Exploring the Supervised Models to Automatically Predict Student Success in Courses based on their Academic Profile

Ali Reza Ibrahimzada[1]

Department of Computer Science and Engineering
2019 - 2020 Academic Year
CS 350 - Database Systems Project Report
Istanbul Sehir University, Istanbul / Turkey
aliibrahimzada@std.sehir.edu.tr
Student Number: 218012083

Supervisor: Dr. Ali Cakmak
alicakmak@sehir.edu.tr

May 29, 2020

Abstract. Based on their skills and interests, students' success in courses may differ greatly. Predicting student success in courses before they take them may be important. For instance, students may choose elective courses that they are likely to pass with good grades. Besides, instructors may have an idea about the expected success of students in a class, and may restructure the course organization accordingly. Instructors can also follow their students' performance and track their graduation progress. In this project, we implement a web-based application interface for students and instructors to process and manage analysis efficiently. An efficient and robust Database Management System is implemented for better management and handling of data.

Keywords: Supervised Learning · Database Management System · Educational Data Mining · Student Success Estimation · Collaborative Filtering

1 Introduction

In colleges, students elect courses based on a variety of motivations. Some may choose a course because they are interested in the covered material, while some others may take a course just because he/she likes the way the course instructor teaches. More often, college students tend to choose courses in which they expect to get good grades, whenever they are given a chance in the form of elective courses. Hence, guiding students about their expected future course performance may allow them to make informed choices. In addition, such guidance may be invaluable for academic advisors and instructors as well. Academic advisors may

suggest certain courses to their students in a personalized manner. Moreover, instructors may design the content of a course according to the academic level of the current set of students in the class at the very beginning of the semester, before it gets late in the middle of the semester. In addition, potential struggling and low-performing students may be identified at an early stage, and these students may be offered additional help in the form of extra tutorial and practice sessions. Hence, providing automated means of estimating future course grades of students may greatly enhance the learning experience of students at colleges. Students' expected grade perception for a future course is usually shaped by (i) what they hear from other fellow students about the course, as well as (ii) how well they did in similar courses in the past. Such a reasoning scheme is actually the basis of what is called collaborative filtering. Collaborative filtering is a machine learning technique that is usually employed in recommender systems where users are recommended movies to watch, songs to listen to, books to read, etc. Based on their past ratings of movies, songs, books, etc. The two main steps of collaborative filtering are (a) locating similar users, and (b) computing a weighted sum of ratings for each item that will be recommended to a user. Moreover, we also implement a regression model on the same dataset to predict student success. Precisely, we extract different features from the given dataset and quantify the performance of different learning algorithms.

In this project, we implement a web-based tool for students and instructors to use and track their academic data. Instructors can follow the performance of students and modify their course material accordingly. On the other hand, students can optimize their performance and get extra information on the courses they are going to select. Moreover, they will be able to get a prediction of their possible grade from a specific course at the start of semester before registering for their course. Here, we consider students' past course grades as the "ratings" that they give to the corresponding courses. Hence, the underlying assumption we make is that if a student gets an A from a course, we consider that the student liked the course a lot. Similarly, a grade F is considered to indicate that the student did not like the course at all. By comparing the past grades of students, we find "similar" students. In order to compute similarity, we employ different measures such as Pearson correlation, Euclidean distance, and Jaccard measure. We also employ two different flavors of collaborative filtering, namely, user- and item-based. Item-based collaborative filtering is usually preferred for large and sparse databases (such as shopping web sites like amazon.com) due to its efficiency.

As part of this study, we further enhance collaborative filtering in the context of course grade problems. More specifically, our proposed enhancements are two-folds: (i) among the courses that a student has already taken, we compute the set of outlier courses, and prevent them from contributing to the computation of the student's future grades, (ii) we compute a running average of grades for a student, and make sure that the student is not getting recommendations from students with significantly higher or lower GPAs than him/herself.

We evaluate the proposed techniques on a real data set of around 55,000 course grades obtained from Istanbul Sehir University, where student names and ids are obfuscated. For better security and safety purposes, we use Google authentication to handle user and instructor login activities. Users' data is being stored in a database for better data management, processing and analyzing. MySQL is used as Database Management System and Flask as Web Framework. Once a student agrees to use this tool and provides permission for their academic data, we use different web scraping techniques to access all their academic profiles available on university's database (SAP for Istanbul Sehir University). The details related to all scraped information of users is available in later sections.

2 Application Requirements

1. The tool will allow students to predict their grade from a course before registration
   - Choose specific subjects to make predictions
   - Choose a minimum grade to filter results even more
   - Best and Worst Case Scenarios
   - Rise/Drop in GPA

2. Instructors will be able to analyze the level of their students and make decisions accordingly
   - Average GPA of students taking the course
   - Department ratio of enrolled students

3. Advisors will be able to track students' graduation progress and approve their courses efficiently during registration period
   - Advisee's academic details
   - List of completed courses
   - List of incompleted courses

4. Students will be able to rate their instructors at the end of semester. The ratings will affect predictions for the next semesters
   - Grading policy
   - Teaching method
   - Recommendation

5. Students will be able to rate their courses at the end of semester. The ratings will affect predictions for the next semesters
   - Taking similar courses
   - Course content
   - Recommendation

6. Students will be able to follow their performance during their studies
   - List of required courses for graduation

   - Graduation progress
7. Students can see a timetable of their courses for a specific day
   - Where ?
   - When ?
8. Students can share their experience after using this tool, and it will be shared in the Testimonials section, publicly available for everyone.

3 Data Requirements

This project is mainly related to educational institutions, therefore, the following data will be stored in a database:

- Student
- Instructor
- Department
- Course
- Section
- Time Slot
- Classroom
- Testimonials
- Ratings (Course and Instructor)

For making a better and robust design, we introduce some participation constraints for each piece of data. The following are the constraints assigned to entities in our design:

- Each student has **at most one** advisor, while an instructor can advise **multiple** students at the same time. Besides, it is also possible that a student has **no** advisor, and an instructor **does not** advise any student.

- Students can enroll in **at least** no course, or **at most several** courses in a semester, if and only if that course is offered. On the other hand, each offered course can be taken by **no** student, or **many** students at the same time.

- Each student can have **at least one** and **at most multiple** departments i.e. a student doing double major has more than 1 department. Departments can have a **single** or **many** students at the same time.

- Students may choose to rate **multiple** courses and instructors, or **not** at all. Besides, each instructor or course can be rated by **no** or **many** students at one time. **Only** a single rating regarding a course or an instructor can be made by students in a given semester.

- A student can write their experience from this tool **several** times or **not** at all.

- Instructors can have **at minimum one** and **at most many** departments. i.e. an instructor can be members of Computer Science and Data Science departments at the same time. Similarly, a department can have a **single** or **more** instructors.

- An instructor can choose to teach **multiple** courses, or **no** course at all. While an offered course can be taught by **at least one** and **at most multiple** instructors at one time.

- A **single** section from the course can be offered **if and only if** the course is offered in that semester. Meanwhile, **at least one** and **at most many** sections of a course can be offered. If so, it is possible that multiple time slots or none be available for an open section of a course. i.e. internship courses have no time slots available.

- A section of offered courses can have **at minimum** none and at most **one** classroom available. Similarly, a classroom can be available for **multiple** sections of courses or **no** course during a semester.

- A course can have **at least one** department and **at most multiple** departments, i.e. freshman courses like ENGR 101, MATH 103, ... has more than one department available. Likewise, a department can have **at least one** and **at most multiple** courses.

- A course can be prerequisite to **at least 0**, and **at most multiple** courses.

4 Entity-Relationship Model Diagram

In this section we will discuss Entity Relationship model and why it is an essential part of designing software applications. Like any other software project, customizing the application based on the needs of the business it is built is important. Software engineers need a solid understanding of the requirements which business people are demanding in order to create a good design. Therefore, designing the Entity Relationship model is the first abstract step after acquiring data requirements. Usually this happens when business people meet with DBA/Engineer and discuss all necessary aspects of the application.

Figure 1 shows the Entity Relationship model we implemented for this project. As per MySQL notations, we use rectangles for entities, diamonds for relationships, ovals for attributes and arrows for participation constraints. Moreover, triangles are used to demonstrate hierarchies between different entities. The concept of hierarchies is very close to Inheritance in Object-Oriented Programming.

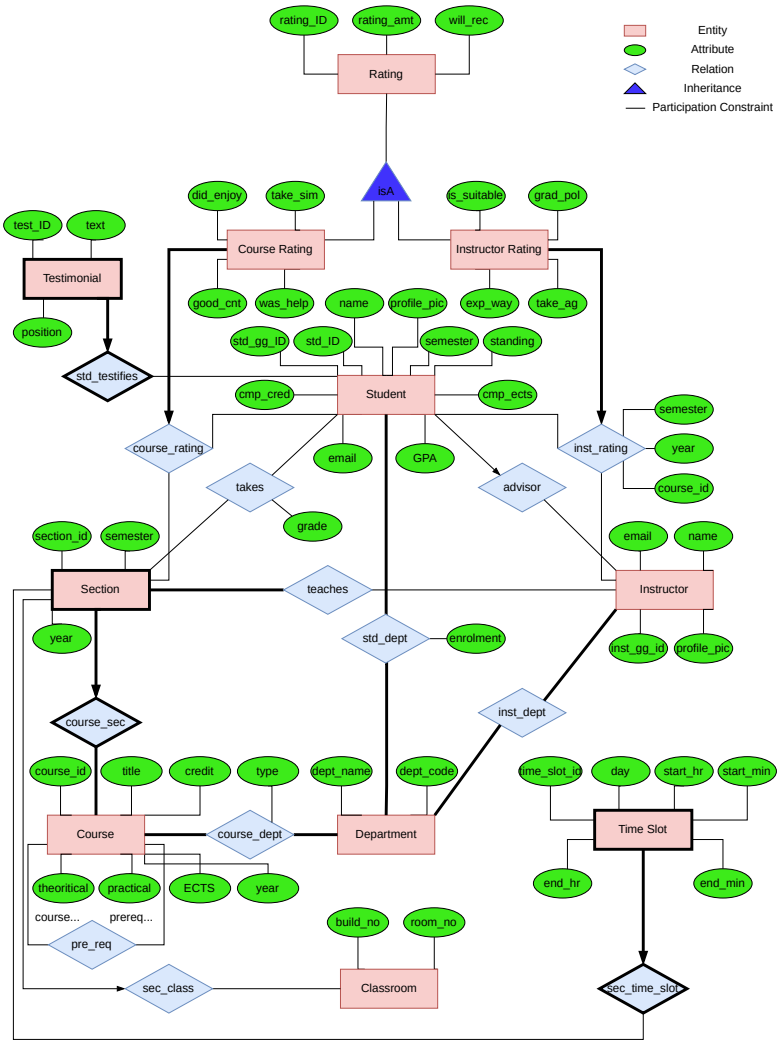

Fig. 1. ER Diagram

5 Logical Model - DDL

```
CREATE TABLE instructor (
  instructor_email VARCHAR(100),
  instructor_google_id VARCHAR(50),
  instructor_name VARCHAR(100),
  instructor_profile_picture VARCHAR(512),
  PRIMARY KEY (instructor_email)
);

CREATE TABLE student (
  student_google_id VARCHAR(50),
  student_id CHAR(9),
  student_name VARCHAR(100),
  student_email VARCHAR(100),
  student_profile_picture VARCHAR(512),
  student_semester TINYINT UNSIGNED,
  standing VARCHAR(15),
  completed_credits SMALLINT UNSIGNED,
  completed_ects SMALLINT UNSIGNED,
  gpa DECIMAL(3, 2),
  advisor VARCHAR(100),
  PRIMARY KEY (student_google_id),
  FOREIGN KEY (advisor) REFERENCES instructor(instructor_email)
);

CREATE TABLE testimonial (
  testimonial_ID INTEGER UNSIGNED AUTO_INCREMENT,
  testimonial_text TINYTEXT,
  testifier_position VARCHAR(50),
  student_google_id VARCHAR(50),
  PRIMARY KEY (testimonial_ID),
  FOREIGN KEY (student_google_id) REFERENCES student(student_google_id)
  ON DELETE CASCADE
);

CREATE TABLE instructor_rating (
  rating_id INTEGER UNSIGNED AUTO_INCREMENT,
  rating_amount TINYINT UNSIGNED,
  will_recommend VARCHAR(100),
  is_suitable VARCHAR(100),
  grading_policy VARCHAR(100),
  explanation_method VARCHAR(100),
  take_again VARCHAR(100),
  student_google_id VARCHAR(50),
  instructor_email VARCHAR(100),
  semester VARCHAR(6) CHECK (semester IN ('Fall', 'Spring', 'Summer')),
  section_year NUMERIC(4, 0) CHECK (section_year > 2009 AND
  section_year < 2050),
```

```
  course_id VARCHAR(15),
  PRIMARY KEY (rating_id),
  FOREIGN KEY (student_google_id) REFERENCES student(student_google_id),
  FOREIGN KEY (instructor_email) REFERENCES instructor(instructor_email)
);

CREATE TABLE course (
  course_id VARCHAR(15),
  title VARCHAR(255),
  credit TINYINT UNSIGNED,
  ects TINYINT UNSIGNED,
  theoritical_credit TINYINT,
  practical_credit TINYINT,
  course_year TINYINT,
  PRIMARY KEY (course_id)
);

CREATE TABLE prerequisite (
  course_id VARCHAR(15),
  prereq_id VARCHAR(15),
  PRIMARY KEY (course_id, prereq_id),
  FOREIGN KEY (course_id) REFERENCES course(course_id) ON DELETE CASCADE,
  FOREIGN KEY (prereq_id) REFERENCES course(course_id)
);

CREATE TABLE classroom (
  building_no VARCHAR(30),
  room_no VARCHAR(15),
  PRIMARY KEY (building_no, room_no)
);

CREATE TABLE section (
  section_id VARCHAR(5),
  semester VARCHAR(6) CHECK (semester IN ('Fall', 'Spring', 'Summer')),
  section_year NUMERIC(4, 0) CHECK (section_year > 2009 AND
  section_year < 2050),
  course_id VARCHAR(15),
  building_no VARCHAR(30),
  room_no VARCHAR(15),
  PRIMARY KEY (section_id, semester, section_year, course_id),
  FOREIGN KEY (building_no, room_no) REFERENCES
  classroom(building_no, room_no),
  FOREIGN KEY (course_id) REFERENCES course(course_id) ON DELETE CASCADE
);

CREATE TABLE course_rating (
  rating_id INTEGER UNSIGNED AUTO_INCREMENT,
  rating_amount TINYINT UNSIGNED,
  will_recommend VARCHAR(100),
  did_enjoy VARCHAR(100),
```

```
  take_similar VARCHAR(100),
  good_content VARCHAR(100),
  was_helpful VARCHAR(100),
  student_google_id VARCHAR(50),
  section_id VARCHAR(5),
  semester VARCHAR(6),
  section_year NUMERIC(4, 0),
  course_id VARCHAR(15),
  PRIMARY KEY (rating_id),
  FOREIGN KEY (student_google_id) REFERENCES student(student_google_id),
  FOREIGN KEY (section_id, semester, section_year, course_id) REFERENCES
  section(section_id, semester, section_year, course_id)
);

CREATE TABLE takes (
  section_id VARCHAR(5),
  semester VARCHAR(6),
  section_year NUMERIC(4, 0),
  course_id VARCHAR(15),
  student_google_id VARCHAR(50),
  grade VARCHAR(5),
  PRIMARY KEY (section_id, semester, section_year, course_id,
  student_google_id),
  FOREIGN KEY (section_id, semester, section_year, course_id) REFERENCES
  section(section_id, semester, section_year, course_id),
  FOREIGN KEY (student_google_id) REFERENCES student(student_google_id)
);

CREATE TABLE teaches (
  section_id VARCHAR(5),
  semester VARCHAR(6),
  section_year NUMERIC(4, 0),
  course_id VARCHAR(15),
  instructor_email VARCHAR(100),
  PRIMARY KEY (section_id, semester, section_year, course_id,
  instructor_email),
  FOREIGN KEY (section_id, semester, section_year, course_id) REFERENCES
  section(section_id, semester, section_year, course_id),
  FOREIGN KEY (instructor_email) REFERENCES instructor(instructor_email)
);

CREATE TABLE department (
  department_name VARCHAR(100),
  department_code VARCHAR(10),
  PRIMARY KEY (department_name)
);

CREATE TABLE student_department (
  student_google_id VARCHAR(50),
  department_name VARCHAR(100),
```

```
  enrolled_type VARCHAR(50),
  PRIMARY KEY (student_google_id, department_name),
  FOREIGN KEY (student_google_id) REFERENCES student(student_google_id),
  FOREIGN KEY (department_name) REFERENCES department(department_name)
);

CREATE TABLE instructor_department (
  instructor_email VARCHAR(100),
  department_name VARCHAR(100),
  PRIMARY KEY (instructor_email, department_name),
  FOREIGN KEY (instructor_email) REFERENCES instructor(instructor_email),
  FOREIGN KEY (department_name) REFERENCES department(department_name)
);

CREATE TABLE course_department (
  course_id VARCHAR(15),
  department_name VARCHAR(100),
  course_type VARCHAR(50),
  PRIMARY KEY (course_id, department_name),
  FOREIGN KEY (course_id) REFERENCES course(course_id),
  FOREIGN KEY (department_name) REFERENCES department(department_name)
);

CREATE TABLE time_slot (
  time_slot_id INTEGER UNSIGNED AUTO_INCREMENT,
  section_day VARCHAR(10),
  start_hr NUMERIC(2, 0) CHECK (start_hr >= 0 AND start_hr < 24),
  start_min NUMERIC(2, 0) CHECK (start_min >= 0 AND start_min < 60),
  end_hr NUMERIC(2, 0) CHECK (end_hr >= 0 AND end_hr < 24),
  end_min NUMERIC(2, 0) CHECK (end_min >= 0 AND end_min < 60),
  section_id VARCHAR(5),
  semester VARCHAR(6),
  section_year NUMERIC(4, 0),
  course_id VARCHAR(15),
  PRIMARY KEY (time_slot_id),
  FOREIGN KEY (section_id, semester, section_year, course_id) REFERENCES
  section(section_id, semester, section_year, course_id) ON DELETE CASCADE
);
```

6 Functional Dependencies and Normalization

- instructor:
  - ◇ $\{instructor\_google\_id\} \longrightarrow \{instructor\_name, instructor\_email, instructor\_profile\_picture\}$
  - ◇ $\{instructor\_email\} \longrightarrow \{instructor\_google\_id\}$

Normalization Check:

The relation is in BCNF since there is no bad FD.

- student:
  - ◇ $\{student\_google\_id\} \longrightarrow \{student\_id, student\_name, student\_email, student\_profile\_picture, student\_semester, standing, completed\_credits, completed\_ects, gpa, advisor\}$
  - ◇ $\{student\_email\} \longrightarrow \{student\_google\_id\}$
  - ◇ $\{student\_id\} \longrightarrow \{student\_email\}$

  Normalization Check:

  The relation is in BCNF since there is no bad FD. However, there is a possible FD ($\{student\_semester\} \longrightarrow \{standing\}$) which may require this relation to decompose. My assumption is, this holds most of the time but there are some rare cases in which this FD does not hold. Therefore, I am not considering it due to consistency purposes.

- testimonial:
  - ◇ $\{testimonial\_id\} \longrightarrow \{testimonial\_text, testifier\_position, student\_google\_id\}$

  Normalization Check:

  The relation is in BCNF since there is no bad FD.

- instructor_rating:
  - ◇ $\{rating\_id\} \longrightarrow \{rating\_amount, will\_recommend, is\_suitable, grading\_policy, explanation\_method, take\_again, student\_google\_id, instructor\_email, course\_id, semester, section\_year\}$

  Normalization Check:

  The relation is in BCNF since there is no bad FD.

- course:
  - ◇ $\{course\_id\} \longrightarrow \{title, credit, ects, theoritical\_credit, practical\_credit, course\_year\}$

  Normalization Check:

The relation is in BCNF since there is no bad FD.

- prerequisite:
  - ◇ $\{course\_id, prereq\_id\} \longrightarrow \{course\_id, prereq\_id\}$

    Normalization Check:

    The relation is in BCNF since there is no bad FD.

- classroom:
  - ◇ $\{building\_no, room\_no\} \longrightarrow \{building\_no, room\_no\}$

    Normalization Check:

    The relation is in BCNF since there is no bad FD.

- section:
  - ◇ $\{section\_id, semester, section\_year, course\_id\} \longrightarrow \{build\_no, room\_no\}$

    Normalization Check:

    The relation is in BCNF since there is no bad FD.

- course_rating:
  - ◇ $\{rating\_id\} \longrightarrow \{rating\_amount, will\_recommend, did\_enjoy, take\_similar, good\_content, was\_helpful, student\_google\_id, section\_id, semester, section\_year, course\_id\}$

    Normalization Check:

    The relation is in BCNF since there is no bad FD.

- takes:
  - ◇ $\{section\_id, semester, section\_year, student\_google\_id, course\_id\} \longrightarrow \{grade\}$

    Normalization Check:

    The relation is in BCNF since there is no bad FD.

- teaches:
  - $\{section\_id, semester, section\_year, course\_id, instructor\_email\} \longrightarrow \{section\_id, semester, section\_year, course\_id, instructor\_email\}$

    Normalization Check:

    The relation is in BCNF since there is no bad FD.

- department:
  - $\{department\_code\} \longrightarrow \{department\_name\}$
  - $\{department\_name\} \longrightarrow \{department\_code\}$

    Normalization Check:

    The relation is in BCNF since there is no bad FD.

- student_department:
  - $\{department\_name, student\_google\_id\} \longrightarrow \{enrolled\_type\}$

    Normalization Check:

    The relation is in BCNF since there is no bad FD.

- instructor_department:
  - $\{department\_name, instructor\_email\} \longrightarrow \{department\_name, instructor\_email\}$

    Normalization Check:

    The relation is in BCNF since there is no bad FD.

- course_department:
  - $\{course\_id, department\_name\} \longrightarrow \{course\_type\}$

    Normalization Check:

    The relation is in BCNF since there is no bad FD.

- time_slot:

- $\{time\_slot\_id\} \longrightarrow \{section\_day, start\_hr, start\_min, end\_hr, end\_min, section\_id, semester, section\_year, course\_id\}$

Normalization Check:

The relation is in BCNF since there is no bad FD.

7 Learning Results and Discussions

In this section, we quantitatively evaluate our techniques from different perspectives. We first discuss the dataset and the evaluation metrics, and then discuss individual experimental results with their implications.

7.1 Dataset

For experimental evaluation, we use a real student course grade dataset that is obtained from Istanbul Sehir University. We are given the dataset after all student personal information (e.g., name, id, nationality, class, etc.) are removed. The raw dataset contains 55,475 rows of course grades spanning between the years of 2010 – 2015. Several courses are designed to be pass/fail courses, that is, there is no letter grade for such courses, and students either pass or fail. There are also several courses with incomplete grades. We eliminated all these rows (6,128 course grade records in total). Furthermore, we remove graduate courses (5,643 rows) since the tool is only for undergraduate students. After all these filtering steps, the final dataset contains 43,704 rows of course grades that belong to 2,132 distinct students. Figure 2 shows the distribution of letter grades which are used as target values in our experiments.

7.2 Preprocessing and Feature Engineering

The original dataset contains the features namely, Course Code, Course Title, Unidentified Student Number, Department Code, Course Level, Letter Grade, Completion Status, Semester, Academic Year and Grading Scale. Istanbul Sehir University have changed some of their course names and course codes, therefore the data is somehow inconsistent with the current courses taught in university. To overcome this problem, we use different web scraping techniques to download the updated course details available on university's website (sehir.edu.tr/en). We use different filters to check the inconsistent courses and update them with their new details. Scraping updated details from university's website provide us with 5 more essential features namely, Course Credit, ECTS, Theoretical Credit, Practical Credit and Course Year which combine these with the other feature set. After all these preprocessing steps, we extract new features concerning both students and courses from the data we own. As this data is time series oriented, for each student in a semester we calculate their GPA, Completed Credit, Completed ECTS, Standing (Freshman, Sophomore, Junior and Senior), average grades in

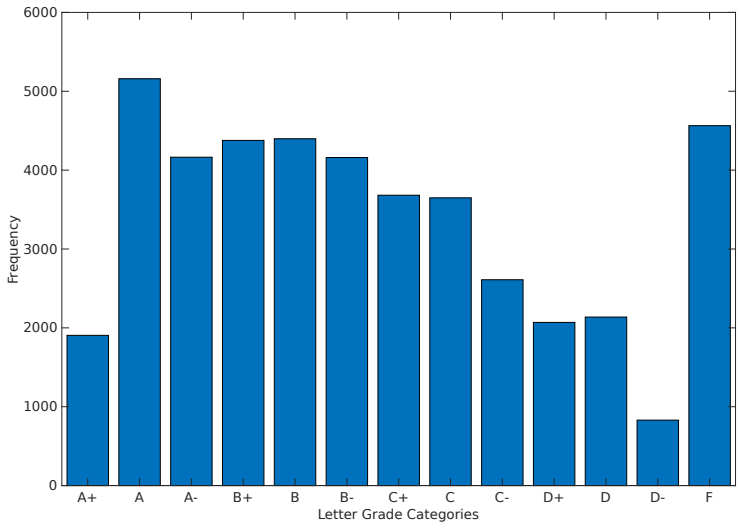

Fig. 2. Letter Grade Frequency Distribution

courses with their corresponding subjects (i.e. if a course has a code like CS 361, we calculate the average grade from all CS subjects up until that semester), average grade of students who have taken that course so far and average grade of students in courses with the same subject. We combine these extracted features with our feature set as well and start testing different learning models.

7.3 Metrics

In order to evaluate the accuracy, we employ mean absolute error (MAE), mean square error (MSE) and R^2 metrics.

- The mean absolute error is a risk metric corresponding to the expected value of the absolute error loss or $l1$-norm loss. If $\hat{y}_i$ is the predicted value of the i-th sample, and y_i is the corresponding true value, then the mean absolute error (MAE) estimated over n_{samples} is defined as

$$\text{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} |y_i - \hat{y}_i| .$$

- The mean square error is a risk metric corresponding to the expected value of the squared (quadratic) error or loss. If $\hat{y}_i$ is the predicted value of the i-th sample, and y_i is the corresponding true value, then the mean squared error (MSE) estimated over n_{samples} is defined as

$$\text{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2.$$

◇ The R^2 score computes the coefficient of determination. It represents the proportion of variance (of y) that has been explained by the independent variables in the model. It provides an indication of goodness of fit and therefore a measure of how well unseen samples are likely to be predicted by the model, through the proportion of explained variance.

As such variance is dataset dependent, R^2 may not be meaningfully comparable across different datasets. Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0.

If $\hat{y}_i$ is the predicted value of the i-th sample and y_i is the corresponding true value for total n samples, the estimated R^2 is defined as:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$$

where $\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i$ and $\sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} \epsilon_i^2$

7.4 Experiments - Regression Models

In this section, we present major experimental results and the associated observations that we made using Regression models. The proposed techniques are implemented in Python 3.6.9, and experiments are run on a Dell machine with 2.00 GHz i7 CPU and 8 GB memory. For all runs, we use the Scikit Learn library from Python to implement different learning algorithms. We use Matlab for visualization purposes.

7.5 Baseline Approach

In this section, we implement different learning models as our baseline approach to quantify the performance of models. In order to split time series data into training and test, at each step we consider training data from the beginning up until a specific semester and the next semester as test data. This method provide us 13 training splits as there are 14 semesters of data available.

◇ **BayesianRidge Regressor:**

Bayesian regression techniques can be used to include regularization parameters in the estimation procedure: the regularization parameter is not set in a hard sense but tuned to the data at hand. This can be done by introducing uninformative priors over the hyper parameters of the model. The

ℓ_2 regularization used in Ridge regression and classification is equivalent to finding a maximum a posteriori estimation under a Gaussian prior over the coefficients w with precision λ^{-1}. Instead of setting lambda manually, it is possible to treat it as a random variable to be estimated from the data.

BayesianRidge estimates a probabilistic model of the regression problem as described above. The prior for the coefficient w is given by a spherical Gaussian:

$$p(w|\lambda) = \mathcal{N}(w|0, \lambda^{-1}\mathbf{I}_p)$$

Figure 3 shows the errors corresponding to BayesianRidge Regressor. As visible from the plot, the model does not overfit and it generalizes well on the given dataset. Moreover, the default hyper-parameters from Scikit Learn are used to train the models.

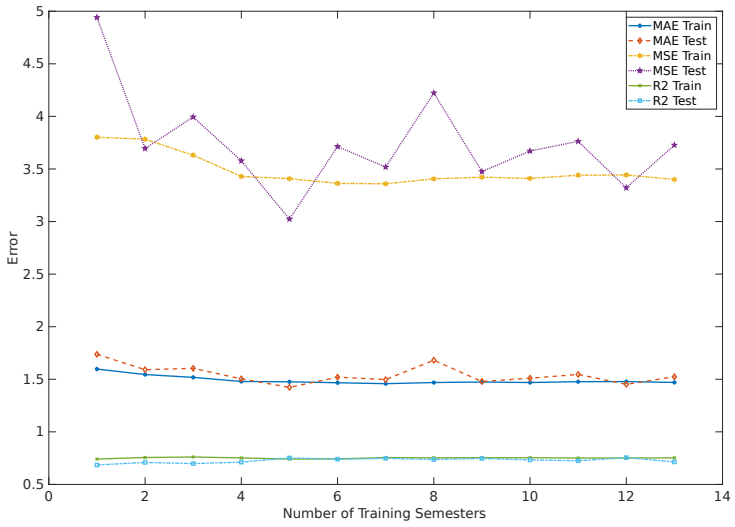

Fig. 3. BayesianRidge Regressor Error Scores

◇ **Ridge Regressor:**

The Ordinary Least Squares refers to minimizing the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation. The coefficient estimates for Ordinary Least Squares rely on the independence of the features. When features are correlated and the columns of the design matrix X have an approximate linear dependence, the design matrix becomes close to singular and as a result, the least-squares

estimate becomes highly sensitive to random errors in the observed target, producing a large variance. This situation of multicollinearity can arise, for example, when data are collected without an experimental design.

Ridge regression addresses some of the problems of Ordinary Least Squares by imposing a penalty on the size of the coefficients. The ridge coefficients minimize a penalized residual sum of squares:

$$\min_w ||Xw - y||_2^2 + \alpha||w||_2^2$$

The complexity parameter $\alpha \geq 0$ controls the amount of shrinkage: the larger the value of α, the greater the amount of shrinkage and thus the coefficients become more robust to collinearity. Figure 4 shows the errors corresponding to Ridge Regressor. As visible from the plot, the model does not overfit and it generalizes well on the given dataset. Moreover, the default hyper-parameters from Scikit Learn are used to train the models.

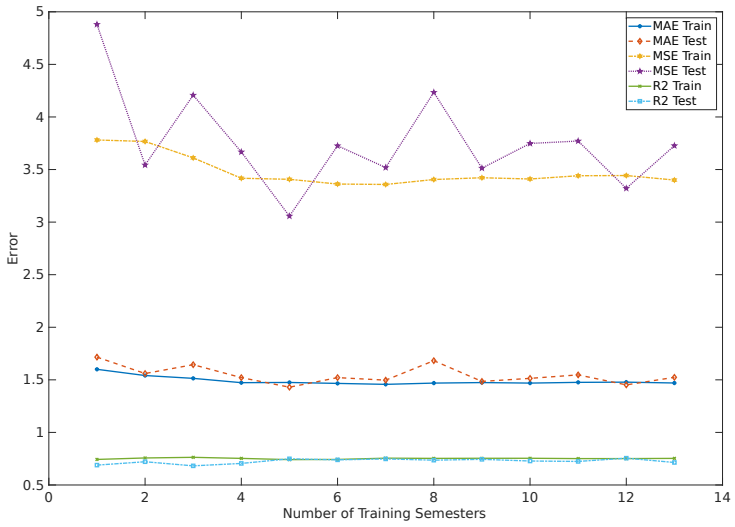

Fig. 4. Ridge Regressor Error Scores

◇ **AdaBoost Regressor:**

The core principle of AdaBoost is to fit a sequence of weak learners (i.e., models that are only slightly better than random guessing, such as small decision trees) on repeatedly modified versions of the data. The predictions from all of them are then combined through a weighted majority vote (or sum) to produce the final prediction. The data modifications at each so-called

boosting iteration consist of applying weights w_1, w_2, ..., w_N to each of the training samples. Initially, those weights are all set to $w_i = 1/N$, so that the first step simply trains a weak learner on the original data. For each successive iteration, the sample weights are individually modified and the learning algorithm is reapplied to the reweighted data. At a given step, those training examples that were incorrectly predicted by the boosted model induced at the previous step have their weights increased, whereas the weights are decreased for those that were predicted correctly. As iterations proceed, examples that are difficult to predict receive ever-increasing influence. Each subsequent weak learner is thereby forced to concentrate on the examples that are missed by the previous ones in the sequence.

An AdaBoost Regressor is a meta-estimator that begins by fitting a regressor on the original dataset and then fits additional copies of the regressor on the same dataset but where the weights of instances are adjusted according to the error of the current prediction. As such, subsequent regressors focus more on difficult cases.

Figure 5 shows the errors corresponding to AdaBoost Regressor. As visible from the plot, the model does not overfit and it generalizes well on the given dataset. Moreover, the default hyper-parameters from Scikit Learn are used to train the models.

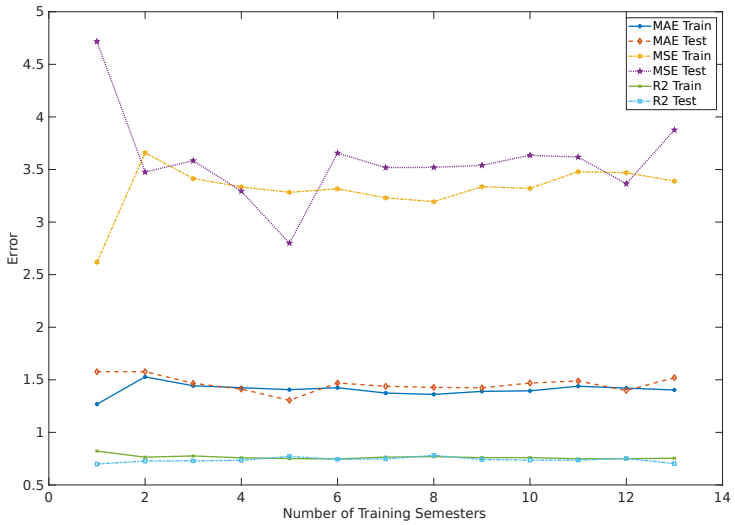

Fig. 5. AdaBoost Regressor Error Scores

◇ **GradientBoosting Regressor:**

Gradient Tree Boosting or Gradient Boosted Decision Trees (GBDT) is a generalization of boosting to arbitrary differentiable loss functions. GBDT is an accurate and effective off-the-shelf procedure that can be used for both regression and classification problems in a variety of areas including Web search ranking and ecology.

In Regression model, GB builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function.

Figure 6 shows the errors corresponding to GradientBoosting Regressor. As visible from the plot, the model does not overfit for larger values of training data and it generalizes well on the given dataset. Moreover, the default hyper-parameters from Scikit Learn are used to train the models.

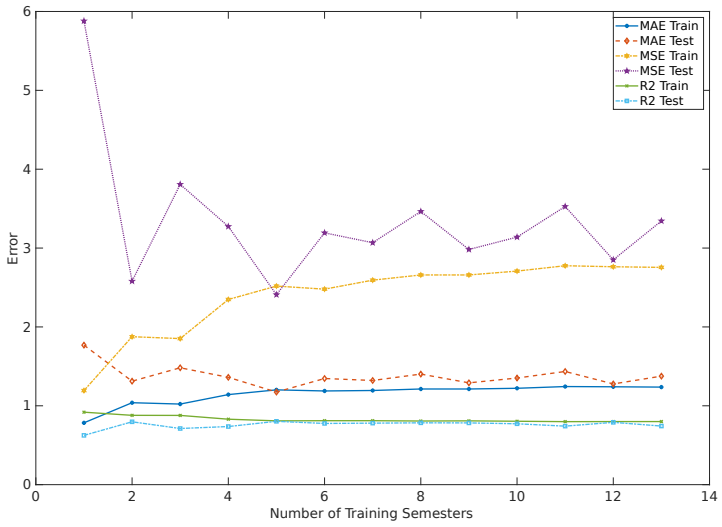

Fig. 6. GradientBoosting Regressor Error Scores

Since the best performance has been achieved by GradientBoostingRegressor, we implement this model to predict course grades and prediction intervals for students. As suggested by [2], ensemble models perform well on these types of data due to the fact that they are made of different models.

7.6 Collaborative Filtering

This section will be completed after a thorough study of algorithms and approaches.

8 Application Design and Implementations

In this section, we will emphasize on the design patterns used in this project. In contrast, we will discuss a high level architecture of the application. Moreover, a set of working features and SQL queries are also indicated in this section.

8.1 High Level Architecture

Software architecture refers to the fundamental structures of a software system and the discipline of creating such structures and systems. Each structure comprises software elements, relations among them, and properties of both elements and relations. In some sense, software architecture is sometimes considered as a metaphor to building architecture. In order to produce an adaptable and robust design, we implement a 5-tier architecture that is easily customize-able to any other requirements. Figure 7 shows how different layers are connected to each other.

1. **Presentation Layer:**

   At the top level you want to run your business logic. That could be in a human-focused screen or an API talking to another system – it doesn't really matter. The point is your business logic should be isolated from the front-end. But at the same time the front-end may need features/requirements to make the application to work in its environment, like showing the user what courses are required for their graduation. This layer consists of end-users who will be interacting with the application through their browsers, and HTML, CSS and JS for displaying and formatting web pages. This layer interacts with Service Layer through HTTP requests.

2. **Service Layer:**

   Service Layer is an essential layer which links everything together. In contrast, it provides a Command pattern and the Adapter pattern between the presentation layer and all the layers below. This layer is mainly consists of components which are independent of Business Logic Layer in terms of their functionality. They only contact with low layers when they have their data ready. For instance, The Business Runner can handle UI-centric Data Transfer Objects, with things like drop down lists, forms etc, and then map just the data that the Business Logic needs to do its job. That insulates the

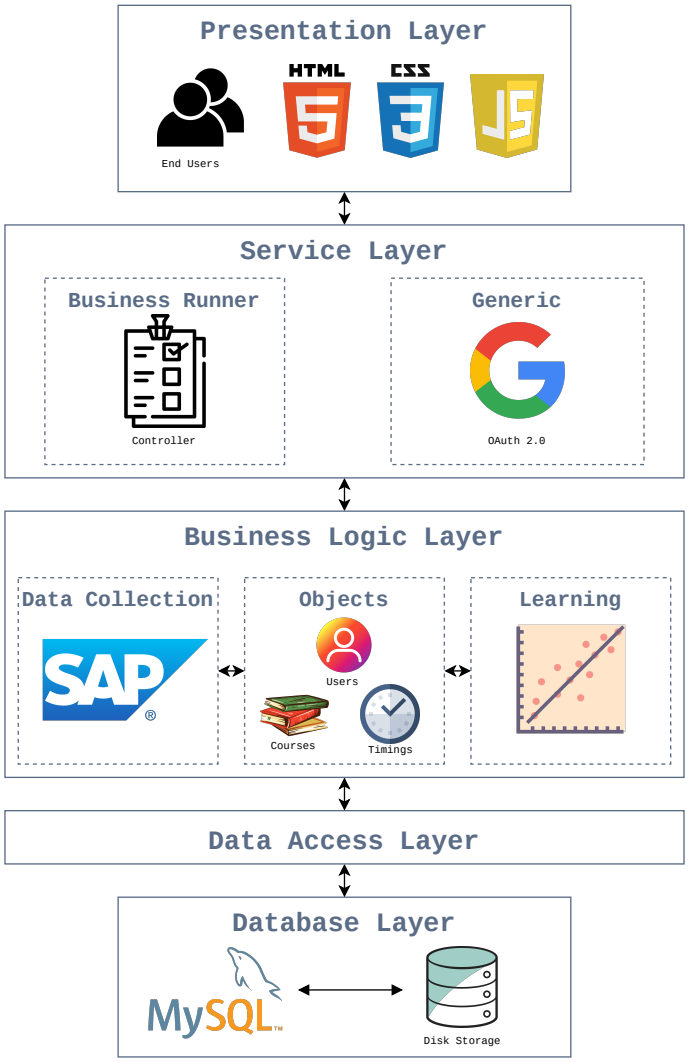

Fig. 7. High Level Application Architecture

Business Logic from any need to think about, or deal with, UI data. Moreover, some Generic functionalities such as user sign-in and sign-up are also handled in this layer.

3. **Business Logic Layer:**

   The Business Logic or Domain Logic is the part of the program that encodes the real-world business rules that determine how data can be created, stored, and changed. It is the layer which is understandable for users other than DBA and programmers, and they are the ones who define specific functionalities for this layer. All instances like Users (Student,Instructor and Admin), Courses, Timings, Departments, Ratings, etc... are defined in this layer. This Layer contains of other sub-sections like Learning and Data Collection as well. The objects mentioned above communicate with Learning section when a grade prediction is being made through exchanging feature sets and predictions. On the other hand, objects interact with Data Collection section when a new user registers or timings of a new semester is being fetched.

4. **Data Access Layer:**

   The Data Access Layer contains a corresponding functionality for each of the Business Logic objects. It is the layer which touches the Database Layer through executing SQL queries. For instance, when a Student object asks for their unrated courses in a semester, a function related to Student is being called from this layer.

5. **Database Layer:**

   The Database Layer houses the Database Management System and Disk Storage. They only communicate with Data Access Layer by means of SQL queries and returning results. It is the job of DBMS to access securely the disk storage and make necessary changes.

8.2 SQL Queries

```
INSERT INTO department (department_name, department_code)
VALUES ('Psychology (English)','PSY');
```

Semantics: Insert Psychology (English) into departments table

```
INSERT INTO course (course_id, title, credit, ects,
                    theoritical_credit, practical_credit,
```

```
                    course_year)
VALUES ('ENGR 101','Introduction to Programming',3,5,2,2,1);
```

Semantics: Insert ENGR 101 into course table

```
INSERT INTO course_department (course_id, department_name,
                               course_type)
VALUES ('HUK 101','Law','Required');
```

Semantics: Insert HUK 101 into course_department table

```
INSERT INTO prerequisite (course_id, prereq_id)
VALUES ('CS 350', 'ENGR 102');
```

Semantics: Insert CS 350 with ENGR 102 as it's pre_req into prerequisites table

```
INSERT INTO instructor (instructor_email, instructor_name)
VALUES ('barisarslan@sehir.edu.tr', 'BARIS ARSLAN');
```

Semantics: Insert BARIS ARSLAN into instructor table

```
INSERT INTO instructor_department (instructor_email,
                                   department_name)
VALUES ('nardic@sehir.edu.tr', 'Sociology (English)');
```

Semantics: Insert NURULLAH ARDIC's department into instructor_department table

```
INSERT INTO classroom (building_no, room_no)
VALUES ('ACAD BUILD 4', '4301');
```

Semantics: Insert classroom 4301 into classroom table

```
INSERT INTO section (section_id, semester, section_year,
                     course_id, building_no, room_no)
VALUES ('01', 'Spring', 2019, 'CS 350', 'ACAD BUILD 4',
        '4301');
```

Semantics: Insert CS 350 section in Spring 2019 into section table

```
INSERT INTO teaches (section_id, semester, section_year,
                     course_id, instructor_email)
VALUES ('01', 'Spring', 2019, 'CS 350',
       (SELECT instructor_email
        FROM instructor
        WHERE instructor_name='ALI CAKMAK'));
```

Semantics: Insert CS 350 01 taught by Ali Cakmak in Spring 2019 into teaches table

```
INSERT INTO time_slot (section_day, start_hr, start_min,
                       end_hr, end_min, section_id,
                       semester, section_year, course_id)
VALUES ('Tuesday', 13, 00, 15, 00, 01, 'Fall', 2018, 'ENGR 101');
```

Semantics: Insert ENGR 101's timing into time_slot table

```
INSERT INTO testimonial (testimonial_text, testifier_position,
                         student_google_id)
VALUES ('This App is Great. Awesome!', 'Athlete', 'X');
```

Semantics: Insert student X's testimony into testimonial table

```
SELECT student_name, testimonial_text, testifier_position,
       student_profile_picture
FROM testimonial AS t, student AS s
WHERE t.student_google_id = s.student_google_id;
```

Semantics: List all testimony details of students

```
UPDATE instructor
SET instructor_google_id = 'X',
    instructor_profile_picture = 'url'
WHERE instructor_email = 'apaydin@sehir.edu.tr';
```

Semantics: Update Mehmet Serkan Apaydin's details

```
SELECT *
FROM instructor
WHERE instructor_email = 'mehmetbaysan@sehir.edu.tr';
```

Semantics: Get Mehmet Baysan's details

```
SELECT department_name
FROM instructor_department
WHERE instructor_email = 'ahmetbulut@sehir.edu.tr';
```

Semantics: Get Ahmet Bulut's department

```
SELECT ts.course_id, sd.title, ts.start_hr, ts.start_min,
       ts.end_hr, ts.end_min, sd.building_no, sd.room_no
FROM time_slot AS ts,
    (SELECT s.section_id, s.semester, s.section_year, s.course_id,
            s.building_no, s.room_no, c.title
     FROM course AS c,
         (SELECT *
          FROM section
          WHERE (section_id, course_id, semester, section_year) IN
                (SELECT section_id, course_id, semester, section_year
                 FROM teaches
                 WHERE instructor_email = 'alicakmak@sehir.edu.tr'
                 AND section_year = 2019
                 AND semester = 'Spring')) AS s
     WHERE c.course_id = s.course_id) AS sd
WHERE ts.section_id = sd.section_id
AND ts.section_year = sd.section_year
AND ts.semester = sd.semester
AND ts.course_id = sd.course_id
AND ts.section_day = 'Wednesday';
```

Semantics: List Ali Cakmak's timetable on Wednesday

```
SELECT res.course_id, c.title, res.department_code,
       res.total_enrolled, res.avg_gpa
FROM course AS c,
    (SELECT res.course_id, res.department_code,
            COUNT(*) AS total_enrolled, AVG(s.gpa) AS avg_gpa
     FROM student AS s,
        (SELECT t.course_id, d.department_code, t.student_google_id
         FROM takes AS t, student_department AS sd, department AS d
         WHERE section_year = 2019
         AND semester = 'Spring'
         AND course_id IN
            (SELECT course_id
             FROM teaches
             WHERE instructor_email = 'ensargul@sehir.edu.tr'
             AND t.student_google_id=sd.student_google_id
             AND d.department_name=sd.department_name) AS res
    WHERE res.student_google_id=s.student_google_id
    GROUP BY res.course_id, res.department_code) AS res
WHERE res.course_id = c.course_id;
```

Semantics: List course statistics taught by Ensar Gul in Spring 2019

```
SELECT *
FROM student
WHERE student_google_id = 'X';
```

Semantics: Get student X's details

```
INSERT INTO student (student_google_id, student_name,
                     student_email, student_profile_picture)
VALUES('X', 'Ali Reza Ibrahimzada',
       'aliibrahimzada@std.sehir.edu.tr', 'url');
```

Semantics: Insert student X into student table

```
UPDATE student
SET student_id = '218012083', student_semester = 4,
    advisor = (SELECT instructor_email
               FROM instructor
               WHERE instructor_name = 'ABDULKERIM KAR'),
    standing = 'Sophomore', gpa = 3.93, completed_credits = 63,
    completed_ects = 100
WHERE student_google_id = 'X';
```

Semantics: Update Ali Reza's academic details

```
SELECT department_name
FROM student_department
WHERE enrolled_type = 'Main Prg'
AND student_google_id = 'X';
```

Semantics: Get student X's main major

```
SELECT COUNT(*)
FROM student_department
WHERE course_type='Required' AND department_name = 'Psychology (English)';
```

Semantics: Get required number of courses to graduate from Psychology department

```
SELECT COUNT(*)
FROM takes
WHERE student_google_id = 'X'
AND grade IS NOT NULL
AND grade NOT IN ('F', 'IA', 'W');
```

Semantics: Get completed number of courses by student X

```
INSERT INTO student_department
VALUES ('X', 'Law', 'Main Prg');
```

Semantics: Insert Law as main major of student X

```
SELECT semester, section_year, course_id, student_google_id
FROM takes
WHERE student_google_id = 'X'
AND course_id = 'ENGR 101'
AND semester = 'Fall'
AND section_year = '2018';
```

Semantics: List all courses student X takes in Fall 2018

```
INSERT INTO takes
VALUES ('05', 'Spring', 2019, 'UNI 123', 'X', 'A+');
```

Semantics: Insert UNI 123 into student X's record

```
UPDATE takes
SET grade = 'A'
WHERE student_google_id = 'X'
AND course_id = 'ENGR 105'
AND semester = 'Fall'
AND section_year = 2018;
```

Semantics: Update student X's grade from ENGR 105

```
SELECT res.section_id, res.semester, res.section_year,
       res.course_id, c.title
FROM course AS c, (SELECT section_id, semester, section_year,
                          course_id
                   FROM takes AS t
                   WHERE student_google_id = 'X'
                   AND grade IS NULL
                   AND (course_id, section_id, semester,
                   section_year) NOT IN
                       (SELECT course_id, section_id, semester,
```

```
                                section_year
                            FROM course_rating AS cr
                            WHERE cr.student_google_id =
                                  t.student_google_id)) AS res
WHERE c.course_id = res.course_id;
```

Semantics: List all unrated courses by student X in the current semester

```
INSERT INTO course_rating (rating_amount, will_recommend,
            did_enjoy, take_similar, good_content, was_helpful,
            student_google_id, section_id, semester, section_year,
            course_id)
VALUES (8, 'No', 'Yes', 'Not Sure', 'No Opinion', 'Not Really',
       'X', '01', 'Fall', 2019, 'CS 201');
```

Semantics: Insert student X's CS 201 rating into course_rating table

```
SELECT res.section_year, res.semester, res.course_id, res.section_id,
      (SELECT title
     FROM course
     WHERE course_id = res.course_id),
    (SELECT instructor_name
     FROM instructor
     WHERE instructor_email = res.instructor_email),
    (SELECT instructor_profile_picture
     FROM instructor
     WHERE instructor_email = res.instructor_email),
     res.instructor_email
FROM (SELECT *
      FROM teaches
      WHERE (section_id, semester, section_year, course_id) IN
            (SELECT section_id, semester, section_year, course_id
             FROM takes as t
             WHERE student_google_id = 'X'
             AND t.grade IS NULL
             AND (course_id, semester, section_year) NOT IN
                 (SELECT course_id, semester, section_year
                  FROM instructor_rating AS ir
                  WHERE t.student_google_id =
                  ir.student_google_id))) AS res;
```

Semantics: List all unrated instructors by student X in the current semester

```
INSERT INTO instructor_rating (rating_amount, will_recommend,
            is_suitable, grading_policy, explanation_method,
            take_again, student_google_id, instructor_email,
            semester, section_year, course_id)
VALUES (8, 'Yes', 'Yes', 'Fair', 'Good', 'Not Sure', 'X',
       'ihsancicek@sehir.edu.tr', 'Fall', 2019, 'EECS 218');
```

Semantics: Insert student X's review of Ihsan Cicek into instructor_rating table

```
SELECT ts.course_id, sd.title, ts.start_hr, ts.start_min, ts.end_hr,
       ts.end_min, sd.building_no, sd.room_no
FROM time_slot AS ts,
    (SELECT c.course_id, res.semester, res.section_year,
            res.section_id, c.title, res.building_no, res.room_no
     FROM course as c,
         (SELECT *
          FROM section
          WHERE (section_id, semester, section_year, course_id) IN
                (SELECT section_id, semester, section_year, course_id
                 FROM takes
                 WHERE student_google_id = 'X'
                 AND grade IS NULL )) AS res
     WHERE c.course_id = res.course_id) AS sd
WHERE ts.section_id = sd.section_id
AND ts.course_id = sd.course_id
AND ts.semester = sd.semester
AND ts.section_year = sd.section_year
AND ts.section_day = 'Thursday';
```

Semantics: List student X's timetable on Thursday

```
SELECT t.course_id, c.title, t.semester, t.section_year,
       t.grade, cd.course_type
FROM course_department AS cd,
    (SELECT c.course_id, c.title, res.semester,
            res.section_year, res.grade
     FROM course AS c,
         (SELECT semester, section_year, course_id, grade
          FROM takes
          WHERE student_google_id = 'X'
          AND grade IS NOT NULL) AS res
     WHERE c.course_id = res.course_id) AS t
WHERE cd.course_id = t.course_id
AND cd.department_name = (SELECT department_name
                          FROM student_department
                          WHERE student_google_id = 'X'
                          AND enrolled_type = 'Main Prg')
ORDER BY t.section_year, t.semester, t.course_id;
```

Semantics: List completed courses by student X

```
SELECT cd.course_id, c.title, cd.course_type, c.credit,
       c.ects, c.course_year
FROM course_department AS cd, course AS c
WHERE cd.course_id = c.course_id
AND cd.course_type = 'Required'
AND cd.course_id NOT IN (SELECT course_id
                         FROM takes
                         WHERE student_google_id='X'
                         AND grade IS NOT NULL)
```

```
AND (SELECT department_name
     FROM student_department
     WHERE student_google_id = 'X') = cd.department_name
ORDER BY c.course_year;
```

Semantics: List student X's incompleted required courses

```
SELECT grade, COUNT(grade)
FROM takes
WHERE course_id = 'CS 350'
GROUP BY grade;
```

Semantics: List grades and their corresponding counts from CS 350 course

```
SELECT grade, COUNT(grade)
FROM takes
WHERE course_id REGEXP '^UNI[[:space:]][0-9]*$'
GROUP BY grade;
```

Semantics: List grades and their corresponding counts from UNI subjects

```
SELECT t.grade, c.credit
FROM takes AS t, student AS s, course AS c
WHERE t.student_google_id = s.student_google_id
AND c.course_id = t.course_id
AND c.course_id REGEXP '^EECS[[:space:]][0-9]*$'
AND s.student_google_id='X'
AND t.grade IS NOT NULL;
```

Semantics: List grades and their corresponding counts of student X from EECS subjects

```
SELECT c.course_id, title, cd.course_type, credit, ects,
       theoritical_credit, practical_credit, course_year, d.department_code
FROM course AS c, course_department AS cd, department as d
WHERE c.course_id REGEXP '^CS[[:space:]][0-9]*$'
AND c.course_id=cd.course_id
AND cd.department_name = 'Computer Science and Engineering (English)'
AND d.department_name = cd.department_name
AND c.course_id NOT IN (SELECT course_id
                        FROM takes
                        WHERE student_google_id = 'X'
                        AND grade IS NOT NULL);
```

Semantics: Get candidate courses from CS subject for student X to make a prediction

8.3 Tools

In this section we will discuss about different tools we used in implementation of the project. In presentation layer, we use HTML, CSS and JAVASCRIPT to produce dynamic web pages. Moreover, we use Javascript's jQuery library to ease manipulation and handling of data in front end. Furthermore, we use Google OAuth 2.0 to access Google APIs in order to create a safe protection

environment for users. All user credentials are stored and protected by Google. Only users having a SEHIR Google account are able to use this tool.

What more is, in the Service and Logic Layers we use Python as programming language in order to manipulate and handle data. Flask is used as the web framework due to it's lightness and easiness. In the Data Access Layer we use mysql.connector library to interact and execute queries between application and the Database Layer. On the other hand, we use Google Visualization API for visualizing statistical data in our tool. Figure 24 shows different statistical data we plot for instructors. Moreover, Scikit Learn is used to implement the learning algorithms the application is using to predict student grades. Besides, we use Numpy, Pandas, Scipy and Matlab as well for calculating and visualizing data. Finally, MySQL is used as Database Management System in the Database Layer.

8.4 Data

In this section we will emphasize on how data is being collected and used. To begin with, the data used for training statistical learning models are provided by Istanbul Sehir University as discussed in section *7.1*. In order to maintain the application in the long term, we scrape students data from their academic web portal (SAP for Istanbul Sehir University). We use Selenium to scrape students data. For privacy purposes, we do not store students SAP credentials and they are required to fetch their up-to-date data prior to the start of each academic semester.

8.5 Working Features

– **Google Authentication**

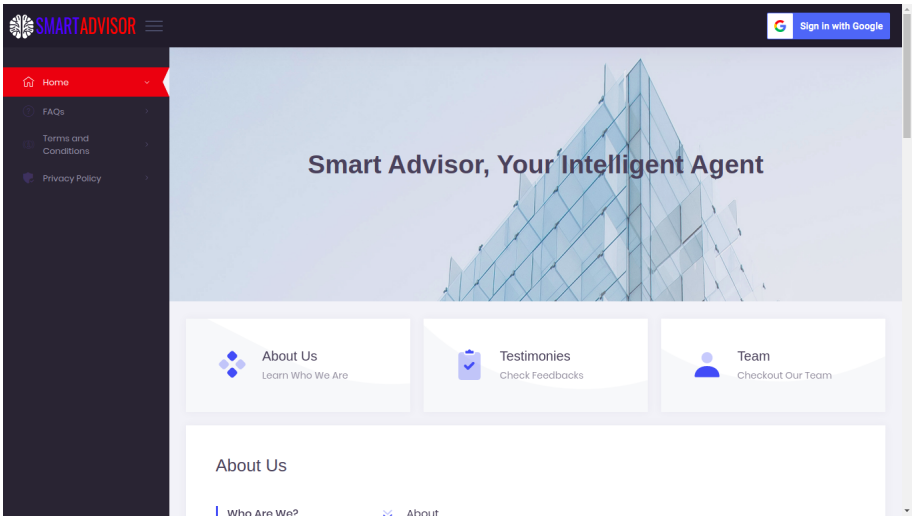

Fig. 8. Google Sign In

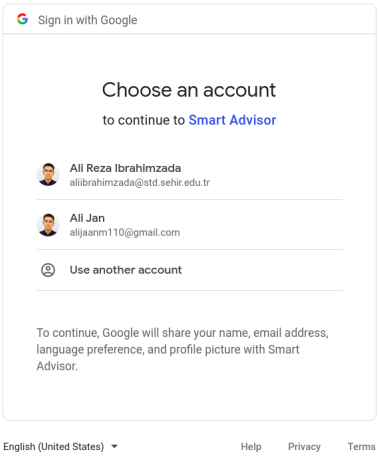

Fig. 9. Google Authentication

– **FAQs**

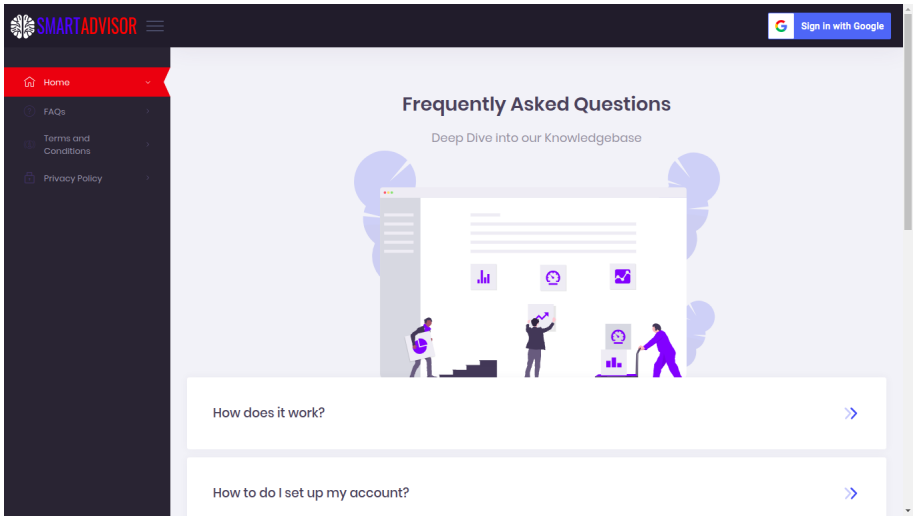

Fig. 10. FAQs

– **Profile and Graduation Progress**

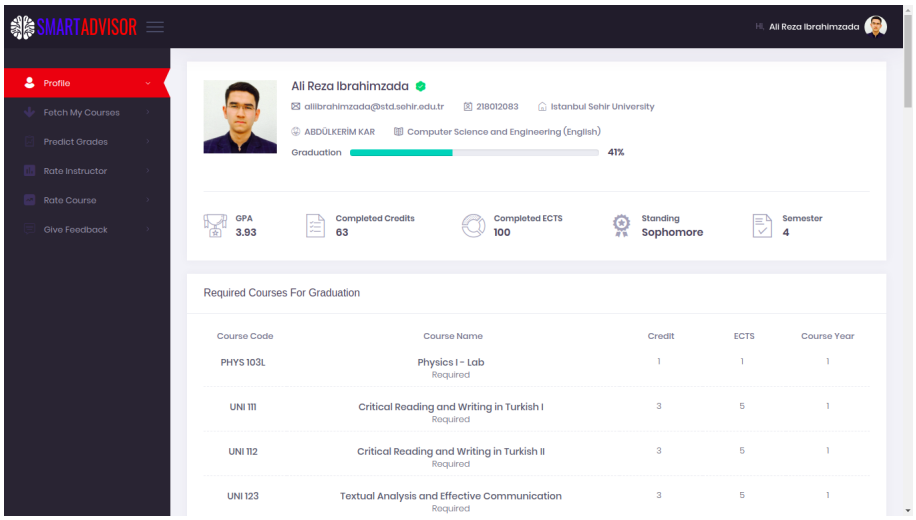

Fig. 11. Student Profile

– **Timetable**

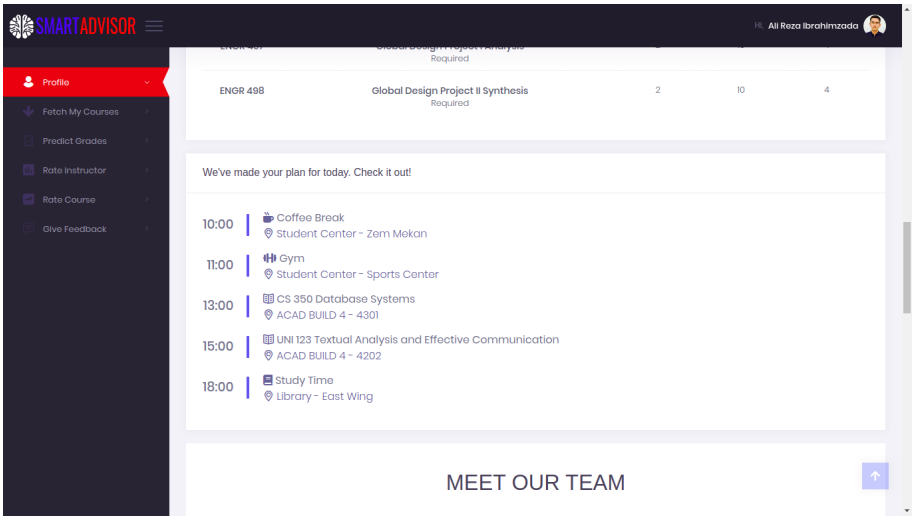

Fig. 12. Timetable

– **Team**

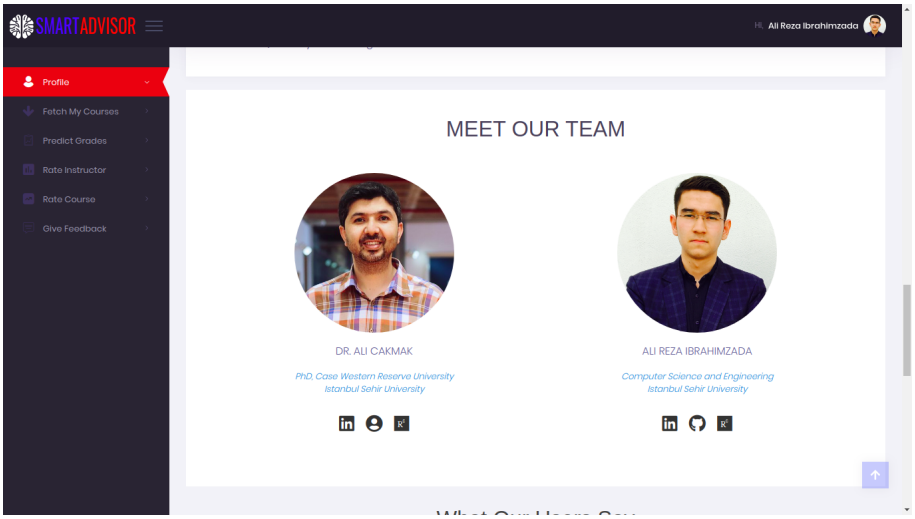

Fig. 13. Team Members

– **Contact**

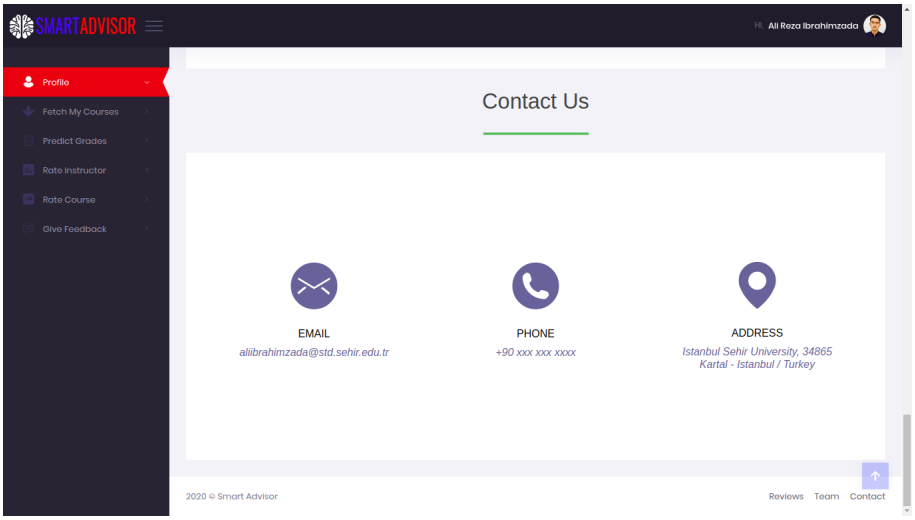

Fig. 14. Contact Details

– **Fetch SAP**

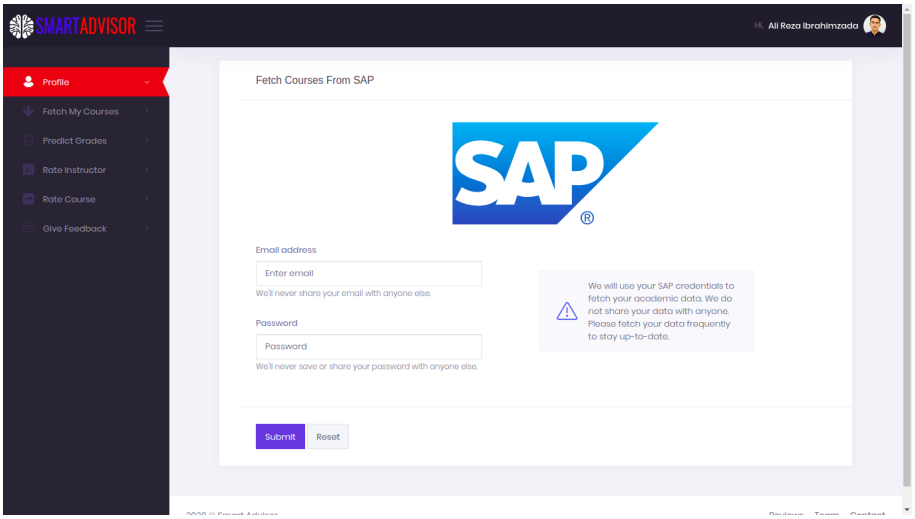

Fig. 15. Fetch SAP data

– **Grade Prediction**

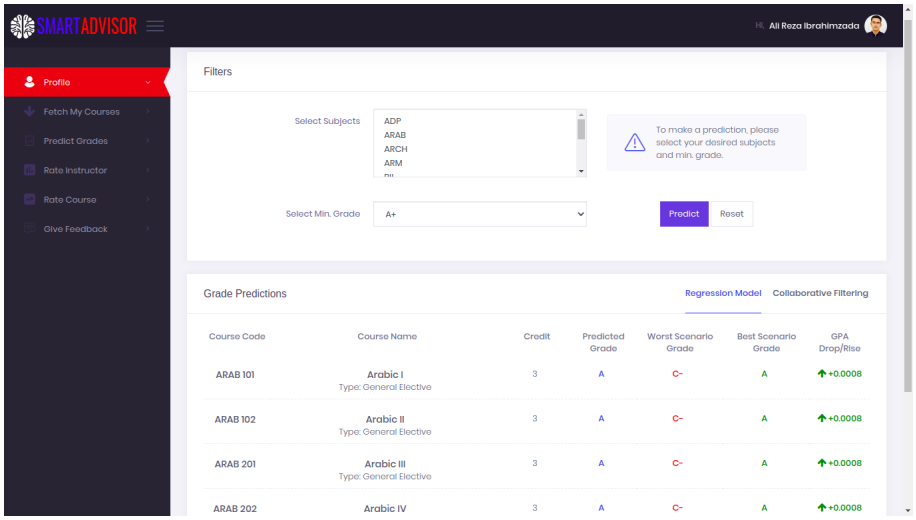

Fig. 16. Search Filters

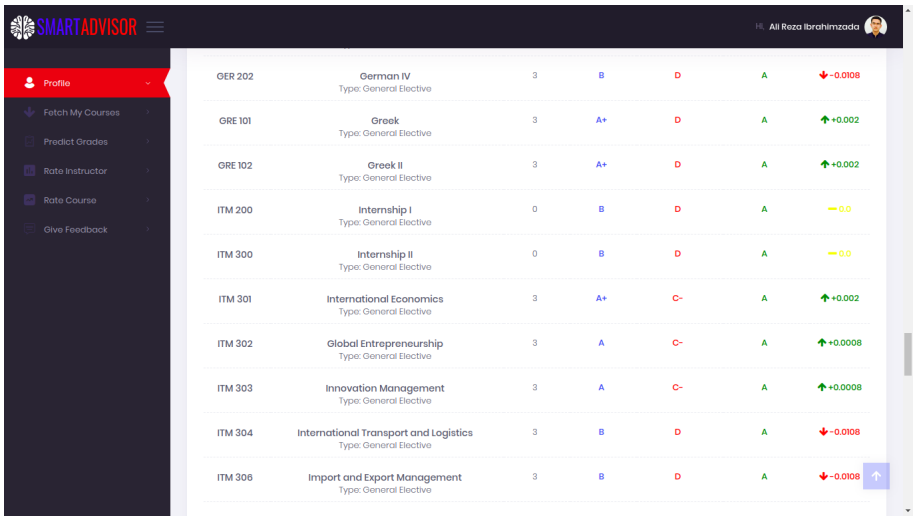

Fig. 17. Prediction Results

– **Rating Instructors**

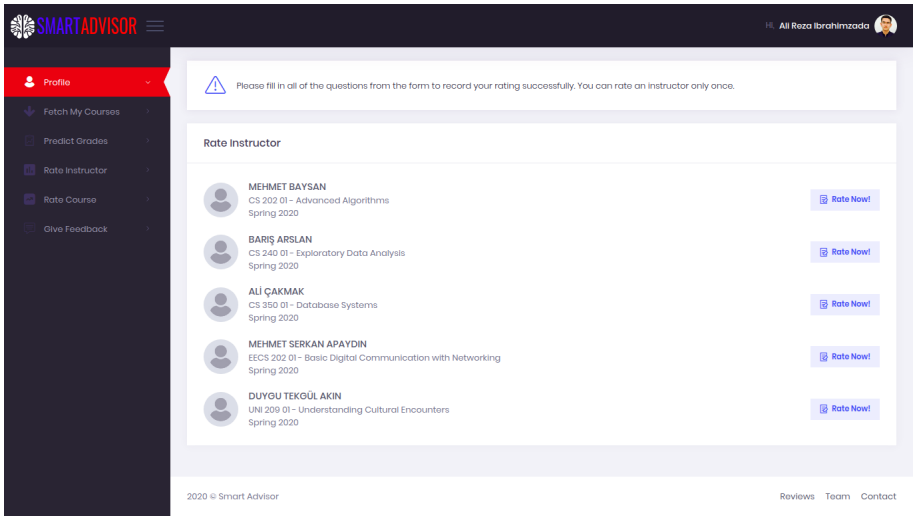

Fig. 18. Rate Instructors

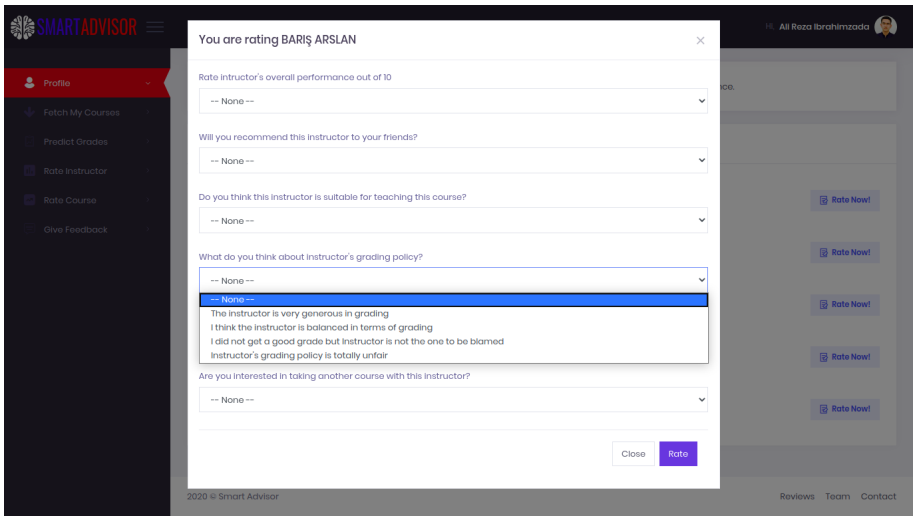

Fig. 19. Rate Instructors Questions

– **Rating Courses**

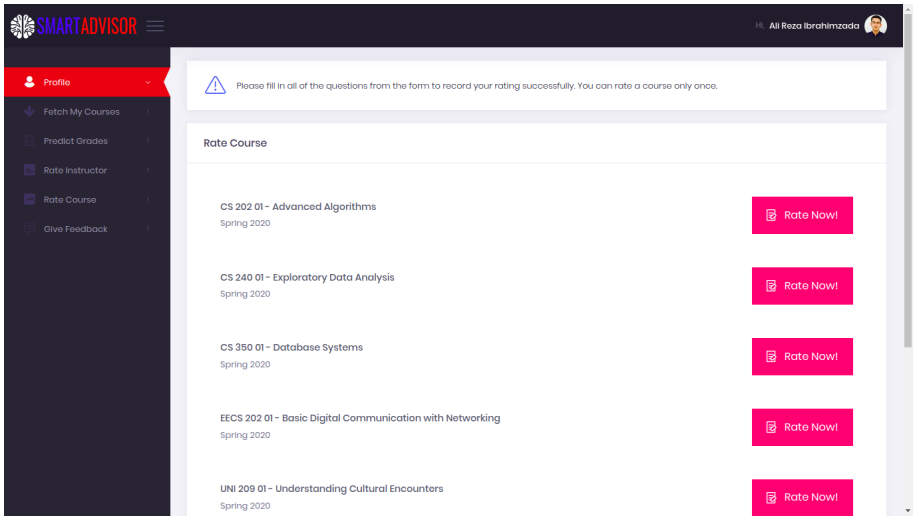

Fig. 20. Rate Courses

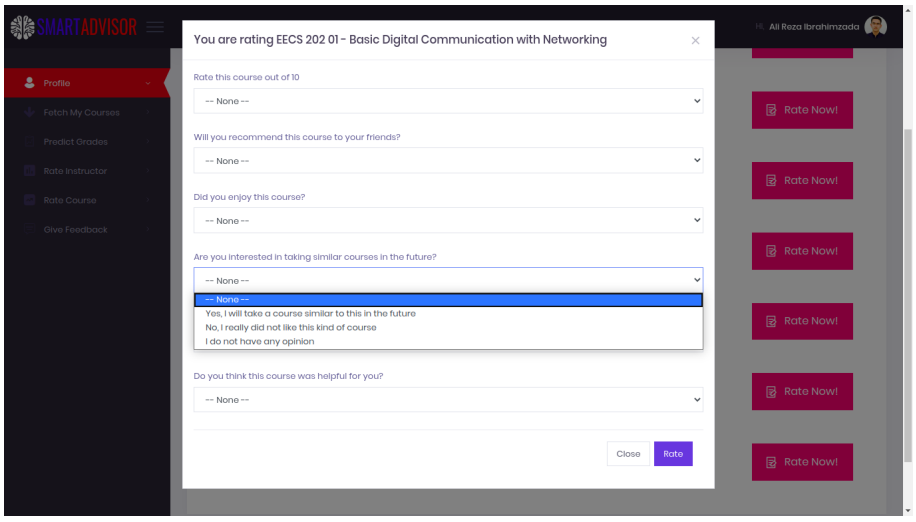

Fig. 21. Rate Courses Questions

– **Testimony**

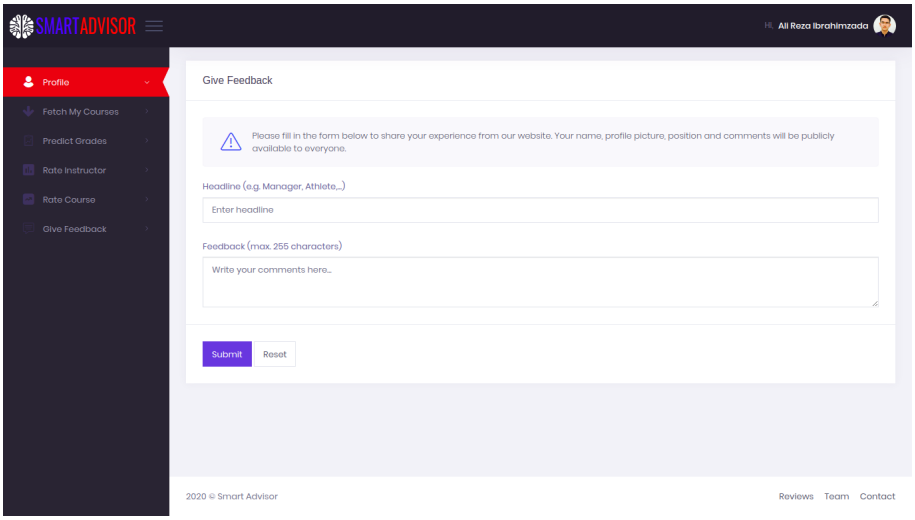

Fig. 22. Writing Feedback

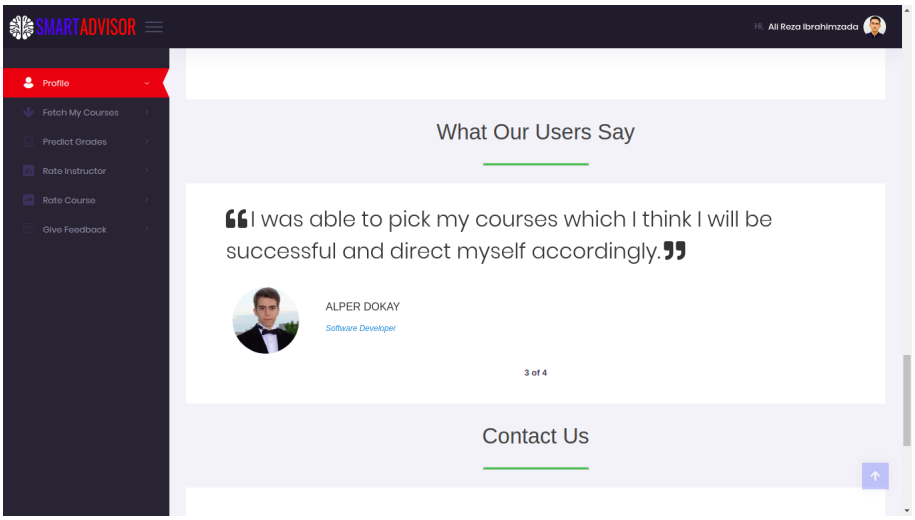

Fig. 23. User Testimony

– **Course Stats**

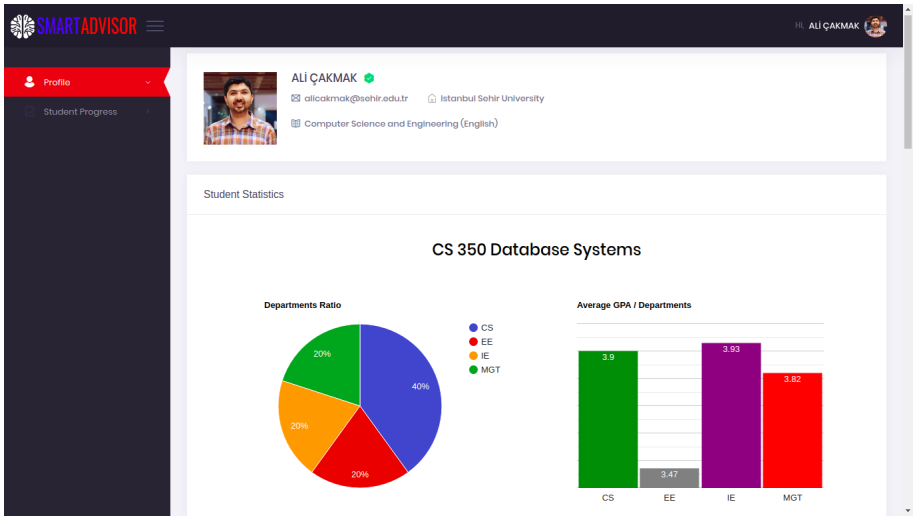

Fig. 24. Course Stats

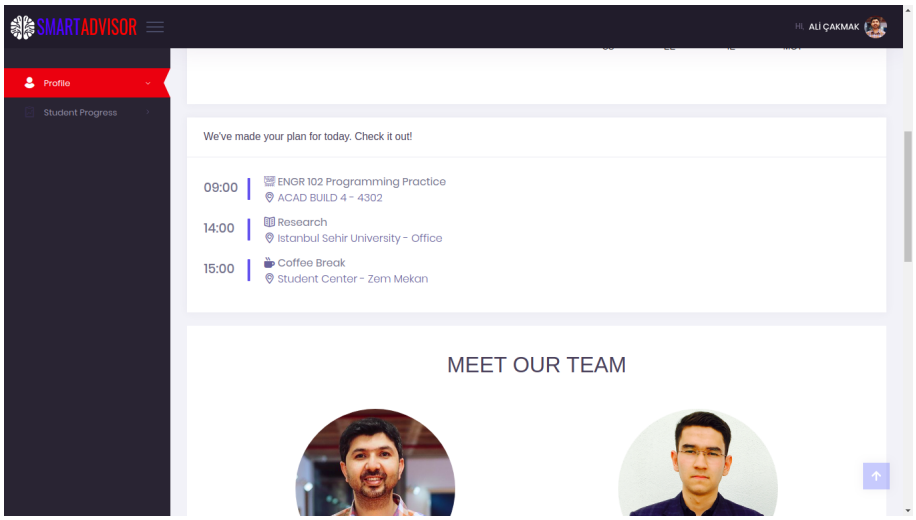

Fig. 25. Instructor Timetable

– **Advisee Progress**

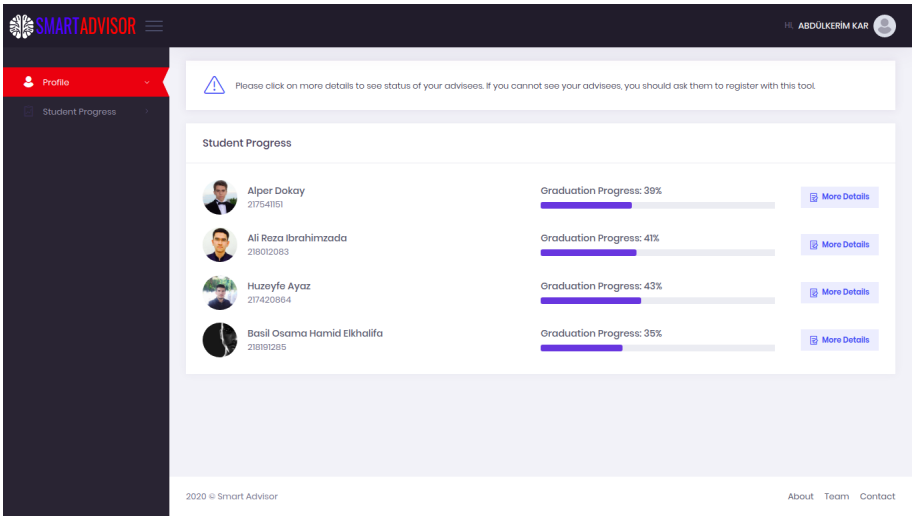

Fig. 26. Advisees

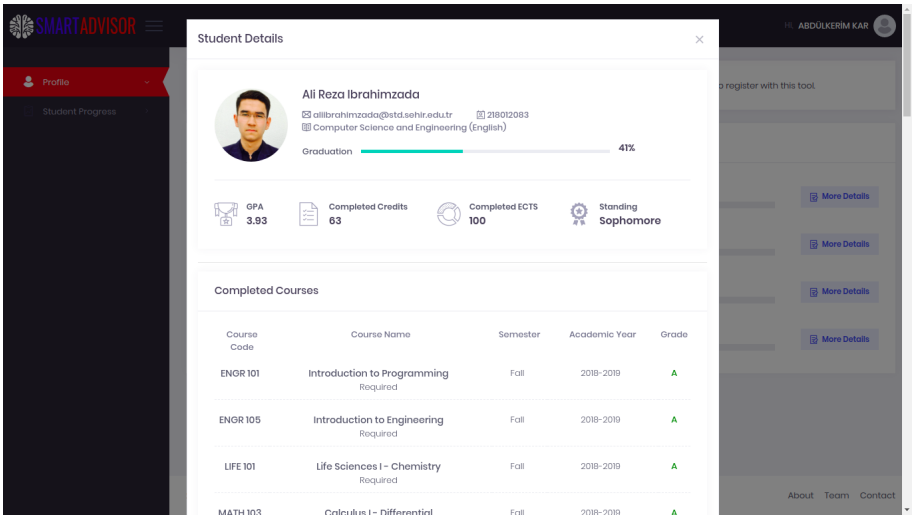

Fig. 27. Advisee's Profile

Fig. 28. Advisee's Profile

8.6 Non-Working Features

All application requirements mentioned in section *2* have been implemented correctly and are working properly.

9 Conclusion and Future Work

In this project we implement a tool for academic institutions in which students are able to predict their grades during the registration period. Advisors can also use this tool in order to track their advisees progress and suggest them some elective courses. Instructors are also able to observe the level of their class and change the course syllabus accordingly. In our future work, we will implement an Admin view to avoid fetching timetable and other processes from command line. We will go through a thorough study on Collaborative Filtering and implement it as yet another model to predict courses. Like it's necessary for every software project, we will write unittests in order to avoid any possible exceptions and errors.

Acknowledgements

We would like to thank Istanbul Sehir University for providing the dataset exploited in this work.

Bibliography

[1] Cakmak, A. (2017). Predicting Student Success in Courses via Collaborative Filtering. International Journal of Intelligent Systems and Applications in Engineering 5(1): 10-17.

[2] Imran M., Latif S., Mehmood D., Shah M. S. "Student Academic Performance Prediction using Supervised Learning Techniques" International Journal of Emerging Technologies in Learning Vol. 14, No. 14, (2019)

[3] Silberschatz, A., Korth, H. F., Sudarshan, S. (2019). Introduction to Database Systems. Database System Concepts, 1-33.

[4] Segaran, T. (2007). Introduction to Collective Intelligence, Making Recommendations. Programming Collective Intelligence, 1-28.

[5] Silberschatz, A., Korth, H. F., Sudarshan, S. (2019). Database Design using the E-R Model. Database System Concepts, 241-301.