ORIE 4741 Final Project - Predicting Cornell Course Enrollment Change During Add/Drop

Jolene Mei, Charlie Ruan, Joe Ye

***Abstract*—This project focuses on a subproblem within the larger prediction challenge faced by Cornell University: forecasting enrollment changes during the Add/Drop period to enable earlier scheduling of final exams. By implementing feature engineering techniques and assessing various machine learning classification models such as logistic regression, random forest, and gradient boosting, we have achieved a high level of prediction accuracy. Our models exhibit mean squared errors below 0.7 during random train/test split. This study contributes to addressing the enrollment prediction problem and demonstrates the potential for improving scheduling efficiency at Cornell.**

I. Background & Motivation

A group of undergraduate students advised by Professor David Shmoys (named Scheduling Team below) has been helping the University Registrar at Cornell to schedule final exams via integer programming models. Currently, the scheduling optimization model requires the student-level course enrollment information after the Add/Drop period as input. This data is typically not available until a month after the semester begins. However, if the team could take advantage of the pre-Add/Drop (or pre-enrollment) data instead of waiting until the end of the Add/Drop period, this process could be completed much earlier in advance, even before the semester begins.

Knowing students' drop activities allows the University Registrar to adjust the number of opening spots for each course. For instance, if the Registrar takes into account the estimated number of students who will drop the course, they can increase the capacity at the beginning of the semester. Thus, the final number of students can match the capacity, allowing more students to take the course they want.

In this project, we are interested in exploring the following question: given the pre-Add/Drop student enrollment information of a semester, can we predict the post-Add/Drop student enrollment of that semester? Due to the limitation of time and data, we will only focus on predicting the number of students *dropping* a course. Though predicting the number of extra students *adding* a course is a key component for predicting the total enrollment changes, it is out of the scope of this project.

The rest of the paper is organized as follows. We first examine the dataset available to us in Section II, then present our naive approach of treating the problem as a binary classification problem in Section III. We then present methods that we believe work the best on this problem in Section IV, and finally analyze our results in Section V.

II. Data Analysis

A. Data Description

The dataset, provided by the Cornell University Registrar, contains pre-Add/Drop and post-Add/Drop enrollment information for all enrolled students at Cornell University during the semesters of Spring 2022, Fall 2022, and Spring 2023. Each Excel sheet represents a snapshot of the enrolled courses for Cornell students on a specific date of the pre- or post-Add/Drop period. Each row in the Excel sheet contains a student-course pair, including the student ID, course subject, course catalog, course enrollment capacity, and other relevant information.

The dataset at our disposal is subject to certain limitations, primarily due to its restricted scope of past enrollment information, spanning only three semesters. Furthermore, we have identified inconsistencies in the time differences between the pre- and post-Add/Drop periods across these semesters. Notably, we have observed that the duration between the pre- and post-Add/Drop data for Spring 2022 and Fall 2022 is relatively shorter compared to that of Spring 2023. As a result, the dataset for Spring 2023 encompasses a more extended time difference period before and after Add/Drop, which may potentially lead to a higher number of course drops

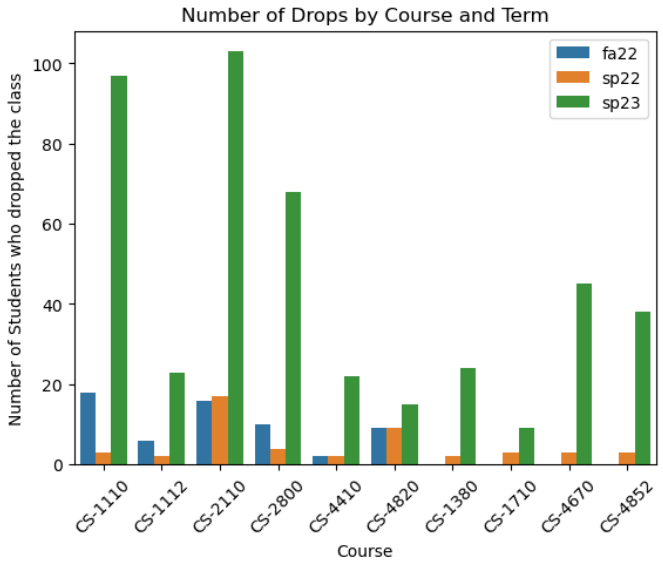

Fig. 1: **Number of drops of CS courses for each semester.**

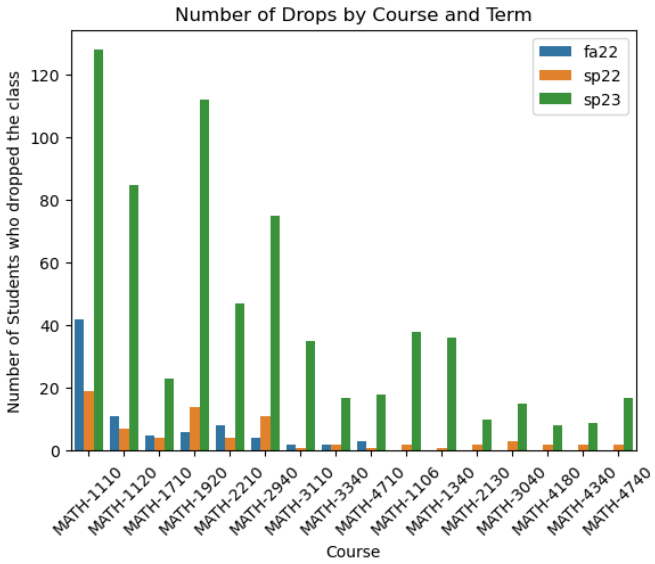

Fig. 2: **Number of drops of MATH courses for each semester.**

during that semester. We discuss more about the impact of these limitations in the following paragraphs.

B. Data Visualization

Acknowledging the data limitations, we have chosen to focus our data analysis on a specific set of Engineering or Engineering-related classes. These classes typically have larger numbers of enrollments, which can provide a richer insight into the students' course-drop situation. Consequently, we have chosen a list of 91 courses spanning the following departments: AEP, BME, CEE, CHEM, CS, EAS, ENGRD, ENGRI, ENMGT, INFO, MAE, MATH, ORIE, PHYS, and STSCI.

Figures 1-2 illustrate the number of drops for selected CS and MATH courses. These graphs reveal a significant number of drops across most classes (some being above 100 in Spring 2023), emphasizing the importance of our project goal. Furthermore, the graphs demonstrate that Spring 2023 exhibits a higher number of drops. This discrepancy can be attributed to the aforementioned data limitations. Note that some courses have zero drops in Fall 2022 because they are not offered in the fall.

C. Feature Engineering

We first pre-process the data by aggregating the data that pertains to the same student. Since we train a model for each course i, we then aggregate students that take course i into a single data set, so that each data point represents a student having course i before Add/Drop. The process involves dealing with ordinal and nominal data, missing values, and table joining and filtering.

We then create several essential features pertaining to students who enrolled in one specific class. These features include "num_course," indicating the total number of courses the student was enrolled in, "num_eng_course," indicating the number of engineering courses, "num_same_dept_course," indicating the number of courses within the same department that the student was enrolled in, and "course_level_sum," calculated by weighing each course with its course level (e.g. a 4000+ course contributes 4 unit to this metric). All of these features correspond to the student's schedule before the Add/Drop period.

The intuition here is that an increase in the number of course credits enrolled by students may result in a higher likelihood of class drops. Besides, if students are taking more same-department courses, it may suggest that the class is related to their major department, which could indicate a lower likelihood of dropping a course.

Figures 3 and 4 present a comparison between students who dropped the class and those who did not, based on the aforementioned features. The takeaway here is that

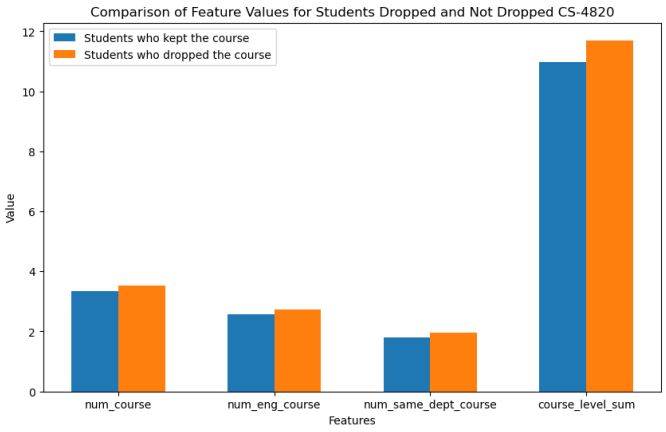

Fig. 3: **Comparison of Feature Values for Students Dropped and Not Dropped CS 4820.**

our feature engineering can indeed distinguish those who drop the course and those who do not.

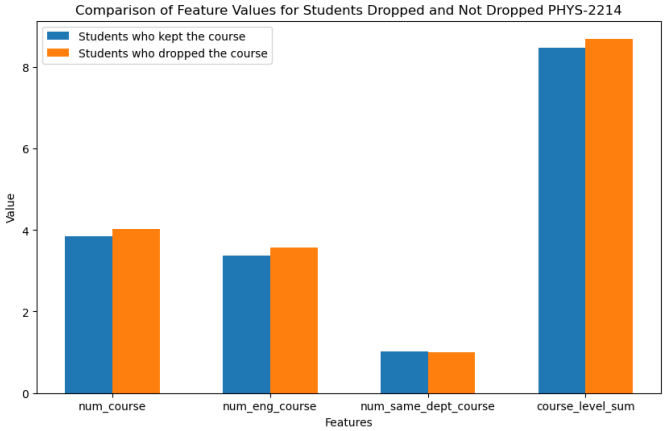

Fig. 4: **Comparison of Feature Values for Students Dropped and Not Dropped PHYS 2214.**

III. Problem Formulation

A. Problem Statement

In order to solve the problem with the available dataset, we need to predict the number of students that will drop the course post-Add/Drop given each student's pre-enrollment schedule for a semester. We plan to train a prediction model for each course. Specifically, for a given course i, the data sample consists of all the students that have enrolled in course i before Add/Drop, across all historical semesters, where each student is a data point. The features of a data point follow the discussion in Section II.B. The data point's label will be 1 if course i is no longer in that student's schedule after Add/Drop, and 0 otherwise.

B. Evaluation Metrics

The ultimate value that we care about is the course-level enrollment information; i.e., the total number of students dropping during Add/Drop for each course. To evaluate the performance of the prediction models, we propose to use the *squared error* (SE) as the key evaluation metric. The SE of a course can be calculated as the square of the difference between the predicted total number of drops (y_pred) and the actual total number of drops for that course (y_true):

$$SE = (y\_pred - y\_true)^2$$

Moreover, there are two types of SEs we will consider. The first one is the common out-of-sample SE after a random train/test split, where the size of the training data is typically set as 80%. Nonetheless, when we actually employ the model, we can only use the past semesters' data to predict the current semester. Therefore, to mimic and evaluate the real-world performance of our model, we also propose a "semester-specific" out-of-sample SE for each course. In the context of our dataset, we will use the Spring 22 and Fall 22 data as training data and the Spring 23 data as test data. It should be noticed that the semester-specific SE might not be the best metric for measuring the accuracy of our model since the dataset is imbalanced as shown earlier in Section II.A (the model will under-predict heavily as a result).

IV. Preliminary Approach

As mentioned in Section III.B., we will train a model for each course i. Each data point represents a student that enrolls in course i before Add/Drop. The input represents the student's schedule before Add/Drop. As a preliminary approach, we treat it as a binary classification problem and predict a zero or one for each data, ultimately summing up the predicted values to determine how many students will drop this course after Add/Drop.

To put it more formally, for each course i, we train a model h_i where given a student x's feature,

$$h_i(x) = \begin{cases} 1 & \text{if x drops i after Add/Drop} \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

Then, $\sum_{x \in X} h_i(x)$ is the total number of students dropping course i for the set of students X.

The models that immediately come to mind are support vector machine and decision tree, both of which are classic classification models that seemingly perfectly suitable for our job. However, as we will mention later, they are not the best approach to the problem.

A. Support Vector Machine

Support vector machine (SVM) is a powerful supervised learning tool that finds a hyperplane in an N-dimensional space to classify the data points, where N is the number of features [1].

There are a couple of hyperparameters that can be tuned: C, the kernel, and γ depending on the kernel used. C represents the penalty when the margin constraint is violated. While a large C makes the margin small, a small C allows some margin violation. On the other hand, a kernel helps transform the feature, which is especially helpful when the data is not linearly separable. Here we consider the radial basis function (RBF) and the sigmoid function. After performing the grid search according to Table I, we find that the parameter settings $(RBF, C = 1000, \gamma = 1)$ and $(RBF, C = 10, \gamma = 1)$ performed equally well, and here we pick $(RBF, C = 10, \gamma = 1)$.

Regardless, the result is not satisfying, as it results in a mean $SE = 20.132$ during testing. In the worst case, it can under-predict 15 students that dropped, as shown in Figure 5. Therefore, we conclude that SVM is not a suitable tool for our problem.

TABLE I: SVM Grid Search

Hyperpamaeter	Values to pick from
C	0.1, 10, 1000
γ	0.0001, 0.01, 1
Kernels	RBF, sigmoid, linear

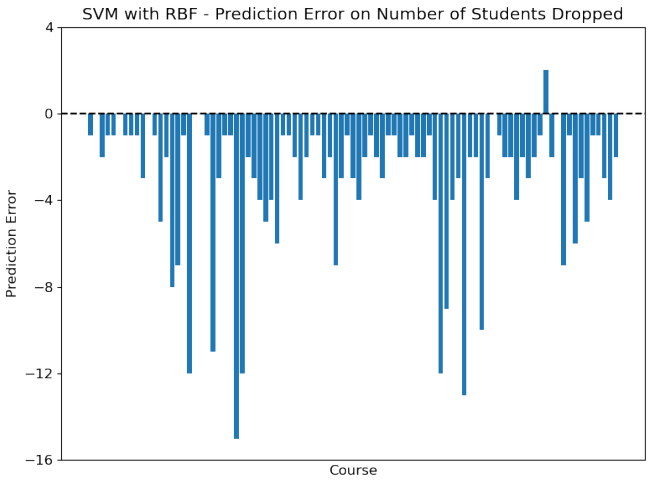

Fig. 5: **Support Vector Machine with RBF - random train/test split out-of-sample error.** Each course corresponds to a bar, which represents the number of students we predict to drop minus the actual dropped. We can see that we constantly under-predict (i.e. predict students that dropped will not drop).

B. Decision Tree

We then decide to attempt a model much different from SVM. A decision tree splits on a node based on a threshold for a feature. We classify data by traversing through the tree, using the assigned label on the leaf node the data ends on [2]. We utilize the default decision tree classifier in sci-kit learn, with the default Gini impurity metric. According to Figure 6, the prediction does not seem as bad as the support vector machine (it under-predicts 12 students in the worst case and has mean $SE = 14.824$). However, we are not satisfied with the performance and decide to explore other approaches.

V. Enhanced Modeling Approach

According to the mean SE and the histograms on the prediction error, the performance of SVM and decision trees are not ideal. We hypothesize that it is due to the fact that some students are on the edge of dropping and not dropping, but we force them to be a zero or one. In other words, SVM and decision trees cannot capture the tendency to drop, which should be a continuous value, or a probability, rather than a discrete binary value. Therefore, we decide to use models that can output a continuous value between zero and one, representing the probability that a student will drop the course.

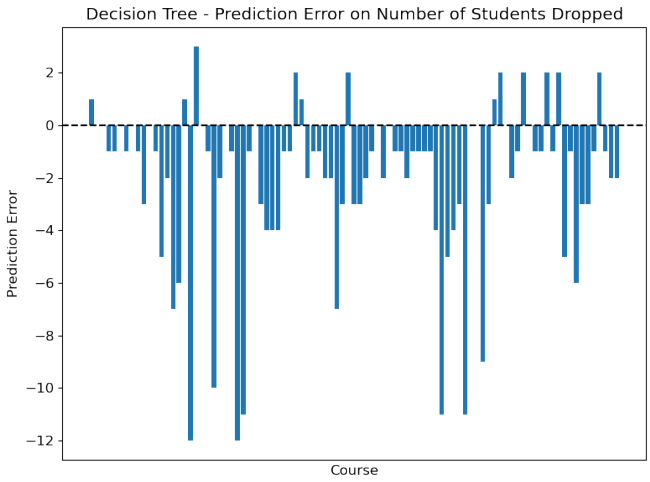

Fig. 6: **Decision Tree - random train/test split out-of-sample error.** Compared to the SVM's error histogram, the decision tree does not under-predict as often, though significantly more than over-predicting.

To put it more formally, for each course i, we train a model h_i where given a student x's feature, $h_i(x) \in (0, 1)$ is the probability that student x drops course i after the Add/Drop period. We then determine $\sum_{x \in X} h_i(x)$ to be the expected number of students dropping course i, which is a continuous value, unlike the previous section.

There are two fronts to explore this direction. Logistic regression intrinsically embodies what we want, as it outputs a distribution. On the other hand, ensemble methods output a continuous value in a different way by aggregating the outputs of a group of classification models.

A. Logistic Regression

Although logistic regression is a classification model that outputs the binary 0 and 1 labels, it inherently calculates a predicted probability for each individual data point by applying the logistic function to the linear combination of the independent variables and their coefficients [3]. By leveraging the *pred_proba* function in scikit-learn, we are able to obtain the probability distribution for every data point of a course's model. Summing up the probability of dropping the course over all data points will give us an estimation of the total number of dropped students for that course.

B. Ensemble Methods

The implementation and usage of the ensemble methods are similar to how we adopt logistic regression. Both methods described below have similar *pred_proba* functions in scikit-learn that can output the probability of dropping for each data point of a course. Therefore, the summation of the dropping probabilities can similarly give us the estimated number of dropped students in a course. More details of the two methods are provided in the following parts.

1) Random Forest: Random forest (RF) is a type of ensemble supervised learning method based on the results of multiple decision trees. RF adopts data bootstrapping and feature subsampling to reduce the correlation between weak learners [4]. The dropping probability estimation of each data point in RF is based on the mean of the predicted dropping probabilities of all trees. For each decision tree, the probability is calculated as the fraction of samples belonging to the drop label in a leaf [4].

2) Gradient Boosting: Gradient boosting (GB) is another ensemble supervised learning method that combines multiple weak learners into a strong one [5]. It does this by iteratively fitting new weak learners to the residual errors of the previous learners and updating the prediction model accordingly. The ultimate prediction probability for each data point is the end result of continually correcting the estimated probability at every stage [5].

C. Hyperparameter Tuning

In order to facilitate a more comprehensive comparison of the aforementioned prediction methods, we opted to enhance the performance of the best models for each course through hyperparameter tuning. This process involved defining a parameter grid, which consisted of a dictionary containing the hyperparameters, for each prediction method (i.e., LR, RF, and GB). Subsequently, we conducted a grid search over the parameter values and assessed each parameter set using cross-validation. To streamline this task, we utilized the *GridSearchCV* function available in scikit-learn. The performance of each

course prediction model was assessed by the negative squared difference between the predicted total number of dropped students and the actual total number of dropped students.

The parameter grid of each prediction method is outlined in Table II.

TABLE II: Hyperparamter Tuning Value Grids

Prediction Methods	**Hyperparameters to tune**
Logistic Regression	C: 0.01, 0.1, 1, 10 penalty: l_1, l_2 solver: liblinear, saga
Gradient Boosting	n_estimators: 50, 100, 200 learning_rate: 0.01, 0.1, 0.2 max_depth: 3, 5, 7 min_samples_split: 2, 4, 6 subsample: 0.8, 1.0
Random Forest	n_estimators: 50, 100, 200 max_depth: 3, 5, 7 min_samples_split: 2, 4, 6 max_features: auto, sqrt, log2

VI. Results & Discussions

The random train/test split and the semester-specific train/test split out-of-sample performances of the three prediction methods are plotted in Figure 7 - 9 and Figure 10 - 12 in the Appendix. In addition, we summarize the average performance of the three enhanced prediction methods over all useful courses in Table III.

The results of our plots are mostly aligned with our hypothesis stated earlier. Specifically, we obtain decent prediction results during random train/test split, with the prediction numbers off by at most 2 students among all courses for all 3 methods. Nonetheless, the performances of the 3 prediction methods are much worse during semester-specific train/test split. This is expected as discussed in Section II.

Moreover, it is noticeable that hyperparameter tuning achieves promising effects in reducing the mean SE in almost all scenarios except the semester-specific mean SE of logistic regression. This exception is understandable since we have observed earlier that the train and test data in the semester-specific setting is highly imbalanced. In addition, the cross-validation approach used in the grid search for parameter tuning is based on the out-of-sample mean SE of random train/test split; therefore, the semester-specific mean SE is not our target.

Judging from Table III purely based on the mean SE of each model, we can conclude that Logistic Regression performs the best on our available dataset. Gradient boosting also comes close after a huge improvement through parameter tuning. Random forest seems to have the highest mean SE but performs the best on semester-specific train/test split, which is a less important metric due to data limitation.

With regards to our confidence in the result, if there is more available data (spanning a longer time interval and including the actual start and end of the Add/Drop dates), our model can safely be used in real applications, given how well it already performs on such limited data (only under or over predicting less than 3 students for each course as shown in Figures 7-9). However, the fact that we have never successfully predicted using the semester-based data splitting is indeed worrisome, but we expect that it can be fixed with more data availability as discussed in Section II and Section III.

TABLE III: Model Mean SE Before vs. After Parameter Tuning

	Before	After
LR-semester	547.252	554.690
LR-random	0.350	0.330
RF-semester	556.62	551.532
RF-random	1.021	0.664
GB-semester	560.152	556.318
GB-random	1.462	0.337

VII. Conclusion

Using techniques we learned in class, such as feature engineering, support vector machine, decision tree, logistic regression, and ensemble methods, we are able to obtain promising results to predict the number of dropped students for each course. Specifically, all 3 prediction methods we ultimately proceed with (logistic regression, random forest, and gradient boosting) achieve a high prediction accuracy during random train/test split,

resulting in average mean squared errors that are less than 0.7.

Admittedly, there are still some limitations to this project. First of all, the dataset, as mentioned in Section II, is imbalanced due to inconsistency in the duration between the pre- and post- Add/Drop period across semesters. We believe that by accumulating more data over an extended period, our models would be capable of making even more accurate predictions. In the future, the Scheduling Team should coordinate with the Registrar to obtain more snapshots of student-level enrollment information across more semesters to better represent the trend of enrollment changes. Moreover, there are additional features that could be potentially included in the prediction models. For example, if given more historical enrollment information, we can track down the course histories of individual students.

It should be noted that this project is the first attempt to address the larger enrollment prediction problem encountered by the University Registrar and the Scheduling Team, and we only contribute to part of this larger prediction problem. To completely tackle this problem, we need to also consider the "add" aspect of enrollment, which is to predict the number of students enrolling in each course during Add/Drop. This classification model should be similar to the "drop" classification model proposed in this project but with more aspects to consider such as the lecture/discussion time conflict between courses. In addition, the scheduling optimization model requires also the input of higher levels of course-level information, such as pairwise co-enrollments and triplet co-enrollments. Those are similar but essential subproblems that can form the piece of the ultimate predict-then-optimize scheduling problem.

A potential fairness concern is how a model may perform better for certain courses than others. For instance, as mentioned in Section II, our model performs well on courses with a larger number of students (since it can have more data to train with). This results in students who take small-size classes (e.g. philosophy-major students) may not enjoy the benefit of our model (e.g. knowing the number of students that will drop ahead of time). A solution to such bias is simply to accumulate more data, which allows the model to perform well even on small-size courses.

Regular evaluations of the model's fairness should be conducted to identify any biases that may arise. This involves analyzing the predictions and outcomes across different demographic groups or other relevant factors, such as race, gender, or socioeconomic status. If any disparities are detected, appropriate adjustments or interventions should be implemented to mitigate the biases and promote fairness.

VIII. Contributions

Jolene Mei was responsible for Section II of the paper and the relevant work. Charlie Ruan was responsible for Section IV of the paper and the relevant work. Joe Ye was responsible for Section V of the paper and the relevant work. For the corresponding work to the remaining sections, each member contributed equally.

References

[1] J. Platt, et al., Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods, Advances in large margin classifiers 10 (3) (1999) 61–74.

[2] F. Vega, J. Matías, M. Andrade, M. Reigosa, E. Covelo, Classification and regression trees (carts) for modelling the sorption and retention of heavy metals by soil, Journal of Hazardous Materials 167 (1-3) (2009) 615–624.

[3] H.-F. Yu, F.-L. Huang, C.-J. Lin, Dual coordinate descent methods for logistic regression and maximum entropy models, Machine Learning 85 (2011) 41–75.

[4] L. Breiman, Random forests, Machine learning 45 (2001) 5–32.

[5] J. H. Friedman, Greedy function approximation: a gradient boosting machine, Annals of statistics (2001) 1189–1232.

IX. Appendix

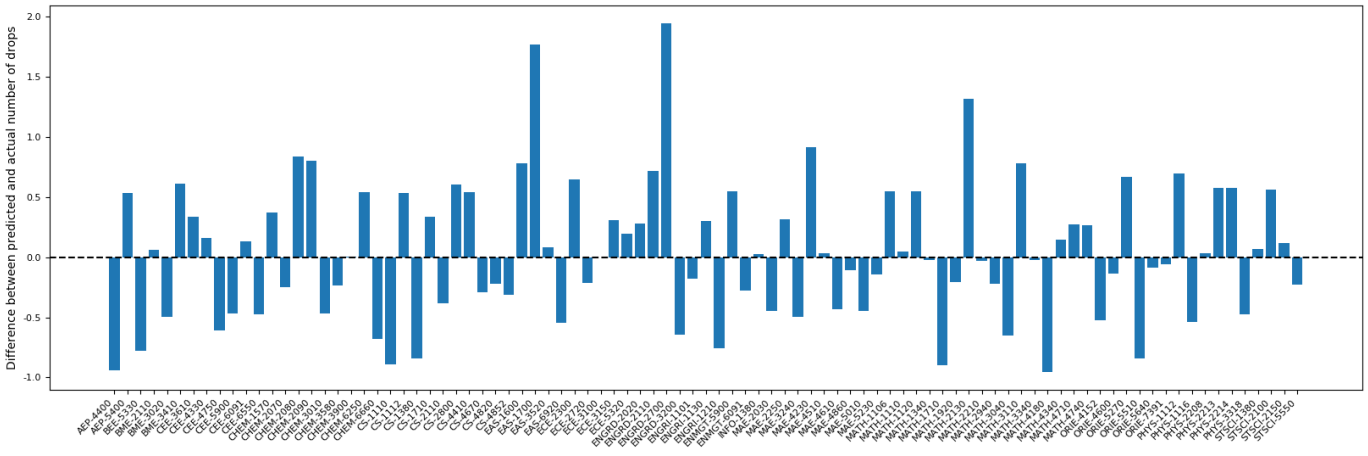

Fig. 7: Logistic Regression - random train/test split out-of-sample error

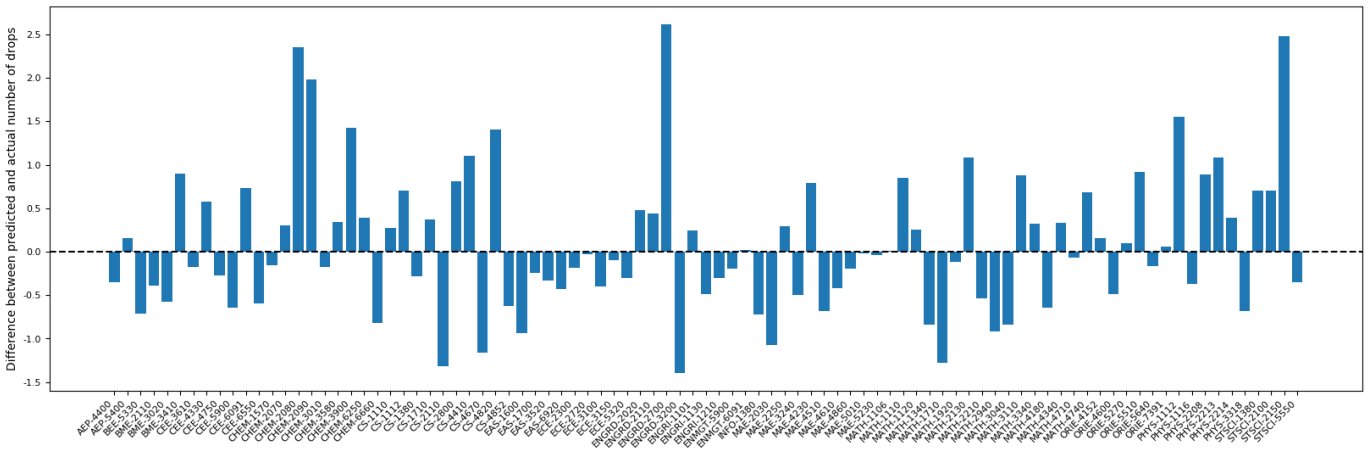

Fig. 8: Random Forest - random train/test split out-of-sample error

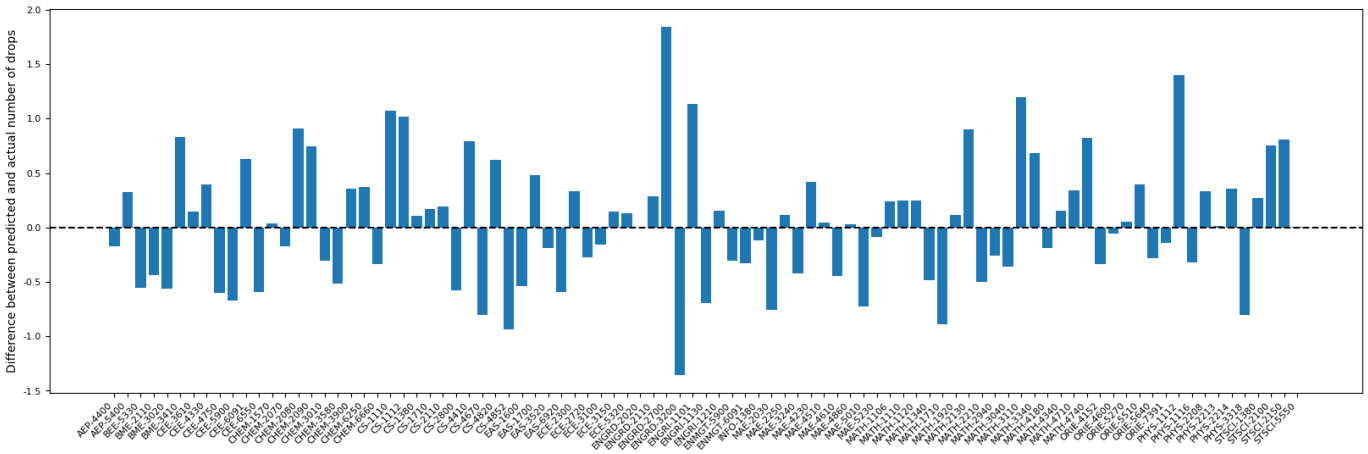

Fig. 9: Gradient Boosting - random train/test split out-of-sample error

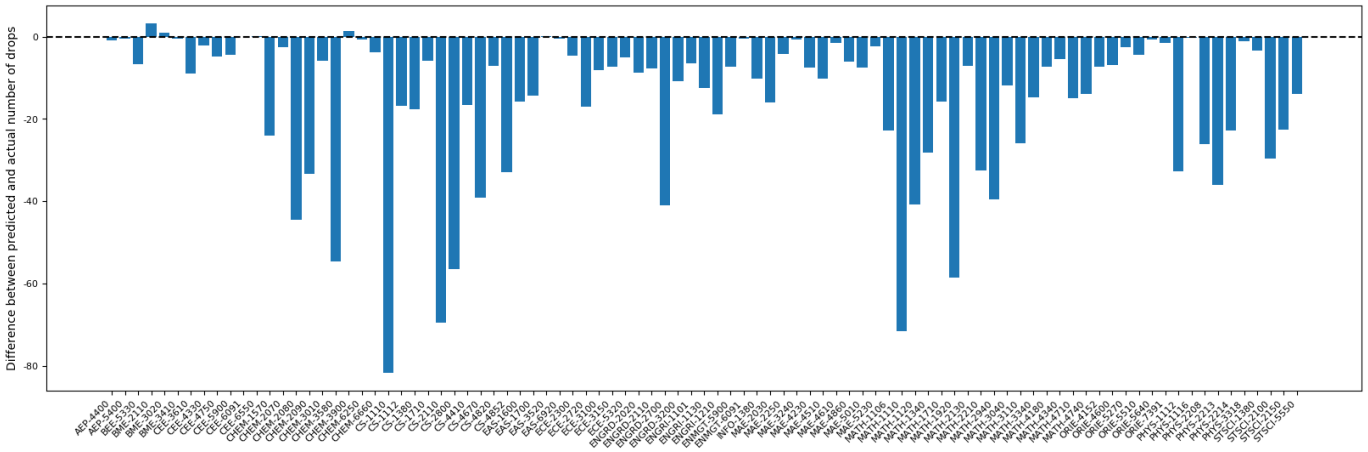

Fig. 10: Logistic Regression - semester-specific train/test split out-of-sample error

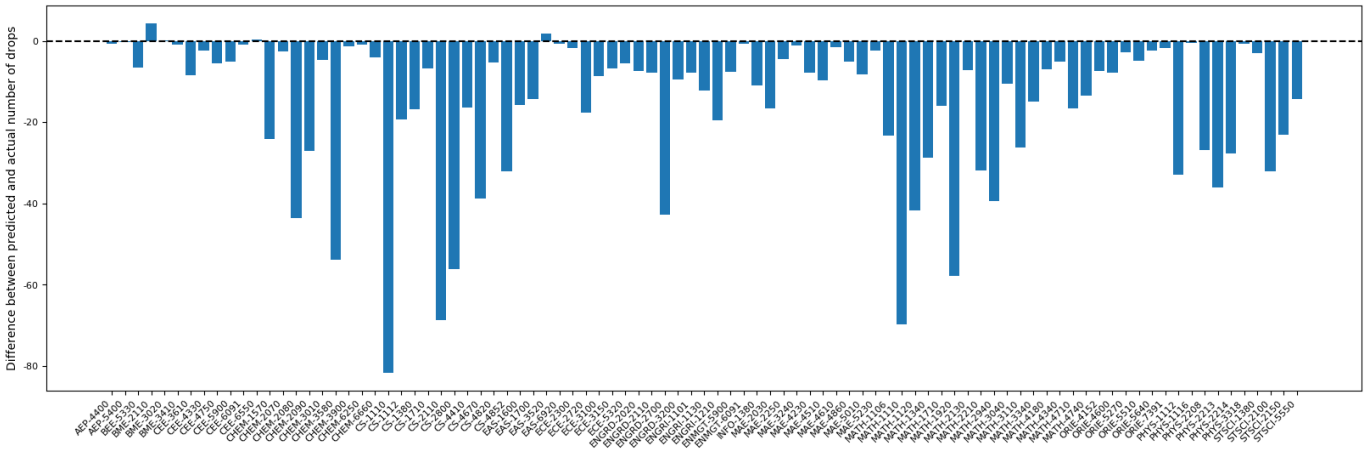

Fig. 11: Random Forest - semester-specific train/test split out-of-sample error

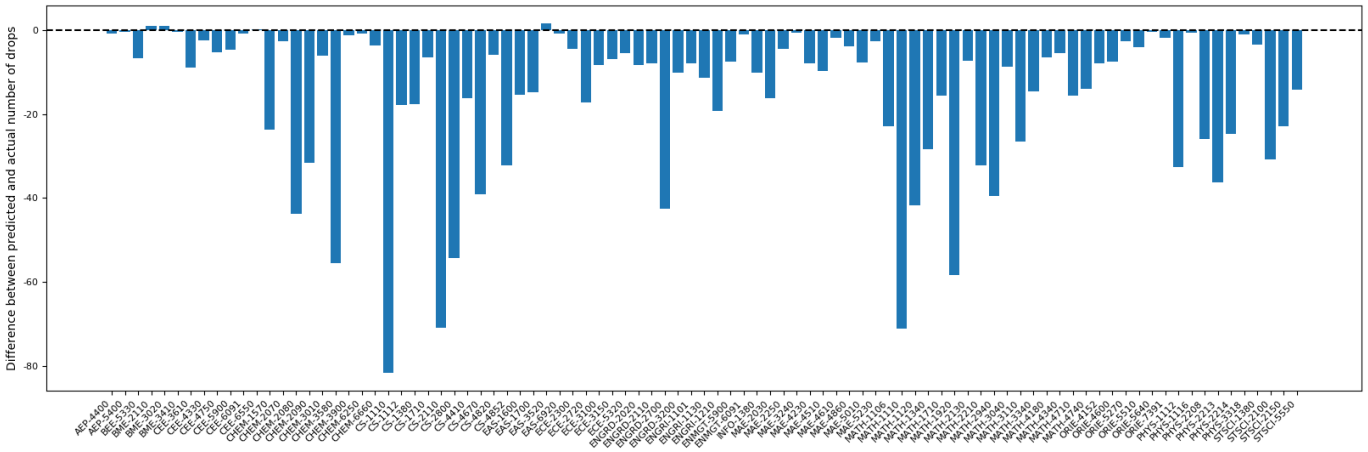

Fig. 12: Gradient Boosting - semester-specific train/test split out-of-sample error