



MIDTERM REPORT

Machine Learning and Artificial Learning

Siddhant Mulkikar
Mentor: Sabyasachi Ray

June 22, 2024

Contents

1	Objective	3
2	Introduction	3
3	Categories in Machine Learning	3
3.1	Supervised Learning	3
3.2	Unsupervised Learning	3
4	Supervised Learning Algorithms	4
4.1	Linear Regression	4
4.2	Polynomial Regression	5
4.3	Logistic Regression	5
4.4	Naive Bayes	5
4.5	Git Init	6
4.6	Git Add	6
4.7	Git Remove	6
4.8	Git Commit	6
4.9	Git Checkout	6
4.10	Git Log	7
4.11	Git Head	7
4.12	Graphs	7
4.13	Stats	7
4.14	Report Card	7
4.15	Help	7
5	Working	7
5.1	Upload	7
5.2	Total	7
5.3	Combine	8
5.4	Update	8
5.5	Rank	9
5.6	Git Init	9
5.7	Git Add	9
5.8	Git Remove	9
5.9	Git Commit	9
5.10	Git Checkout	10
5.11	Git Log	10
5.12	Git head	10
5.13	Graphs	10
5.14	Stats	12
5.15	Report Card	12
6	References	13

1 Objective

This midterm report goes through the basics of Machine Learning, followed a dive into the various popular ML algorithms.

2 Introduction

Machine Learning is the study of building mathematical models and algorithms to help understand data. It is often seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions. Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or infeasible to develop conventional algorithms to perform the needed tasks.

3 Categories in Machine Learning

At the most fundamental level, machine learning can be categorized into two main types: supervised learning and unsupervised learning.

3.1 Supervised Learning

Supervised learning involves modelling the relationship between input features and the target variable. The model is trained on a labelled dataset, which means that each training example is paired with the correct target value. It is further subdivided into two categories, regression and classification. Regression has continuous labels, like a real number in a range, while classification has discrete labels i.e. categories.

3.2 Unsupervised Learning

Unsupervised learning involves modelling the underlying structure or distribution in the data in order to learn more about the data. The model is trained on an unlabelled dataset, which means that the model is not provided with the correct target value. These models involve algorithms like clustering, anomaly detection and dimensionality reduction. Clustering involves grouping similar data points together, while dimensionality reduction involves reducing the number of input variables in the dataset, representing the data in a more clearly and anomaly reduction involves detecting outliers in the data.

4 Supervised Learning Algorithms

4.1 Linear Regression

Linear regression is a good starting point for regression models. The simplest linear regression model is fitting a straight line to a dataset, but it can be extended to model more complicated data behaviour. The simplest linear regression model is a straight line fit, which is a model of the form :

$$f_{w,b}(x) = wx + b \quad (1)$$

The ultimate goal of the model is to find a weight vector w and bias b , such that the predicted value of the target variable is as close as possible to the actual value over all training examples.

Given a feature input vector of the i^{th} training example, $x^i = \langle x_1, x_2, \dots, x_m \rangle$, the model predicts the output \hat{y} , which is called the target variable, as :

$$\hat{y}^{(i)} = f_{w,b}(x^i) = w \cdot x^i + b \quad (2)$$

where, $w = \langle w_1, w_2, \dots, w_m \rangle$ is a vector of weights, b is the bias term, both of which are learned by the model during the learning process.

In order to train the model, a cost function is defined, which measures the accuracy of the model's prediction. One of the most popular cost functions for linear regression is Mean squared error (MSE), which measures the average square deviation between the actual and predicted target, values which is defined as:

$$MSE = \frac{1}{m} \sum_{i=1}^n (y^i - \hat{y}^i)^2 = L \quad (3)$$

The model is trained by minimizing the cost function, which is done by using another algorithm like gradient descent.

Gradient Descent

Gradient descent is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient. The algorithm updates the weights vector and bias on each iteration, till it reaches a minima for the cost function.

$$w' = w - \alpha \frac{\partial L(w, b)}{\partial w} \quad (4)$$

$$b' = b - \alpha \frac{\partial L(w, b)}{\partial b} \quad (5)$$

where α is the learning rate, which determines the size of the step taken in the direction of the gradient, and can be tuned to get the best predictions. If the number of training examples are m and the number of features are n , then the gradient of the cost function with respect to the bias and j^{th} weight is given by:

$$\frac{\partial L(w, b)}{\partial b} = \frac{2}{m} \sum_{i=1}^m (\hat{y}^i - y^i) \quad (6)$$

$$\frac{\partial L(w_j, b)}{\partial w_j} = \frac{2}{m} \sum_{i=1}^m x_j^i (\hat{y}^i - y^i) \quad (7)$$

However, gradient descent can give inaccurate predictions if the algorithm doesn't reach the minima, i.e. it diverges from the minima. The convergence of gradient descent into a minima is determined by the learning rate α .

Learning rate (α)

Learning rate is a parameter that determines the size of the step taken in each iteration of gradient descent.

4.2 Polynomial Regression

Conceptually, polynomial regression is pretty similar to linear regression. The difference is instead of fitting a straight line, a polynomial curve is fitted to the data.

Lets consider an univariate polynomial regression model ($n = 1$), for simplicity, of degree d , with the weights being $w = \langle w_1, w_2, \dots, w_d \rangle$:

$$f_{w,b}(x) = w_1x + w_2x^2 + \dots + w_dx^d + b \quad (8)$$

This can be thought of a multivariate linear regression model of d features, where the features are not independent, but rather are powers of the one input feature x , i.e. $x_i = x^i$.

4.3 Logistic Regression

Despite the name suggesting regression, logistic regression is usually used as a binary classification model, i.e. it has two target variable values - 1 and 0, or True and False, also known as classes. Given an input feature vector $x = \langle x_1, x_2, \dots, x_m \rangle$, the model gives the probability of the target variable being 1, as:

$$f_{w,b}(x) = \sigma(w \cdot x + b) = P(\hat{y} = 1|x) \quad (9)$$

where σ is the sigmoid function, defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (10)$$

The sigmoid function maps any real number to the range (0,1), which is useful for binary classification, as it can be interpreted as the probability of the target variable being 1.

If the input vectors and their labels are represented as points in the vector space, the hyperplane $z = w \cdot x + b$, divides the vector space into two regions, i.e. two classes, henceforth classifying the dataset.

While one might think of using the MSE cost function for logistic regression, it turns out that the loss function is not convex, which means that gradient descent might not converge to the minima. Hence a different cost function known as the logistic loss function is used, which is defined as:

$$L(f_{w,b}(x), y) = \frac{1}{m} \sum (-y \log(f_{w,b}(x)) - (1 - y) \log(1 - f_{w,b}(x))) \quad (11)$$

Gradient descent can be used here to train the model in the similar way as linear regression. The gradients for the j^{th} weight and the bias are given by:

$$\frac{\partial L(w, b)}{\partial b} = \frac{1}{m} \sum_{i=1}^m (f_{w,b}(x) - y^i) \quad (12)$$

$$\frac{\partial L(w_j, b)}{\partial w_j} = \frac{1}{m} \sum_{i=1}^m x_j (f_{w,b}(x) - y^i) \quad (13)$$

4.4 Naive Bayes

Naive Bayes models are a set of supervised learning algorithms based on the Bayes theorem, which is:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (14)$$

Lets consider a dataset with discrete features at first. The probability of the desired class y for a single input feature x_i , is given by:

$$P(y|x_i) = \frac{P(x_i|y)P(y)}{P(x_i)} \quad (15)$$

Considering the dataset has n features, the model calculates the probability of the desired class y for the input feature vector $x = \langle x_1, x_2, \dots, x_n \rangle$, as:

$$P(y|x_1, x_2, \dots, x_n) = \frac{P(x_1, x_2, \dots, x_n|y)P(y)}{P(x_1, x_2, \dots, x_n)}$$
$$P(y|x_1, x_2, \dots, x_n) = \frac{P(y|x_1)P(y|x_2) \dots P(y|x_n)P(y)}{P(x_1)P(x_2) \dots P(x_n)}$$

$$P(y|x_1, x_2, \dots, x_n) \propto P(y) \prod_{i=1}^n P(x_i|y) \quad (16)$$

If the problem is of binary classification, with the classes being y_1 and y_2 , we can decide between the two classes by taking the ratio of both the probabilities i.e. $\frac{P(y_1|x_1, x_2, \dots, x_n)}{P(y_2|x_1, x_2, \dots, x_n)}$ and check if this ratio is greater or lesser than 1.

More generally for multi-class classification, the class with the highest probability \hat{y} is chosen as the predicted class by:

$$\hat{y} = \operatorname{argmax}_y P(y) \prod_{i=1}^n P(x_i|y) \quad (17)$$

Gaussian Naive Bayes

If instead of discrete features, our model has continuous features, the Gaussian Naive Bayes model can be used. The only difference is the way in which the probabilities are calculated. While it is quite self-evident in the case of discrete features, it is not that obvious in the case of continuous features.

The model assumes that the dataset has a Gaussian distribution for each of the classes with no covariance between features (dimensions) i.e. every feature is independent from the other. The model then calculates the standard deviation σ_y and μ_y for the desired class y , after which the probability for each feature is calculated as:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right) \quad (18)$$

The steps after this are the same as the discrete feature case.

4.5 Git Init

Run the following command to initialize a remote git repository.

```
bash submission.sh git_init <remote-repo-path>
```

4.6 Git Add

Run the following command to add files from the current directory to the staging area.

```
bash submission.sh git_add
```

4.7 Git Remove

Run the following command to remove files from the staging area.

```
bash submission.sh git_remove
```

4.8 Git Commit

Run the following command to commit the files in the staging area to the remote repository.

```
bash submission.sh git_commit -m "commit message"
```

4.9 Git Checkout

Run the following command to checkout a particular commit from the remote repository.

```
bash submission.sh git_checkout <commit-id>
bash submission.sh git_checkout -m <commit-message>
```

4.10 Git Log

Run the following command to get the commit history log.

```
bash submission.sh git_log
```

4.11 Git Head

Run the following command to get the head of the git repository, i.e. the current version of directory.

```
bash submission.sh git_head
```

4.12 Graphs

Run the following command to get the graphs for a particular exam or all exams.

```
bash submission.sh graphs
```

4.13 Stats

Run the following command to get the stats for a particular exam or `main.csv`.

```
bash submission.sh stats
```

4.14 Report Card

Run the following command to get the report card/s for a particular student or all students.

```
bash submission.sh report_card
```

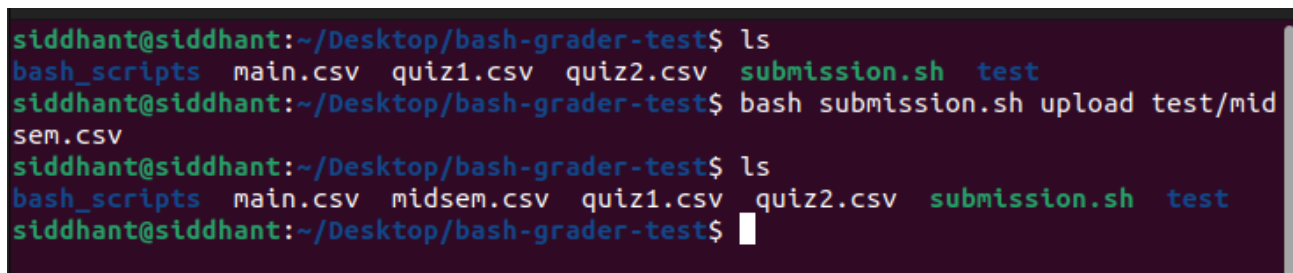
4.15 Help

Run the following command to get the help page.

```
bash submission.sh help
```

5 Working

5.1 Upload

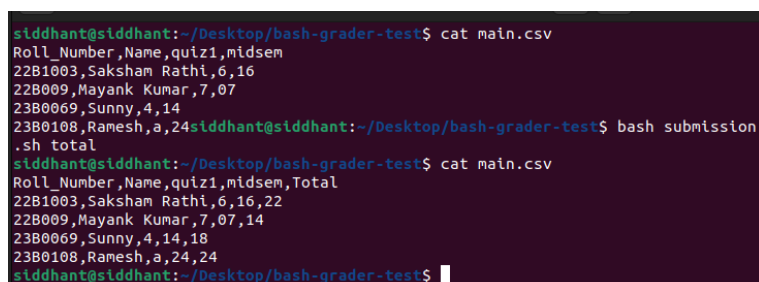


```
siddhant@siddhant:~/Desktop/bash-grader-test$ ls
bash_scripts  main.csv  quiz1.csv  quiz2.csv  submission.sh  test
siddhant@siddhant:~/Desktop/bash-grader-test$ bash submission.sh upload test/midsem.csv
siddhant@siddhant:~/Desktop/bash-grader-test$ ls
bash_scripts  main.csv  midsem.csv  quiz1.csv  quiz2.csv  submission.sh  test
siddhant@siddhant:~/Desktop/bash-grader-test$
```

In the above example, the file `quiz1.csv` is uploaded i.e. copied into the current directory from the filepath given as argument by the user. The script checks if the file exists and if it is a csv file. If it is, only then is the file copied into the current directory.

Script files : `upload.sh`

5.2 Total



```
siddhant@siddhant:~/Desktop/bash-grader-test$ cat main.csv
Roll_Number,Name,quiz1,midsem
22B1003,Saksham Rath,6,16
22B009,Mayank Kumar,7,07
23B0069,Sunny,4,14
23B0108,Ramesh,a,24
siddhant@siddhant:~/Desktop/bash-grader-test$ bash submission.sh total
siddhant@siddhant:~/Desktop/bash-grader-test$ cat main.csv
Roll_Number,Name,quiz1,midsem,Total
22B1003,Saksham Rath,6,16,22
22B009,Mayank Kumar,7,07,14
23B0069,Sunny,4,14,18
23B0108,Ramesh,a,24,24
siddhant@siddhant:~/Desktop/bash-grader-test$
```

In the above example, the script totals the marks of each student over all exams and adds a new column **Total** to the **main.csv** file. If the marks column has **a** as its entry, the script skips that row while totaling.

Script files : **total.awk**

5.3 Combine

In the below example, the script combines all the csv files in the current directory into a single **main.csv** file. The script checks if the file is a csv file and if it is not the **main.csv** file.

main.csv is constructed from scratch in the following steps:

1. First, an array of all unique roll numbers and names is created.
2. Then, based on the exam files, a header is created with the roll numbers and names.
3. A mesh of **a**'s is created with the dimensions of the header.
4. Then the script iterates over all exam files and fills in the marks of each student in the mesh, whenever they are found. This ensures that anyone who is not present in a particular exam file is marked as **a**(absent) in **main.csv** for that particular exam.

```
main.csv > data
1 Roll Number,Name,midsem,quiz1
2 23B0901,Kaushal Vijayvergiya,a,11.38
3 23B0902,Arya Suwalka,4.91,20
4 23B0903,Abhi Jain,6.4,9.28
5 23B0904,Prakhar Jain,7.39,a
6 23B0905,Sagnik Nandi,7.23,8.92
7 23B0906,Velagala Deeraj Sathvik Reddy,6.44,14.46
8 23B0907,Hari Shankar Karthik,a,13.3
9 23B0908,Raghav Goyal,8.37,7.53
10 23B0909,Varri Shyam Sri Vardhan,6.05,a
11 23B0910,Sonal Kumari,4.41,9.7
```

(a) Before combine

```
main.csv > data
1 Roll Number,Name,endsem,midsem,quiz1,quiz2
2 23B0901,Kaushal Vijayvergiya,a,a,11.38,14.47
3 23B0902,Arya Suwalka,4.57,4.91,20,a
4 23B0903,Abhi Jain,0.91,6.4,9.28,9.7
5 23B0904,Prakhar Jain,8.72,7.39,a,14.03
6 23B0905,Sagnik Nandi,6.31,7.23,8.92,11.74
7 23B0906,Velagala Deeraj Sathvik Reddy,5.63,6.44,14.46,7.88
8 23B0907,Hari Shankar Karthik,6.4,a,13.3,10.26
9 23B0908,Raghav Goyal,6.08,8.37,7.53,4.31
10 23B0909,Varri Shyam Sri Vardhan,8.15,6.05,a,11.72
11 23B0910,Sonal Kumari,4.78,4.41,9.7,a
```

(b) After combine

Script files : **combine.sh**, **combine.awk**

5.4 Update

```
quiz1.csv > data
1 Roll Number,Name,Marks
57 23B0956,Siddhant Mulikar,5.85
58 23B0957,Kunumalla Gautam Siddharth,14.14
59 23B0958,Satyam Sinoliya,17.14
60 23B0960,J Adarsh,14.82
61 23B0961,Arijit Mandal,12.47
62 23B0962,Banothu Sonusree,7.1
63 23B0963,Komrelly Snigdha Reddy,12.79
64 23B0964,Seemala Varsha,6.33
65 23B0965,Nakka Mahathi,12.62
```

(a) quiz1.csv before update

```
quiz1.csv > data
1 Roll Number,Name,Marks
57 23B0956,Siddhant Mulikar,10.85
58 23B0957,Kunumalla Gautam Siddharth,14.14
59 23B0958,Satyam Sinoliya,17.14
60 23B0960,J Adarsh,14.82
61 23B0961,Arijit Mandal,12.47
62 23B0962,Banothu Sonusree,7.1
63 23B0963,Komrelly Snigdha Reddy,12.79
64 23B0964,Seemala Varsha,6.33
65 23B0965,Nakka Mahathi,12.62
```

(b) quiz2.csv after update

```
main.csv > data
1 Roll Number,Name,endsem,midsem,quiz1,quiz2>Total
57 23B0956,Siddhant Mulikar,6.61,4.56,5.85,33.01,50.03
58 23B0957,Kunumalla Gautam Siddharth,9.98,7.76,14.14,15.84,47.72
59 23B0958,Satyam Sinoliya,7.62,6.38,17.14,9.93,41.07
60 23B0960,J Adarsh,4.88,5.48,14.82,7.79,32.97
61 23B0961,Arijit Mandal,6.23,10.12,47.10,73.39.43
62 23B0962,Banothu Sonusree,7.0,7.25,7.1,a,21.35
63 23B0963,Komrelly Snigdha Reddy,1.54,3.34,12.79,8.4,26.07
64 23B0964,Seemala Varsha,6.05,6.33,6.33,a,18.71
65 23B0965,Nakka Mahathi,5.53,1.62,12.62,12.04,31.81
```

(a) main.csv before update

```
main.csv > data
1 Roll Number,Name,endsem,midsem,quiz1,quiz2>Total
57 23B0956,Siddhant Mulikar,6.61,4.56,10.85,33.01,55.03
58 23B0957,Kunumalla Gautam Siddharth,9.98,7.76,14.14,15.84,47.72
59 23B0958,Satyam Sinoliya,7.62,6.38,17.14,9.93,41.07
60 23B0960,J Adarsh,4.88,5.48,14.82,7.79,32.97
61 23B0961,Arijit Mandal,6.23,10.12,47.10,73.39.43
62 23B0962,Banothu Sonusree,7.0,7.25,7.1,a,21.35
63 23B0963,Komrelly Snigdha Reddy,1.54,3.34,12.79,8.4,26.07
64 23B0964,Seemala Varsha,6.05,6.33,6.33,a,18.71
65 23B0965,Nakka Mahathi,5.53,1.62,12.62,12.04,31.81
```

(b) main.csv after update

In the above example, the script updates the marks of **23B0956** in **quiz1.csv** and **main.csv**. The script first asks the user for the roll number and name of the student whose marks are to be updated. It then checks if the student's name matches with the name paired up with the roll number given by the user. If the names do not match, the script confirms with the user with a prompt for the correct name. It then checks if the student is present in the **main.csv** file and if the exam file exists. If the student is present in the **main.csv** file, the script updates the marks of the student in **main.csv** by calling **update_main.awk** and the exam file by calling **update_exam.awk**.

If the student's record does not exist in the `main.csv` file, the script prompts the user to confirm if they want to add the student's record to an exam file and `main.csv`. If the user confirms, the script adds the student's record to the exam file and `main.csv`.

It also asks the user if they want to update the marks of the student in some other exam and repeats the same process. If the user does not want to update the marks of the student in any other exam, the script checks if `main.csv` had been totaled before. If it was, then the script calls `total.awk` to correct the total, as the marks in `main.csv` have been updated.

Script files : `update.sh`, `update_exam.awk`, `update_main.awk`, `total.awk`

5.5 Rank

This command takes the examname as a command line argument and ranks the students based on the marks in that exam. If the user enters `main` as the examname, the script ranks the students based on the total marks in `main.csv`. The ranked marklists are sorted according to rank and are stored in the folder `ranked_marklists` in the current directory.

Script files : `rank.sh`

5.6 Git Init

This command initializes a remote git repository at the path given by the user as a command line argument. It also checks if the path given is a valid path. If a remote repository is already initialized for the current directory, the script prompts the user to confirm if they want to reinitialize the repository. If the user confirms, the script reinitializes a fresh remote repository at the path provided by the user.

It also creates a hidden folder `.gitrepo` in the current directory. This folder contains a hidden file `.gitreponame` which stores the name of the remote repository and `.git_log` which records the commit history.

Apart from initialization, the script also copies all the csv files in the current directory at the time of initialization to a hidden folder `.ogfiles` in the remote repository. The `.ogfiles` folder contains the first version of every file which was ever committed. This serves as a reference point for all files, for my **diff patch** system for `git checkout` to work.

Script files : `git_init.sh`

5.7 Git Add

This command first checks if a remote repository has been initialized for the current directory. If it has not, the script prompts the user to initialize a remote repository first.

This command then prompts the user to enter all the files, which they want to add to the staging area, separated by spaces. It then copies all the files to the staging area in the remote repository. If any particular file doesn't exist in the current directory, it gives an error message.

Script files : `git_add.sh`

5.8 Git Remove

This command first checks if a remote repository has been initialized for the current directory. If it has not, the script prompts the user to initialize a remote repository first.

It then prompts the user to enter all the files, which they want to remove from the staging area, separated by spaces. It then deletes all the files from the staging area in the remote repository. If any particular file doesn't exist in the staging area, it gives an error message.

Script files : `git_remove.sh`

5.9 Git Commit

The command first checks if a remote repository has been initialized for the current directory. If it has not, the script prompts the user to initialize a remote repository first.

It then checks what all files in the staging area are modified with respect to the **first version of the file** in the `.ogfiles` folder. It also checks for any new files which were created and committed for the first time. It

then updates the `.git_log` file with these modified and added files, along with the commit message and the files which were modified or added in this commit, which is uniquely identified by the commit message, given by the user and the 16 character hash commit id, generated randomly using the `uuidgen` command. A folder with the commit-id as its name is created in the `commits` folder of the remote repository.

If any file in the staging area is not present in the `.ogfiles` folder, it adds it to the `.ogfiles` folder. It then copies the diff of the file from the `.ogfiles` folder and the file in the staging area into the commit-id folder.

It also gets the list of all csv files in the current repository at the time of the commit and stores their name in a hidden `.files` file in the commit-id folder. This provides a snapshot of all the files in the current repository at the time of the commit, which is useful at the time of `git_checkout`. The stage is deleted after the commit is made.

Script files : `git_commit.sh`

5.10 Git Checkout

The command first checks if a remote repository has been initialized for the current directory. If it has not, the script prompts the user to initialize a remote repository first.

The user can checkout a particular commit by entering the commit-id (even entering the first few characters of the commit-id works too, if they are unique and not shared with any other commit-id) or the commit message. The script then parses and checks for matches to the commit-id or the commit message in the `.git_log` file. If it finds a match, it goes to the commit-id folder.// There, it gets the snapshot of the current repository at the time of the commit from the `.files` file. It also gets the list of patch files in the commit-id folder.

It deletes all the `.csv` files from the current directory and then iterates over all files in the snapshot. If the patch file exists for a particular file, it applies `patch -b` command to this patch file and the corresponding file in the `.ogfiles` folder, which gives back the version of the file at that commit as `<filename>.csv` and also the original file in the `.ogfiles` as `<filename>.csv.orig`. The `.csv` file is then copied from the `.ogfiles` folder to the current directory and the `.csv.orig` file is renamed to `.csv` file, hence restoring the original file in the `.ogfiles` folder.

Script files : `git_checkout.sh`

5.11 Git Log

The command first checks if the remote repository has been initialized for the current directory. If it has not, the script prompts the user to initialize a remote repository first. The script then reads the `.git_log` file and prints the commit history log in a readable format, with the commit-id, commit-message and time of commit. It also displays any new files created in the current directory, when they are committed for the first time, by the name **New files added:**, and if any files were modified from the time when the time the remote repository was initialized, it also shows them under the name **Files modified:**

While this command only displays the commit history log, it is actually the `git_commit` command which actually updates the `.git_log` file, at every commit.

Script files : `git_log.sh`

5.12 Git head

This command shows the present head of the current directory.

Script files : `git_head.sh`

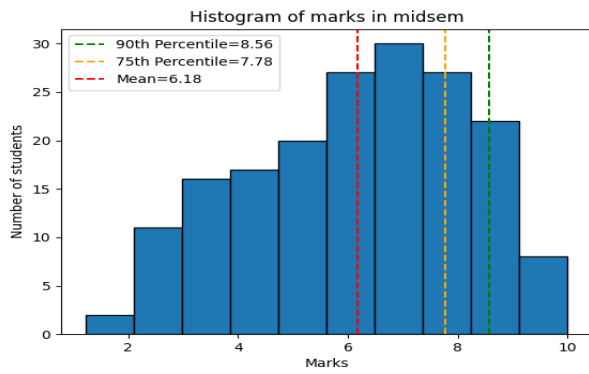
5.13 Graphs

This command calls the `graphs.py` python script, which has a interface where users can choose between a histogram and a scatter plot. Then, the user can choose between a particular exam or `main` (for `main`, it considers the total marks for mark generation). It then calls the corresponding function from `graph_functions.py` to plot the graph, if the csv file of the exam exists.

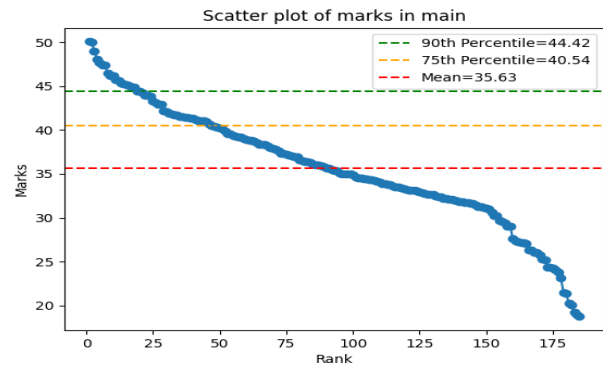
The graphs also show the demarcation for 90th percentile, 75th percentile and mean marks.

Modules used for graph generation: os, csv, matplotlib

Script files: graphs.py, graph_functions.py



(a) Histogram for midsem marks



(b) Scatter plot for main.csv

5.14 Stats

This command calls the `stats.py` python script, which has a interface where users can choose between a particular exam or `main` (for `main`, it considers the total marks for mark generation). It then displays the stats for that particular exam or `main`, if the csv file for the exam exists.

The stats include the mean, median, mode, standard deviation, 25th percentile, 75th percentile, 90th percentile and maximum marks.

All these stats are calculated by calling their respective functions from `stats_functions.py`.

Modules used for stats generation: `os`, `csv`, `statistics`

Script files: `stats.py`, `stats_functions.py`

```
siddhant@siddhant:~/Desktop/bash-grader$ bash submission.sh stats
Enter the exam name : quiz1
-----
Statistics for quiz1
Mean score: 11.86
Median score: 11.84
Mode: 20.0
Standard deviation: 3.74
marks for 25th Percentile: 9.1
Marks for 75th Percentile: 14.48
Marks for 90th Percentile: 16.93
Maximum marks: 20.0
siddhant@siddhant:~/Desktop/bash-grader$
```

5.15 Report Card

This command calls the `report_card.py` python script, which has a interface where users can choose between a particular student or all students. It then displays the report card for that particular student or saves the pdf of all student report cards in the `report_cards` folder in the current directory.

It gets the grades and percentile rubrics for the grades from the instructor by calling functions from the `grade.py` script. It then gets all the exams conducted and the maximum marks scored in each of these exams. Student marks for all students are also fetched by using the module `csv`.

It then calculates the weighted percentage of each student by using the formula:

$$W.P. = \frac{\sum_{i=1}^n M_i \times M_{max_i}}{\sum_{i=1}^n M_{max_i}^2} \times 100 \quad (19)$$

where M_i is the marks scored by the student in exam i , M_{max_i} is the maximum marks scored in exam i and n is the number of exams.

This is considering the maximum marks scored in each exam as the total marks for that exam. This ensures that a person with more marks in an exam with a greater weightage gets a higher weighted percentage.

The script then calculates the grade of each student by using the weighted percentage and the rubrics provided by the instructor. Then the percentiles of each student are calculated with this weighted percentage using the `scipy` module. Then, the report card is generated from a template report card and text is put on the template using the `PIL` module and then converted to pdf form for all students, while it's only displayed for one person.



PROGRESS REPORT

Student Name Siddhant Mulkikar

Roll No. 23B0956

Grade C

EXAM	MARKS	MAX MARKS
midsem	5.0	10.0
quiz1	5.85	20.0
quiz2	18.01	20.0

Modules used for report card generation: os, csv, statistics, scipy, PIL

Script files: report_card.py, grade.py

6 References

- [1] Bash manual: <https://www.gnu.org/software/bash/manual/bash.html>
- [2] Awk manual: <https://www.gnu.org/software/gawk/manual/gawk.html>
- [3] Python manual: <https://docs.python.org/3/>
- [4] Matplotlib manual: <https://matplotlib.org/>
- [5] Scipy module: <https://www.scipy.org/>
- [6] PIL module: <https://pillow.readthedocs.io/en/stable/>
- [7] csv module: <https://docs.python.org/3/library/csv.html>
- [8] os module: <https://docs.python.org/3/library/os.html>