

EXAMLY ROUND 3 – PROJECT

DIFFICULTY OF THE QUESTION

PROBLEM STATEMENT



Derive an algorithm to find difficulty of a question given the following details

- 1) Question type -- MCQ, Fillup, Programming, Match the following
- 2) Manually assigned difficulty -- Easy, Medium, Hard
- 3) Total number of students who have attended this question
 - a) Time taken by each student to answer this question
 - b) Number of times the answer was changed if it is MCQ type question
 - c) Number of times the program was compiled if it is programming question
 - d) Number of hints used
 - e) Programming language used if it is programming question (c, c++ , java etc)
- 4) Feedback given for this question by other students
- 5) Total number of students who have answered it right
- 6) Total number of students who have answered it wrong
- 7) Total number of students who have answered it partially correct
- 8) Maximum marks allocated for this question

The main goal of this exercise is to come up with a scalable solution. The total number of questions will be in the range of 100000 to 1 million. Total number of students will be in the range of 1 million to 10 million. The total number of question attempts in the worst case will be 30 million.

Assume all possible corner cases with respect to the question and question attempt data (Eg - Question was not attempted by anyone, the same student has attempted the same question multiple times with increasing or decreasing accuracy)

The following is expected as part of the solution

- 1) Algorithm as pseudocode or formula
- 2) Explanation and proof for the Algorithm in plain easy to understand english
- 3) Create the input data on your own for the volume required in the performance metrics specified below
- 4) Code in github as a standalone API which will accept the input data as file and print out the difficulty (in programming language of your choice)
- 5) Performance run metrics for
 - a) 100 question 100 attempts
 - b) 10000 question 10000 attempts
 - c) 100000 question for 1 million attempts

This project is to find the difficulty level of the question for various question types considering the factor like total number of student attended, average time taken of all students, the number of times answer was changes, the number of times compiled in case of program, number of hints used, question answered right, wrong, partially correct in case of program, marks allocated.

PREREQUISITES

To run this project the system should be installed with Flask (for API), latest version of Python (for ML).

PROJECT STRUCTURE

The project has four main parts:

- i) Difficulty_Calculator.py
- ii) app.py
- iii) templates
- iv) Dataset for training the model

Difficulty_Calculator.py – To Calculate the difficulty as easy, medium, hard for the given dataset and output the training set for the model.

app.py – To train the model by taking the training set outputted by Difficulty_Calculator.py. It also predicts the difficulty of the dataset file submitted from API using KNN algorithm trained model and outputs the result.

templates - This folder contains the HTML template to allow user to upload csv file for which difficulty is to be predicted.

Dataset – An excel file and csv file containing 4096 rows and 11 columns generated as per the conditions and details provided in the problem statement.

DATASET

To train the model data set is required, the data set is generated for four types of questions including 11 fields as mentioned in the problem statement and for each question type 1024 rows of data are used satisfying the performance metric conditions and to make the model well trained to work efficiently.

ALGORITHM OF Difficulty_Calculator.py

The Difficulty_Calculator file is inputted with the dataset csv file and using `csv.reader()` its stores as a list of list.

The csv file stored in a list of list is in the form of strings so its type casted to float and stored.

To store the Difficulty along with fields specified in an excel sheet `xlsxwriter` is imported and a new worksheet is created in a new workbook.

The code traverses through the list of list which contains the values of csv file inputted in the form of float, along with three variables easy, medium and hard.


In case of the time taken to answer for MCSs, Fill ups and match, if the average time taken is below 60seconds easy variable is incremented, if time taken is between 60 and 90 seconds medium is incremented else hard is incremented. For programs same happens in terms of minutes.

The number of times options changed and number of times compiled for MCQs and Programs respectively also carries the same way as if option is changed less than 2 times and program is compiled less than 2 times then the question is easy, if the options is changed or program is compiled between 2 to 6 times question is medium and else the question is hard (variable increments).

The number of hints used is also taken into consideration in case of programming and the based on the certain condition the hard, medium and east variables are incremented.

Regarding the programming language the C is easy, C++ is medium and Java is taken as hard.

Further based on the student attended and student correct count the variables are incremented, if ratio of correct answer and student attended is $> 66\%$ then easy, if less than 33% then hard else medium.

 *Difficulty_Calculator.py - C:\Users\admin\Desktop\Examly_R3\Difficulty_Calculator.py (3.7.4)*

File Edit Format Run Options Window Help

```
temp = []
if(dataset[i][0] == 0 or dataset[i][0] == 1 or dataset[i][0] == 3):
    if(dataset[i][7] != 0 and dataset[i][1]/dataset[i][7] > 0.6):
        easy += 1
    elif(dataset[i][7] == 0 or dataset[i][1]/dataset[i][7] < 0.3):
        hard += 1
    else:
        medium += 1
if(dataset[i][2] <= 60):
    easy += 1
elif(dataset[i][2] > 90):
    hard += 1
else:
    medium += 1
if(dataset[i][3] == 0):
    easy += 1
elif(dataset[i][3]>3):
    hard += 1
else:
    medium += 1
if(dataset[i][6] == 0):
    easy += 1
elif(dataset[i][6] == 1):
    medium += 1
else:
    hard += 1
if(dataset[i][10] == 2):
    easy += 1
elif(dataset[i][10] == 4):
    medium += 1
else:
    hard += 1
else:
    if(dataset[i][7] != 0 and dataset[i][1]/dataset[i][7] > 0.6):
        easy += 1
    elif(dataset[i][7] == 0 or dataset[i][1]/dataset[i][7] < 0.3):
        hard += 1
    else:
        medium += 1
    if(dataset[i][2] <= 60):
        easy += 1
    elif(dataset[i][2] > 90):
        hard += 1
    else:
        medium += 1
    if(dataset[i][4] <= 1):
        easy += 1
    elif(dataset[i][4]>2):
```

Based on the above mentioned criteria the variable having the maximum count is the difficulty level of the question for the dataset inputted.

The difficulty levels for training datasets are appended as a last field and the excel file is outputted.

The above mentioned process can be even carried out to predict the difficulty since the dataset to train the model is not available and it is not that much efficient it is considered only for calculating the difficulty level for the dataset.

ALGORITHM OF app.py

The inputted training set csv file is converted into list of list.

The list of list values are converted to float values excluding the last value since it acts as a label to classify the data by calling function `str_column_to_float`.

The last value of list of list is passed to an another function named `str_column_to_int` and the labels for classification is found.

0 means Easy, 1 means Hard and 2 means Medium.

The inputted csv file via web api is reverted to this python page and the file is converted to list of list and stored in form of float.

The `k` (number of nearest neighbors) value is given as 10.

The `predict_classification` function is called with the actual parameters as training dataset, predict dataset and `k`.

The `get_neighbours` function is called from the `predict_classification` function and the nearest `k` neighbours are found by calculating the Euclidean distance between the datasets and sorting them.

The `get_neighbors` function returns the `k` neighbor dataset values to the `predict_classification` function and the `predict_classification` function returns the max label(Hard, Medium , Easy) value.

The above three steps are carried out for all the rows in a dataset inputted via web-api.

The label values predicted is appended along with the csv file imported and converted into excel and then stored in device in the name Result.xlsx via an import xlsx reader.

```
# Calculate the Euclidean distance between two vectors
def euclidean_distance(row1, row2):
    distance = 0.0
    for i in range(len(row1)-1):
        distance += (row1[i] - row2[i])**2
    return sqrt(distance)

# Locate the most similar neighbors
def get_neighbors(train, test_row, num_neighbors):
    distances = list()
    for train_row in train:
        dist = euclidean_distance(test_row, train_row)
        distances.append((train_row, dist))
    distances.sort(key=lambda tup: tup[1])
    neighbors = list()
    for i in range(num_neighbors):
        neighbors.append(distances[i][0])
    return neighbors

# Make a prediction with neighbors
def predict_classification(train, test_row, num_neighbors):
    neighbors = get_neighbors(train, test_row, num_neighbors)
    output_values = [row[-1] for row in neighbors]
    prediction = max(set(output_values), key=output_values.count)
    return prediction

# Make a prediction with KNN
filename = 'training_set_csv.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
    str_column_to_float(dataset, i)

# define model parameter
num_neighbors = 10
@app.route('/')
def home():
    return render_template('index.html')
```

ALGORITHM OF TEMPLATES

The templates folder contain two files html and css which acts as a web-api through which the csv file to predict difficulty is uploaded.

The work is carried out in Anaconda and Spyder and Anaconda prompt is used to execute the flask commands to set up the flask environment and to run the python files.

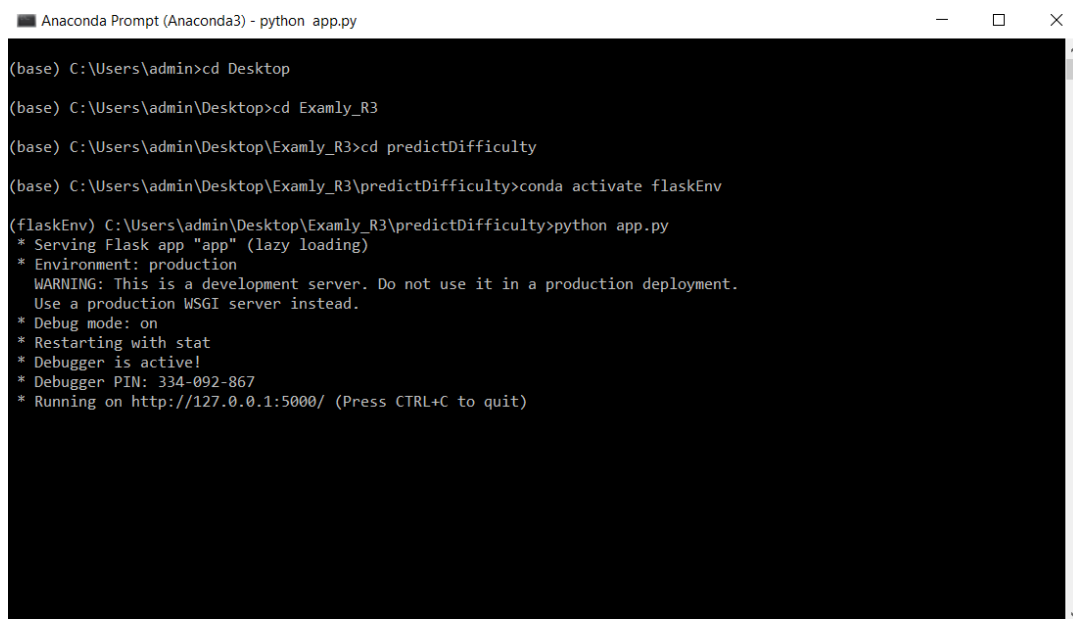
RUNNING THE PROJECT

In the Anaconda Prompt ensure that you are in a project home directory to create and install the flask environment type `conda create -n flask pip flask`.

Activate the flask environment using `conda activate flaskEnv`.

Run the `app.py` in the flask environment.

By default flask will run on port:5000, Navigate to URL <http://localhost:5000>.

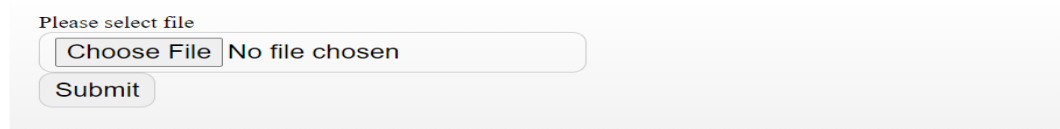


```
Anaconda Prompt (Anaconda3) - python app.py
(base) C:\Users\admin>cd Desktop
(base) C:\Users\admin\Desktop>cd Examly_R3
(base) C:\Users\admin\Desktop\Examly_R3>cd predictDifficulty
(base) C:\Users\admin\Desktop\Examly_R3\predictDifficulty>conda activate flaskEnv
(flaskEnv) C:\Users\admin\Desktop\Examly_R3\predictDifficulty>python app.py
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger PIN: 334-092-867
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

The web-api will be visible as:

Question Difficulty Predictor

You can select the file (csv) and click Submit button



Please select file

No file chosen