MACHINE LEARNING AS A TOOL TO IDENTIFY CRITICAL ASSIGNMENTS

Jeffrey J. Heys
Montana State University • Bozeman, MT

The goal of this article is to demonstrate the use of machine learning (ML) as a tool to potentially identify key assignments or critical sections in engineering courses. The article begins with a brief overview of the field of ML, including some of the tools and terminology used in ML. The discussion is not intended to be comprehensive but, instead, to provide some common terminology and highlight some of the more common ML techniques for someone with a background in engineering. The main topic here is an illustration of how ML tools can provide insight into grade analysis, student assessment, and the relative importance of different course components in predicting the overall course performance of a student.

Courses on ML have existed for decades at many academic institutions.[1,2] ML is considered a sub-field of computer science and the students in ML courses are likely to be in computer science or a closely related field such as data science. It is easy to document the widespread use of ML tools for solving problems in computer science such as spam filtering and imaging recognition, and there is recent evidence that ML use is growing in chemical engineering.[3,4] Growth in the range and diversity of problems where ML approaches are being applied should cause almost anyone to pause and consider the potential role of ML in the future. Before discussing grade analysis and prediction, a brief overview of ML is provided.

OVERVIEW OF MACHINE LEARNING

If it is possible, machine learning—which includes artificial intelligence and deep learning as closely related and overlapping fields—is both over-hyped and underappreciated simultaneously.[5] The mathematical algorithms that form the foundation of ML go back more than a century (*e.g.*, optimization of an objective function, regression, and clustering). Critics point to the deep history of the underlying mathematics as evidence that the recent surge in ML interest is merely hype. However, this fails to recognize extraordinary improvements made by ML researchers on new algorithms for ML that are more flexible, more adaptable, easier to implement, easier to use, require minimal mathematical training, and can be executed on very large data sets (*e.g.*, millions of data points) using only modest, inexpensive computational hardware. The first two major applications of ML were optical character recognition in the 1980s and email spam filtering that started in the 1990s.[6] Research in ML actually declined during the first decade of the 21st century, but ML interest has exploded with the recent demand for speech recognition, image recognition, self-driving vehicles, and other related technologies. Simultaneously, the availability of graphical processing units (GPUs) to support the computational burden and large datasets provided by the internet were significant enablers of ML.[7,8] The use of ML in engineering applications is relatively recent, but ML methods including support vector machines and artificial neural networks for chemical process modeling are growing rapidly, especially in areas where large datasets are now available.[3,4]

***Jeffrey J. Heys** is a professor and department head in the Chemical and Biological Engineering Department at Montana State University. He received his B.S. in chemical engineering in 1996 from Montana State University, and his M.S. and Ph.D. from the University of Colorado at Boulder in 1998 and 2001, respectively. His research area is computational transport and computational fluid dynamics in biological systems with an emphasis on fluid-structure interaction and porous media flows.*

ML approaches can be roughly divided into two categories: supervised learning and unsupervised learning.[9] Supervised learning requires an existing dataset with the values of the desired output variables included. Optical character recognition is an example where supervised learning is used. The data typically consists of a pixelated image of a single handwritten character, *i.e.*, the data is a vector of pixel color values for every pixel in the image and the corresponding character shown in the image is given. In ML, the color value of each pixel is called a "feature" and the true character shown in the image is called the "label." In short, supervised learning requires labeled input data, and the result after training is an algorithm that can predict labels for future, unlabeled, input data. The most common approaches for supervised learning include:

- ***Multivariable Regression***[6]*: the large datasets with many features (*i.e.*, independent variables) in typical ML applications can lead to very large-scale multivariable linear regression problems (and, sometimes, multivariable polynomial or nonlinear regression problems) where the minimization of the objective function is usually performed using a gradient descent algorithm because the data sets are too large for a direct solution algorithm.*
- ***Logistical Regression (or Logit Regression)****: this approach starts with the use of linear regression to predict a probability that some set of features from a data set should have a certain label applied. For example, the probability that a vector of pixel colors corresponds to the letter "b." Then, the logistic of the probability is taken (*i.e.*, a sigmoid function is used) to determine the final label that is applied. Hence, logistical regression is an extension of linear regression so that it can be used as a classification technique.*[7]
- ***Decision Trees****: these algorithms build decision trees by first identifying the most important features in the data (*e.g.*, the most important pixels in an image for determining the character in the image) and then building decision trees for classification.*[9]
- ***Support Vector Machines (SVM)****: these algorithms identify boundaries that optimally separate data based on the different labels connected with each item in the data set. For example, the pressure, temperature, opacity, and other data might be collected for various steps in a chemical process, and then the final product is labeled as "off-spec" or "acceptable." An SVM algorithm could be used to determine the boundaries between conditions that frequently lead to the different outcomes so that the final product quality could be predicted in the future as the process measurements are being made at each step in the process.*[7]

Beyond the basic learning methods mentioned above, much of the research in ML today is directed towards the use of artificial neural networks (ANN) and other deep learning approaches. These methods use multiple, hidden layers of inputs and outputs to develop networks with more flexibility and the potential to provide significantly greater predictive power than methods with a single layer of weights. For example, imagine if the output of a multivariable linear or nonlinear regression model was used as the input for another, hidden, linear or nonlinear regression model. The resulting network would have twice as many parameters and would require significantly more data for training, but it would have the potential to provide a better model of the data. Deep learning approaches are beyond the scope of the current discussion, but they are some of the most promising and exciting methods in ML when *very* large quantities of data are available.

Unsupervised learning uses clustering algorithms to identify related or correlated features among the different samples in a dataset.[6] These algorithms do not require that the data be labeled or that a result is known. For example, a website might have data on the various items individuals purchase. The website could use a clustering algorithm to recognize that individuals that purchased shoes also frequently purchased socks and then use that information to advertise socks to every future customer searching for shoes. The most common clustering algorithms include k-means and hierarchical cluster analysis. Principle component analysis is also an unsupervised learning algorithm used for dimensionality reduction with large datasets. It is the author's opinion that the potential applications for supervised learning in chemical engineering are significant while the applications for unsupervised learning are probably limited.

There are a number of commercial software packages for ML, but the field is dominated by open source libraries that support a number of different programming languages, including C++, Java, R, and MATLAB, but the most popular language in the field of ML is Python.[11] Some of the most widely used libraries are Torch,[12] Caffe,[13] Keras,[14] Therano,[15] TensorFlow,[7] and Scikit-Learn.[7] Most libraries provide tools that help with importing data, preprocessing and scaling the data, multiple ML algorithms for supervised and unsupervised learning, and postprocessing algorithms. The libraries include computationally efficient implementations with some even supporting parallel execution of many calculations on a GPU.

MACHINE LEARNING FOR GRADE ANALYSIS AND PREDICTION

Many studies have been published on the topic of grade prediction, including the prediction of grades in engineering programs.[16,17] Most of these studies focus on the identification of factors that can be used to predict students that might need additional support in order to be successful in their engineering program.[18] While most of these studies used traditional statistical analysis to identify differences (*e.g.*, t-tests) and trends (*e.g.*, linear regression), a few recent studies have examined the use of ML algorithms to predict students' overall GPAs at graduation[19,20] or student retention.[21,22] The only examples of using ML to predict performance in a single class that were found were a pair of studies on predicting student performance in distance learning courses by Kotsiantis *et al.*,[23, 24] and no publications on predicting engineering course performance using ML were found. The focus here is on both predicting the final grade in a chemical engineering course—based on assessment elements like homework or quizzes within the course—and using the results of the ML analysis to identify especially important elements within the course. This application illustrates the use of ML in modeling and it illustrates how one can gain insight into key factors (*e.g.*, important assignments) affecting course grades. The goal here is not to develop an extremely accurate tool for predicting course grades, but to illustrate the use of ML in identifying particularly important assignments or key moments in a course.

The application of ML to grade prediction is illustrated through a step-by-step example below. The computer code uses the Python 3 programming language, and three additional libraries are used:

- ***Numpy*** *<www.numpy.org>: this library adds additional vector, array, and linear algebra tools for use in Python programs. The tools are computationally efficient as most of the underlying computer code is written in FORTRAN or C.*
- ***Pandas*** *<pandas.pydata.org>: the Pandas library is designed specifically for data science on large data sets and includes extensive tools for reading and writing data from/to different sources and then statistically analyzing (and modifying) the data.*
- ***Scikit-Learn*** *<scikit-learn.org>: this is the most important library for this discussion as it adds the machine learning tools that are illustrated below in the Python language.*

The full Python code (and Jupyter notebook, <jupyter.org>) and anonymized grade data sets that are illustrated below are available at <www.chbe.montana.edu/heys/mlandche>. All names and other identifiable information have been removed from the data sets and a few of the grades have been randomly modified by a small amount. An exemption was granted by the Montana State University Institutional Review Board for the data set. The interested user who wants to explore the example code below, but is unfamiliar with Python, is encouraged to download the Anaconda Python distribution (<https://www.continuum.io/downloads>) for MacOS or Windows, as it includes Python and additional libraries that are typically used in data science or engineering.[25]

IMPORTING DATA

The data for this first example is all the grades from a course on computational methods for chemical and biological engineers taught at Montana State University. Two different data sets are examined from two different years of teaching the course, but the data sets are not merged together because of changes in the course. The graded assignments in the course include:

1. *Ten homework assignments that cumulatively accounted for 40% of the final grade. Note that the each homework assignment grade is out of 10 possible points and the final grade percentages in the spreadsheet reflect the final percentage after dropping the lowest homework score.*
2. *Two midterm exams that each account for 25% of the final grade. The midterm exams are graded out of 100 points, and additional points are sometimes included so exam scores of more than 100 points are possible.*
3. *A final project that accounts for 10% of the final grade and is graded out of 10 points.*

The course data sets have 152 students in year A and 146 students in year B. As expected, not all assignments and exams were completed by all the students so there are many blanks in the data set that will ultimately be equivalent to zero points.

Fortunately, the Pandas library supports the importing of data from Excel files. The following Python code imports the full course grade data set for year A from file "GradesA.xlsx."

```
import pandas # import the pandas library for read-
ing the data files
import numpy
gradeData = pandas.read_excel(open("GradesA.
xlsx",'rb'),sheetname='Sorted')
gradeData = gradeData.fillna(0) # fill missing grades
with zero
```

The first two lines import external libraries, and a single line of code reads the data set from an Excel spreadsheet using the read_excel() function in the Pandas library. The final line of code above fills in the missing data points, *i.e.*, unsubmitted assignments, with zeros.

LINEAR REGRESSION

Since most chemical engineers are comfortable with multivariable linear regression, that is a good place to begin illustrating ML algorithms. The code below performs linear regression on just the nine best homework grades for each student (stored in matrix X and referred to as the <u>features</u>) in order to predict the final course grades (stored in vector y and referred to as the <u>labels</u>).

```
from sklearn.linear_model import LinearRegression
X = gradeData.loc[:,'Homework 1':'Homework 10'] #
features
y = gradeData['Final Pct'] # labels
lin_reg = LinearRegression() # setup for multivari-
able regression
w = lin_reg.fit(X, y) # perform regression
print("Weights",w.coef_)
```

The linear regression algorithm in Scikit-Learn, LinearRegression(), minimizes the root mean square error just like traditional linear regression, and the minimization of alternative objective functions is straightforward. The weights, stored in variable w, correspond to the coefficients (or slope) associated with each independent variable (or feature). Each retained homework assignment is equally important in calculating the final grade *(i.e.*, each homework assignments represents 4.4% of the final course grade), but they are not equally important for predicting overall performance in the course. The weights from linear regression for the 10 different homework assignments are: 0.90, 0.91, 0.89, 0.16, 0.72, 1.03, 0.50, 0.83, 1.08, and 0.41, and they tell a different story. If an instructor wants to predict final course grades, the fourth and tenth homework assignments are almost useless based on the small weights of 0.16 and 0.41, respectively. The sixth homework assignment, on the other hand, is very important in predicting final grades and students may want to pay special attention to their grade on this assignment if they want to predict how they might finish the course. The mean scores on the 10 different homework assignments varied by less than 10% over the semester so the average assignment score is less variable than the linear regression weights. It is also recommended that the standard correlation coefficient matrix be checked because the linear regression weight for any assignment with low correlation may not be meaningful. For this data set, all the correlation coefficients between individual homework assignments and the final grade percentage were between 0.5 and 0.7 with the exception of homework 1, which had a value of 0.41. Homework 1 often correlates poorly because students are unfamiliar with instructor expectations.

The second data set from the same course but a different year is stored in the file "GradesB.xlsx" and has similar variability for the weights corresponding to each assignment. The trend in data sets for two separate offerings of the course suggest that middle-of-the-semester homework assignments falling roughly in weeks 6–8 of a 15-week semester are particularly influential if multivariable linear regression is used to predict final percentage grades.

There are a number of gaps in the analysis above. First, the accuracy of the grade predictions using only homework scores and leaving out exams and a project has not been analyzed. Are homework scores a good predictor *(i.e.*, do they correlate strongly) or should we focus only on exam scores? If we repeat the analysis above using just the two midterm exam scores to predict the final grade and leave out the project and homework scores, we find that for both years the second midterm is twice as important compared to the first midterm in predicting the final grade in the class. Was the first midterm too easy? The average grade was only slightly higher than the second midterm for both years so there is no evidence for the "easy" midterm. Finally, nothing in this analysis is new or groundbreaking. However, the tools provided by ML did make the importing of data and analysis simple and straightforward, and simplicity becomes even more important as the data sets and analysis become more complex.

LOGISTIC REGRESSION AND TEST SETS

To put the example above in more of an ML framework, two changes are helpful. The first change is to divide the data into a training set and a test set. The majority of the data (70-80%, typically) is put in the training set and used to determine the regression weights. The remaining data is placed in the test set and used to assess the predictive ability of the ML model. (n.b., ML practitioners will often split the full data set into three sub-sets: a training set, a test set, and a validation set. The reason for this is that the lengthy process of tuning the model often results in a model that is especially well tuned to the test set because the test set is always used for assessment. The validation set provides a data set that should only be used to assess the quality of the final model and should never be used for improving/tuning the model.) The second change is that the model is now used to predict the final letter grade of A, B, C, D, or F instead of a final percentage. Hence, the model is now a classifier as it classifies the input data as that associated with a typical A grade or F grade, for example. In ML, characters and strings such as letter grades, words, or phrases are typically encoded to unique numbers (normally integers), which are required by most ML algorithms and minimizers. The encoder used below replaces the letter grades A, B, etc. with the integers 0, 1, etc. (n.b., This simple encoding works fine in this example, but there are more complex ML applications where this encoding is unfavorable because integers have an order and their use suggests an ordering to the labels that may not exist. For these cases, an encoding called "one hot encoding" should be used instead.) The code below illustrates the full process of splitting the data into training and test sets along with encoding.

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
trainData, testData = train_test_split(gradeData, 
test_size=0.2)
encoder = LabelEncoder() # setup label encoder
X_train = trainData.loc[:,'Homework 1':'Homework 
10']
y_train = encoder.fit_transform(trainData['Final 
Grade'])
X_test = testData.loc[:,'Homework 1':'Homework 10']
y_test = testData['Final Grade']
log_reg = LogisticRegression() # setup for logit 
regression
w = log_reg.fit(X_train, y_train) # perform regres-
sion
gradePredictions = log_reg.predict(X_test) # check 
model performance on test data
```

The performance of the Logistic Regression model can be assessed using the test data in a number of different ways. One of the simplest methods is to calculate the root mean square error (RMSE) on the test set (20% of the class or

30 students) and this is typically 5–8% or a little less than one letter grade. In other words, for this class, we can predict the final grade correctly about two-thirds of the time using only the homework scores. Homework scores could be a much better or a much poorer predictor for other courses. Another assessment is to just look at the predictions for the first 10 students in the test set and compare their predicted grades to their actual grades as shown in Table 1. In only 10% of the cases did the prediction miss by more than a single letter grade and in about 50% of the cases, the prediction was correct. (n.b. Even if all available grades are used in the prediction—all homework, midterm exams, and project grades—the prediction is still incorrect about 1% of the time because the model does not account for the dropping of the lowest homework score when calculating the actual final percentage.) If +/- grading is used, the prediction would be incorrect more frequently due to the larger number of possible value, but the average size of the error (*i.e.,* RMSE) would be similar.

TABLE 1
Predicted and actual grades for 10 students from the test set. The two different data sets are for two different years, and logistic regression was used for the predictions.

Data Set: GradesA.xlsx										
Predicted:	D	B	B	A	A	A	B	A	C	A
Actual:	D	B	F	B	A	A	A	B	C	B
Data Set: GradesB.xlsx										
Predicted:	C	A	F	A	B	F	D	B	A	B
Actual:	B	A	F	B	B	F	C	C	C	B

TABLE 2
Predicted and actual final grades using homework scores and a decision tree algorithm. The data sets are from two different years.

Data Set: GradesA.xlsx										
Predicted:	B	B	B	A	A	A	B	A	F	A
Actual:	D	B	F	B	A	A	A	B	C	B
Data Set: GradesB.xlsx										
Predicted:	B	A	F	A	A	F	B	B	A	B
Actual:	B	A	F	B	B	F	C	C	C	B

DECISION TREES

One of the advantages associated with ML maturing as a research field is that the current libraries make it easy to try out different types of algorithms quickly and easily to determine the best choice for a particular data set and modeling project. Decision trees are typically used for classification problems like the letter grade prediction example above. The algorithm first determines the most important feature (in this case, the most important homework assignment) for predicting the labels or final letter grade, and then creates a branch based on the optimal dividing threshold for that feature as determined by the algorithm. Additional layers of branches can be added to the decision tree as desired.

The code below is very similar to the previous logistic regression code, but it uses the decision tree algorithm from Scikit-Learn, DecisionTreeClassifier(), to predict final grades.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_
test_split
from sklearn.preprocessing import LabelEncoder
# split data into training and test sets
trainingData, testData = train_test_
split(gradeData, test_size=0.2)
encoder = LabelEncoder()
X_train = trainingData.loc[:,'Homework
1':'Homework 10']
y_train = encoder.fit_
transform(trainingData['Final Grade'])
X_test = testData.loc[:,'Homework 1':'Home-
work 10']
y_test = testData['Final Grade']
dec_tree = DecisionTreeClassifier(max_depth=2)
w = dec_tree.fit(X_train, y_train) # build de-
cision tree
gradePredictions = dec_tree.predict(X_test) #
test decision tree on test data
```

The predictions and actual grades for 10 random students in the test set are shown in Table 2. For this particular example, the performance of the decision tree is close but not quite as good as logistic regression, but for many other applications, decision trees can be as good or better for supervised learning. The decision tree performance here could also be improved by allowing more than two layers in the decision tree; a limitation set by the max_depth=2 parameter in the Python code, but increasing the number of layers risks over fitting the data in the training set, which would lead to worse predictions in general.

One of the advantages to building decision trees is that the resulting model can help in identifying both critical features and potentially important values associated with those features.

The decision tree shown in Figure 1 is based on predicting final letter grades using only the first six homework assignments from the GradesA.xlsx data set. The decision tree starts with 121 individuals in the training set (recall that the other 30 students are in the test set), and the values shown for each box in the decision tree show the number of students in that group that ultimately receive an A, B, C, D, or F, respectively. The first branch in the decision tree is based on the score for homework 3. No student that ended with an A in the class received less than a 7.25 on this homework assignment. Further, all the students receiving D or F at the end of the course received less than a 7.25 on this assignment. This could be helpful information to the students in the course that might see this as a relatively unimportant homework assignment. The next layer on the decision tree looks at homework 5 for students that did well (>7.25 points) on homework 3 and at homework 6 for students that did not score above 7.25 on homework 3. Recall that the weights determined using linear regression also suggested that homework 6 was very important in predicting final grades and homework 4 was not important. Homework 4 does not appear in the decision tree and homework 6 is present. Finally, Figure 1 also shows the gini score for each branch point (or node) and this is a measure of the impurity of a node where gini=0 is a "pure" node.[7] The gini score is a measure of how often a randomly chosen data point from the data set would be incorrectly labeled with a grade if it was assigned a random grade using the distribution of grades in the data set.

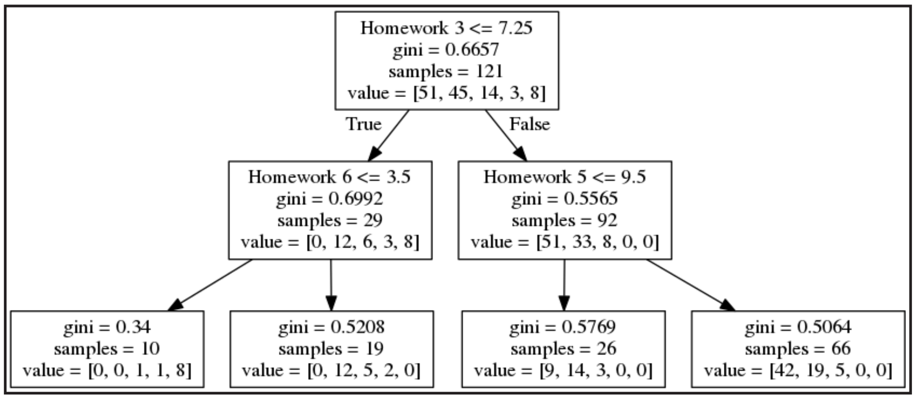

Figure 1. *Decision Tree for predicting final grades using only the first six homework assignments in the GradesA.xlsx data set.*

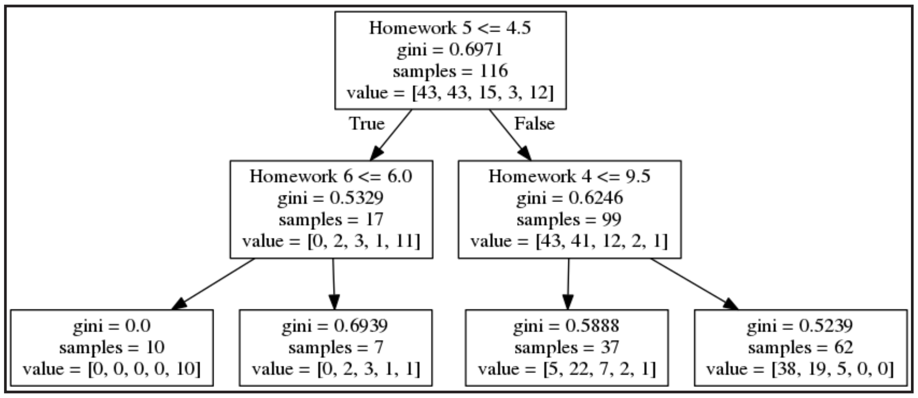

Figure 2. *Decision Tree for predicting final grades using only the first six homework assignments in the GradesB.xlsx data set.*

Constructing a decision tree on the second data set from GradesB.xlsx gives a similar result as shown in Figure 2. It is important to note that the year the course was taught corresponding to the year-B data set had one fewer homework assignment on the early course material so homework 4, which was less important in the year-A data set, would be most similar to homework 3 in this data set. Once again, the most important assignments for predicting final grades are from the middle of the semester. Students scoring a 5/10 or better on homework 5 and a 10/10 on homework 4 usually received an A and always passed with a C or better. Students receiving a 4/10 or less on homework 5 and less than a 6/10 on homework 6 always received an F for this data set.

MATERIAL AND ENERGY BALANCES COURSE

The two data sets used previously to predict final grades based on homework (or midterm) grades during a semester were from the same course on engineering computations over two different years. A third data set of grades from a material and energy balances course was examined using the same tools from ML. The goal was to identify similarities and differences in the ML models that are developed for predicting final grades. The final grades in this course were based on the completion of 10 homework assignments (10% of final grade), nine in-class quizzes (10%), two midterm exams (25% each), and a final exam (30%). The homework assignments are graded based only on the fraction of problems that were attempted. There were 132 students that received a final grade in the course.

Using multivariable linear regression on the homework scores to predict final grades resulted in a low performance model. The weights associated with various homework assignments were highly variable (including some negative weights!), and the root mean square error on the test set was

12–15%, which means that the model predicts the correct letter grade less than half the time. This result is probably expected as the homework assignments were not graded for accuracy or correct answers, and, instead, points were given for attempting the problems. The nine quiz scores, on the other hand, gave a more accurate model even though the scores combined to represent only 10% of the final course grade. The root mean square error was typically 7–10%, which means the final grade was correctly predicted about half the time. Using the two midterm exam scores to predict the final grade resulted in a root mean square error of less than 5% typically, but this is not surprising since 50% of the final grade was from the midterm exams.

A decision tree was built using just the nine quiz scores and it was similar to those built for homework scores in the engineering computations course. A quiz grade from the middle of the semester, quiz #4, was at the top of the decision tree and the branch separated the "likely to pass" group from the "unlikely to pass" group. Interestingly, both branches in the second layer of the decision tree used quiz #8 from near the end of the course (this was the first quiz to contain an energy balance question).

Finally, a few remarks on using ML with more complex data sets. First, depending on the ML tool that is used, data scaling—which was not used here because the numerical values used were always between 0 and 100—may be critical for both good computational performance and accurate predictions. Second, these few examples were not intended to demonstrate the full potential of ML. These were relatively small data sets and only three different algorithms were tested. There are many other ML algorithms that could potentially be used to analyze course grades and anyone wanting to explore ML tools further for grade prediction or other applications is encourage to consider References 7–9.

CONCLUSIONS

The primary goal here was to illustrate how ML can be used to identify critical elements in a course that strongly impact overall performance. ML libraries are becoming easier to use and more flexible, allowing a number of different algorithms to be tested quickly using data from a standard data source like an Excel spreadsheet file. We close by asking if the type of analysis that results from grade prediction might be useful to students. Would students benefit from learning that certain assignments or quizzes are more important for predicting their final grade? Would students benefit from knowing that if they score below a certain percentage on a particular homework assignment, they are statistically unlikely to get an A in the course?

Hopefully, this discussion has provided a useful brief overview of terminology and types of tools used in ML applications. Using ML algorithms, instructors can take a few minutes to see if they can identify the most important and least important items used for student assessment in their courses and use the information if it is helpful. Finally, departments and programs are encouraged to "keep an eye" on the field of ML, as it is having a growing role in chemical engineering.

The primary goal here was to illustrate how ML can be used to identify critical elements in a course that strongly impact overall performance. ML libraries are becoming easier to use and more flexible, allowing a number of different algorithms to be tested quickly using data from a standard data source like an Excel spreadsheet file.

REFERENCES

1. Georgiopoulos M, et al. (2009) A sustainable model for integrating current topics in machine learning research into the undergraduate curriculum. *IEEE Transactions on Education* 52(4):503.
2. Mellody M (2014) *Training Students to Extract Value from Big Data: Summary of a Workshop* (National Academies Press, Washington, DC p. 66).
3. Qin SJ (2014) Process data analytics in the era of big data. *AIChE Journal* 60(9):3092.
4. Beck DAC, Carothers JM, Subramanian VR, & Pfaendtner J (2016) Data science: accelerating innovation and discovery in chemical engineering. *AIChE Journal* 62(5):1402.
5. Witten IH, Frank E, Hall MA & Pal CJ (2017) *Data Mining : Practical Machine Learning Tools and Techniques*, Fourth Edition (Elsevier, Amsterdam).
6. Alpaydin E (2014) *Introduction to Machine Learning*, Third edition, Adaptive computation and machine learning (The MIT Press, Cambridge, Mass.).
7. Géron A. (2017) *Hands-On Machine Learning With Scikit-Learn and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems* (O'Reilly, Sebastopol, CA).
8. Grus J (2015) *Data Science From Scratch : First Principles with Python* (O'Reilly, Sebastopol, CA).
9. Raschka S (2015) *Python Machine Learning: Unlock Deeper Insights Into Machine Learning With This Vital Guide To Cutting-Edge Predictive Analytics*. Community Experience Distilled (Packt Publishing, Birmingham, UK).
10. Due to updates at press time, see Reference 7.
11. Verma A (2016) Most Popular Programming Languages For Machine Learning And Data Science. Accessed June 9, 2017; Available from

<https://fossbytes.com/popular-top-programming-languages-machine-learning-data-science/>.

12. Collobert R, Farabet C, & Kavukcuoğlu K (2008) Torch: scientific computing for luajit, in *NIPS Workshop on Machine Learning Open Source Software*.
13. Jia Y, Shelhamer E, Donahue J, Karayev S, Long J, Girshick R, Guadarrama S, & Darrell T (2014) Caffe: Convolutional architecture for fast feature embedding, in *Proceedings of the 22nd ACM International Conference on Multimedia*.
14. Chollet F (2018) *Deep Learning with Python* (Manning Publications, Shelter Island, NY).
15. Al-Rfou R et al. (2016) Theano: A Python framework for fast computation of mathematical expressions. doi: arXiv:1605.02688. 472.
16. Yadav SK, & Pal S (2012) Data mining: a prediction for performance improvement of engineering students using classification. *World of Computer Science and Information Technology Journal* 2(2):51.
17. Huang S, & Fang N (2013) Predicting student academic performance in an engineering dynamics course: A comparison of four types of predictive mathematical models. *Computers & Education*. 61:133.
18. Wood DA, & Langevin MJ (1972) Moderating prediction of grades in freshman engineering. *Journal of Educational Measurement* 9(4):311.
19. Marbouti F, Diefes-Dux HA, & Madhavan K (2016) Models for early prediction of at-risk students in a course using standards-based grading. *Computers & Education*. 103:1.
20. Tekin A (2014) Early prediction of students' grade point averages at graduation: a data mining approach. *Eurasian Journal of Educational Research*. 54:207.
21. Lykourentzou I, Giannoukos I, Nikolopoulos V, Mpardis G, & Loumos V (2009) Dropout prediction in e-learning courses through the combination of machine learning techniques. *Computers & Education*. 53(3):950.
22. Dekker G, Pechenizkiy M, & Vleeshouwers J (2009) Predicting students drop out: A case study. in *Educational Data Mining 2009*.
23. Kotsiantis S, Pierrakeas C, & Pintelas P (2004) Predicting students' performance in distance learning using machine learning techniques. *Applied Artificial Intelligence* 18(5):411.
24. Kotsiantis S, Patriarcheas K, & Xenos M (2010) A combinational incremental ensemble of classifiers as a technique for predicting students' performance in distance education. *Knowledge-Based Systems*. 23(6):529.
25. Heys JJ (2017) *Chemical and Biomedical Engineering Calculations Using Python* (John Wiley & Sons, Hoboken, NJ). ❐