the following scripts for this milestone:
    - `main.py`
    - `methods/linear_regression.py`

        - We suggest your linear (ridge) regression object takes the argument "lambda". You can search for the best lambda value using cross-validation. When lambda is set to 0, the method becomes linear regression.
    - `methods/logistic_regression.py`
        - We suggest your logistic regression object takes the arguments "learning rate" and "max epochs". You can search for either the best learning rate or the best number of max epochs using cross-validation.
        - Is logistic regression a regression or classification method? Be careful!
    - `methods/cross_validation.py`
        - We have partially filled in this file for you.
            1. In the beginning, we check whether the method is a classification or a regression method and determine the metric according to this. If the method solves a regression task, then the metric is set to MSE, and you should find the hyperparameter that gives the *minimum* MSE. If the method solves a classification task, then the metric is set to macro F1, and you should find the hyperparameter that gives the *maximum* macro F1.
            2. We shuffle the list of indices.
            3. We have a for loop that iterates over `search_arg_vals`. For each argument in this list, we call the `set_arguments` function of the method object.
            4. We have a for loop for each fold in the cross validation.
            5. And the rest is up to you!
- Make sure that `test/test_ms1.py` runs without any problems! This ensures that your code compiles and we are able to import your implementations without problems.
- Make sure that your method classes do not modify the data they are passed. If you are doing some form of data augmentation (such as adding a bias term, doing feature expansion etc.), then you must do so *outside* these classes, such as in your `main.py` script.
- Make sure your implementations of these methods are not dataset specific - they should work for arbitrary numbers of samples, dimensions, classes etc.

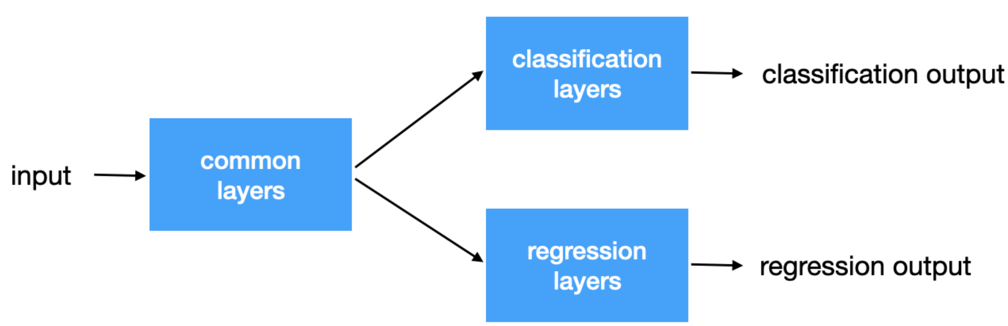
Expected results for each dataset:

We have posted the results we have achieved on these datasets.
You can use these results to sanity-check your implementations. We do not expect you to get the same numbers. However, if you have significantly worse results, then you might have a bug.

	Linear Regression	Ridge Regression	Logistic Regression
H36M	331 MSE	0.37 MSE	69% acc, 0.67 F1 score
Movies	0.85 MSE	0.85 MSE	49% acc 0.39 F1 score
Music	0.51 MSE	0.50 MSE	80% acc, 0.67 F1 score

Milestone 2

- We will update this part of the document once we conclude Milestone 1. A rough tentative description of Milestone 2 is described below.
- What you need to submit: your code and a 2 page project report (zipped).
- For this milestone, you'll need to design a neural network in PyTorch that can do both classification and regression at the same time. Therefore, we will be doing *multi-task learning.* A simple architecture you can try is one that starts with common layers, then branches into classification and regression branches.



- The project report should include a brief summary of the results you have achieved with the methods (it can be a continuation of your Milestone 1 report). Make sure to also discuss the neural network architecture you have designed for this milestone.
- What you are expected to have running by the end of Milestone 2 (this is tentative):

- PCA
- kNN
- PyTorch neural network

- We will run a mini competition where you are supposed to upload your classification and regression predictions from the neural network model. This part is **not** for grading but just a fun exercise. The top performers will earn a chocolate gift from us. We will provide more details about the platform and its usage later in the semester.

Grading

- Each milestone is worth 10% of your grade.
- We will be grading 75% based on the correctness of your implementation and 25% on your report (comprehensiveness, creativity, discussions).

Edits

Here we will keep track of the edits made to this document.

21.10.22:

Under the section "Milestone 1", we have added some clarifications for the scripts you are supposed to complete. We have also added a warning to not edit the data *within* the methods. We have also added the expected results for each dataset. These edits are currently in red.

Fixes

Here we will keep track of the fixes you need to make for your project framework. Sorry about these!

21.10.22:

In `logistic_regression.py`: We have noticed that the comment under the "`fit`" function is incorrect, the labels should not have the shape (N, regression_target_size), they should have the shape (N,).

Please change the comment to:

```
"""
     Trains the model, returns predicted labels for training data.
     Arguments:
          training_data (np.array): training data of shape (N,D)
          training_labels (np.array): regression target of shape (N, )
     Returns:
          pred_labels (np.array): target of shape (N, )
"""
```