AutoML Feature Engineering for Student Modeling yields High Accuracy, but Limited Interpretability

Nigel Bosch
University of Illinois Urbana-Champaign
pnb@illinois.edu

Automatic machine learning (AutoML) methods automate the time-consuming, feature-engineering process so that researchers produce accurate student models more quickly and easily. In this paper, we compare two AutoML feature engineering methods in the context of the National Assessment of Educational Progress (NAEP) data mining competition. The methods we compare, Featuretools and TSFRESH (Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests), have rarely been applied in the context of student interaction log data. Thus, we address research questions regarding the accuracy of models built with AutoML features, how AutoML feature types compare to each other and to expert-engineered features, and how interpretable the features are. Additionally, we developed a novel feature selection method that addresses problems applying AutoML feature engineering in this context, where there were many heterogeneous features (over 4,000) and relatively few students. Our entry to the NAEP competition placed 3rd overall on the final held-out dataset and 1st on the public leaderboard, with a final Cohen's kappa = .212 and area under the receiver operating characteristic curve (AUC) = .665 when predicting whether students would manage their time effectively on a math assessment. We found that TSFRESH features were significantly more effective than either Featuretools features or expert-engineered features in this context; however, they were also among the most difficult features to interpret based on a survey of six experts' judgments. Finally, we discuss the tradeoffs between effort and interpretability that arise in AutoML-based student modeling.

Keywords: AutoML, Feature engineering, Feature selection, Student modeling

1. INTRODUCTION

Educational data mining is time-consuming and expensive (Hollands & Bakir, 2015). In student modeling, experts develop automatic predictors of students' outcomes, knowledge, behavior, or emotions, all of which are particularly costly. In fact, Hollands & Bakir (2015) estimated that costs approached $75,000 for the development of student models in one particularly expensive case. Although some of the expense is due to the inherent cost of data collection, much of it is due to the time and expertise needed for machine learning. This machine learning work consists of brainstorming and implementing features (i.e., feature engineering) that represent a student and thus largely determine the success of the student model and how that model makes its decisions. The time, expertise, and monetary costs of feature engineering reduce the potential for applying student modeling approaches broadly, and thus prevent students from realizing the full potential benefits of automatic adaptations and other improvements to educational software driven by student models (Dang & Koedinger, 2020). Automating parts of the machine-learning process may ameliorate this problem. In general, methods for automating machine-learning model-development processes are referred to as *AutoML* (Hutter et al., 2019). In this paper, we focus specifically on the problem of feature engineering, which is one of the most time-consuming and costly steps of developing student models (Hollands & Bakir, 2015). We explore AutoML feature engineering in the context of the National Assessment of Educational Progress (NAEP) data mining competition,[1] which took place during the last six months of 2019.

Building accurate student models typically consists of data collection, data preprocessing and feature engineering, and developing a model via machine learning or knowledge engineering (Fischer et al., 2020). In some cases, models are also integrated into educational software to provide enhanced functionality such as automatic adaptations, which requires additional steps (Pardos et al., 2019; Sen et al., 2018; Standen et al., 2020). Unfortunately, the expertise needed for such student modeling makes it inaccessible to many (Simard et al., 2017). Fortunately, recent methodological advances have made the machine learning and implementation steps cheaper and more accessible via user-friendly machine-learning software packages such as *TensorFlow*, *scikit-learn*, *mlr3*, and *caret* (Abadi et al., 2016; Kuhn, 2008; Lang et al., 2019; Pedregosa et al., 2011). Such packages are often used in educational data mining research (F. Chen & Cui, 2020; Hur et al., 2020; Xiong et al., 2016; Zehner et al., 2020). The feature-engineering step of modeling, however, remains difficult. Feature engineering consists of brainstorming numerical representations of students' activities (in this study, from records stored in log files), then extracting those features from the data either manually via data management software (e.g., SQL, spreadsheets) or programmatically. The brainstorming aspect of feature engineering can be a particular barrier to success because it may require both extensive knowledge of how students interact with the software in question and theoretical knowledge of constructs (e.g., self-regulated learning, emotion) to inspire features (Paquette et al., 2014; Segedy et al., 2015). Although theoretical inspiration for features benefits models by providing semantics and interpretability to the features, it does come at the cost of human labor. Explorations of AutoML feature engineering, like those in this paper, are relevant to understanding the spectrum of feature-engineering approaches and to informing future work that helps to combine the benefits of expert and AutoML approaches.

[1] https://sites.google.com/view/dataminingcompetition2019/home

We focus on two AutoML approaches with little prior use for feature engineering on student interaction log data. The first is TSFRESH (Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests), a Python package specifically for extracting features from time series data (Christ et al., 2018). The second is Featuretools, which extracts features based on relational and hierarchical data. TSFRESH features are largely inspired by digital signal processing (e.g., the amplitude of the first frequency in the discrete Fourier transform of the time between student actions), whereas Featuretools extracts features primarily by aggregating values across tables and hierarchical levels (e.g., how many times a student did action X while completing item Y). We compare these two methods along with expert feature engineering in the context of the NAEP data mining competition. NAEP data consist of interaction logs from students completing a timed online assessment in two parts; in the competition, we predict whether students will finish the entire second part without rushing through it (described more in the Method section). NAEP data offer an opportunity to compare AutoML feature engineering approaches for a common type of student-modeling task (a binary performance outcome) in a tightly controlled competition environment. Our contribution in this paper consists of answering three research questions using the NAEP data, supplemented with a survey of experts' perceptions of feature interpretability. Additionally, we describe a novel feature selection procedure that addresses issues applying AutoML feature engineering in this context. Our research questions (RQs) are:

RQ1: Are student models with AutoML features highly accurate (specifically, are they competitive in the NAEP data mining competition)?

RQ2: How do TSFRESH and Featuretools compare to each other and to expert-engineered features in terms of model accuracy?

RQ3: How interpretable are the most important AutoML features in this use case?

We hypothesized that AutoML features would be effective for prediction (RQ1), and compare favorably to expert-engineered features in terms of predictive accuracy (RQ2), but that it may be difficult to glean insights about specific educational processes from models with AutoML features given their general-purpose, problem-agnostic nature (RQ3). We selected TSFRESH — which extracts time series features — in part because we also expected that time-related features would be the most important from among many different types of features, given that NAEP assessment is a timed activity and timing is part of the definition of the outcome to be predicted.

The research questions in this paper focus specifically on AutoML for feature engineering, though that is only one aspect of AutoML research. We discuss AutoML more broadly next, as well as methods specifically for feature extraction.

2. RELATED WORK

AutoML methods vary widely based on the intended application domain. For example, in perceptual tasks such as computer vision, deep neural networks are especially popular. Consequently, AutoML methods for perceptual tasks have focused on automating the difficult parts of deep learning — especially designing effective neural network structures (Baker et al., 2017; Zoph & Le, 2017). Conversely, tasks with structured data, as in many student modeling tasks, are much more likely to make use of classical machine learning algorithms, which have different problems to solve.

2.1. AutoML for Model Selection

One of the best-studied areas in AutoML research is the CASH (Combined Algorithm Selection and Hyperparameter optimization) problem (Thornton et al., 2013). The goal of CASH is to produce a set of accurate predictions given a dataset consisting of outcome labels and features already extracted. Addressing the CASH problem thus consists of selecting or transforming features, choosing a classification algorithm, tuning its hyperparameters, and creating an ensemble of successful models. Methods that address CASH, or closely-related problems, include *auto-sklearn*, *TPOT* (Tree-based Pipeline Optimization Tool), and others (Feurer et al., 2020; Hutter et al., 2019; Le et al., 2020; Olson et al., 2016). CASH-related methods are quite recent, but not unheard of in student modeling research (Tsiakmaki et al., 2020). These methods include basic feature transformation methods, such as one-hot encoding and principal components analysis, but engineer only those new features that incorporate information already present in the instance-level dataset.

2.2. AutoML Feature Engineering

Deep learning methods offer an alternative means for automating instance-level feature extraction from lower-level data. For example, a recurrent neural network can learn patterns of sequential values that lead up to and predict an important outcome, such as whether a student will get a particular problem correct or even drop out of a course (Fei & Yeung, 2015; Gervet et al., 2020; Piech et al., 2015). In fact, the primary distinguishing characteristic of deep learning methods is this capability to learn high-level features from low-level data (LeCun et al., 2015). Deep learning may thus reduce the amount of expert knowledge and labor needed to develop a model, and can result in comparable prediction accuracy versus models developed with expert feature engineering (Jiang et al., 2018; Piech et al., 2015; Xiong et al., 2016). Moreover, deep learning models have proven practical in real educational applications (Pardos et al., 2017). However, as Khajah et al. (2016) noted, deep learning student models have "tens of thousands of parameters which are near-impossible to interpret" (p. 100), a problem which may itself require a substantial amount of effort to resolve. Moreover, these methods work best in cases where data are abundant (Gervet et al., 2020; Piech et al., 2015). This is not the case in the NAEP data mining competition dataset, where there are many low-level data points (individual actions) but only 1,232 labels. Hence, other approaches to automating feature engineering may be more appropriate. We explored methods that automate some of the most common types of expert feature engineering, such as applying statistical functions to summarize a vector in a single feature, all without deep learning or the accompanying need for large datasets.

TSFRESH and Featuretools are two recent methods that may serve to automate feature extraction even with relatively little data. Both are implemented in Python, and integrate easily with *scikit-learn*. TSFRESH extracts features from a sequence of numeric values (one set of features per independent sequence) leading up to a label (Christ et al., 2018). Natural applications of TSFRESH include time series signals such as audio, video, and other data sources that are relatively common in educational research contexts. For instance, Viswanathan & VanLehn (2019) applied TSFRESH to a series of voice/no-voice binary values generated by a voice activity detector applied to audio recorded in a collaborative learning environment. Similarly, Shahrokhian Ghahfarokhi et al. (2020) applied TSFRESH to extract features from the output of openSMILE, an audio feature extraction program that yields time series features (Eyben et al., 2010). In each of these cases, TSFRESH aggregated lower-level audio features to the appropriate level of the label, such as the student level, which were then fed into machine learning models. TSFRESH has also been applied to mouse movement data (Alyuz et al., 2017), facial muscle movements (Goswami et al., 2020), and time series records of the number of correct problems solved in an intelligent tutoring system (Karumbaiah et al., 2019). Although Gervet et al. (2020) noted that deep learning is likely to work well in situations where temporal patterns are important and data are plentiful, TSFRESH offers a method to automate the feature engineering process for temporal data even with only approximately 40 students (Shahrokhian Ghahfarokhi et al., 2020).

Highly comparative time series analysis (*hctsa*; Fulcher & Jones, 2017) is an approach that is very similar to TSFRESH, and provides AutoML time series feature extraction for MATLAB. *hctsa* extracts over 7,700 time series features intended for analyses of phenotypes in biological research, including many of the same types of features as TSFRESH. Comparisons of *hctsa* and TSFRESH on educational data would be valuable future work, though we focused on Python-based methods in this study. Instead, we compared TSFRESH to a much different type of method (i.e., Featuretools).

Featuretools offers a contrasting approach to automatic feature engineering that focuses on hierarchical, relational data rather than the exclusively numeric sequence data used in TSFRESH (Kanter & Veeramachaneni, 2015). Featuretools creates features from both numeric and nominal data using a method called *deep feature synthesis*, in which features are extracted by applying conditional statements and aggregation functions across many hierarchical levels (hence the name deep, not to be confused with deep neural networks). Featuretools has not yet been explored for student modeling purposes, with one exception; Mohamad et al. (2020) applied Featuretools to predict whether students would receive a course completion certificate in massive open online courses. In their study, Featuretools extracted features capturing discrete events, such as video watching or forum usage, as well as timing features related to these events.

In another approach, researchers have also explored AutoML methods that combine existing features to generate new features. *autofeat* is one such method implemented in Python (Horn et al., 2020). *autofeat* transforms and combines input features by, for example, multiplying two features, or squaring one feature and dividing it by the log of another. The transformed features are then fed into a linear classifier, under the assumption that nonlinear relationships have already been accounted for in the feature generation process. ExploreKit is a similar, Java-based approach in which features are transformed and combined to produce new features that may capture nonlinear patterns or interactions between features. However, these methods assume features have already been extracted by some previous method (e.g., expert feature engineering). Like *auto-sklearn* and TPOT, they do not address the problem of engineering features from sequential or relational data in students' log files, which is the focus of this paper.

2.3. Related Work vs. Current Study

As described above, TSFRESH and Featuretools have both been explored to some extent for student modeling purposes. However, in this paper, we explore novel uses of both methods. For the first time,[2] we apply TSFRESH to sequences of students' time taken per action, and we apply Featuretools to multiple types of categorical values such as events and problem IDs. Moreover, we incorporate these features with expert-engineered features and briefly explore their interpretability.

[2] Based on a review of all results for a Google Scholar search for "students" within articles citing TSFRESH (Christ et al., 2018) and Featuretools (Kanter & Veeramachaneni, 2015) papers.

3. Method

Our method was intended to extract features that would produce competitive results in the NAEP data mining competition with straightforward machine learning model training procedures. Hence, we focused on feature engineering and feature selection, rather than exploring complex deep neural network approaches. The full source code and instructions for applying our methods are available online at https://github.com/pnb/naep, including exhaustive lists of features extracted.

3.1. NAEP Data Preparation

The NAEP data mining competition dataset was collected from 2,464 8th grade students in the United States. NAEP includes assessments of several different topics, but this specific competition dataset consisted exclusively of data from the math assessment module. As noted earlier, the NAEP assessment includes two parts (blocks A and B); data were provided only from block A, whereas ground truth (the outcome label) was defined in terms of students' behaviors in block B. Specifically, each student was labeled as "efficient", the positive class, if they completed every problem in block B and if they were in the 5th or higher percentile for time spent on every problem. The base rate of the positive class defined this way was .604. Otherwise, students were labeled as the negative class — i.e., they either did not manage to finish the whole assessment or they rushed through at least one problem without spending much time on it. This ground truth definition is a measure of students' time management skills, rather than an academic outcome such as their score on the assessment. Presumably, students' time management skills in block A are related to those same skills in block B. Hence, block A was provided to competitors as training data to predict the time management label in block B.

Competition organizers split the data into equal-sized training and holdout sets of 1,232 students. Each subset consisted of logs of students' NAEP software interactions from block A and labels derived from block B. We had access to labels for the training data but not the holdout data; the goal of the competition was to make predictions on the holdout data, which the competition organizers evaluated. Half of the holdout data were evaluated for a public leaderboard where everyone's ranking was visible and half were evaluated for the final ranking to determine competition winners. The public leaderboard was automatically updated once daily, so it was possible to overfit a solution to the public leaderboard by submitting many variations of methods and capitalizing on random chance. The final ranking, which was evaluated only once, is thus a better measure of prediction accuracy.

Competition organizers also divided the holdout data into thirds (411, 411, and 410 students) and truncated two of those thirds: the first to only the initial 10 minutes of students' interactions and second to the initial 20 minutes. The last third had all 30 minutes of block A interactions. We preprocessed the training data to match the holdout data by creating one version with the first 10 minutes of interactions, one version with 20 minutes, and keeping one original copy with all 30 minutes. We treated each time length individually, training separate 10, 20, and 30-minute models — each of which included all students from the training data. Except where noted, the feature extraction and model training processes detailed below were identical for each of the three different time length models.

3.2. AutoML Feature Engineering

As mentioned previously, we extracted features via two AutoML feature engineering methods: TSFRESH and Featuretools.

3.2.1. Time Series Features (TSFRESH)

TSFRESH extracts many features from numerical sequence data. The feature extraction methods are configurable and may be set to extract an arbitrarily large number of features. We applied its default configuration, however. On our data, the default configuration extracted over 100 valid (i.e., non-zero variance) features per sequence for several different sequences. Specifically, we made sequences of the number of seconds spent per action, seconds per problem, seconds per "item" (which includes both problems and additional minor activities like introductory instructions), and seconds per action in the last 5 minutes (whether that was the last 5 minutes of the first 10 minutes or the last 5 minutes of the whole 30-minute session). TSFRESH takes these sequences and generates features such as mean, standard deviation, linear slopes, coefficients of polynomial functions, frequencies (from a discrete Fourier transform), and many others.

In total, TSFRESH extracted 467 features for 10-minute data, 516 for 20-minute data, and 460 for 30-minute data. The number of features varied slightly per time length because TSFRESH discards features that encounter errors (e.g., division by zero). Although not all TSFRESH features are inscrutable, many are. Features include, for example:

- *per_problem_sec__cwt_coefficients__widths_(2, 5, 10, 20)__coeff_2__w_10*
    - The second coefficient from a continuous wavelet transform with width 10 applied to the sequence of seconds spent per problem
- *delta_sec_last5__large_standard_deviation__r_0.2*
    - Whether the standard deviation of the time spent per action in the last five minutes is larger than 0.2 times the range of those same values

Such features may capture behaviors such as students speeding up or slowing down as they go through problems, rushing through the end of the problems, or budgeting time per problem like their peers. However, because many extracted features are likely irrelevant to the label in any particular modeling task, TSFRESH also includes feature selection functionality. It measures the Pearson correlation between every feature and the outcome label, then selects those that are significantly correlated (p < .05) with the outcome after a post-hoc correction for multiple tests. However, we found this method to be overly conservative in initial tests: it would often select 0 features. This is expected for large numbers of features where even the best features are only moderately correlated with the outcome because the post-hoc correction in TSFRESH (i.e., Benjamini & Hochberg, 1995) controls the false discovery rate at a fixed level (5%). When correlations are all modest, it may be impossible to select any features without expecting to exceed the false discovery rate. TSFRESH allows the false discovery rate to be adjusted, but we instead developed a unified feature selection framework that could easily be applied to all types of features, including TSFRESH, Featuretools, and expert-engineered features (see Section 3.3). Our feature-selection framework minimizes redundancy by removing highly correlated features, but never selects fewer features when additional irrelevant features are added, unlike false discovery rate methods.

3.2.2. Hierarchical and Relational Features (Featuretools)

Featuretools automatically engineers and extracts features from tabular data arranged in a format similar to a relational database. For example, given a specific set of relationships such as *variable A in table 1 corresponds to variable A in table 2*, Featuretools will extract features like *how often value X occurs in variable B of table 2 given that value Y occurred in variable C of table 1*. It also extracts simple per-table aggregate features, such as means, modes (for categorical variables), and others. Featuretools only requires specifying variable types (e.g., which variables are dates versus which are numeric) and relationships between tables.

We specified three levels of hierarchy for the NAEP dataset: the row-level (lowest level) nested within the item/problem level, nested within the student level (Figure 1). Featuretools then computes features at each level and aggregates them at whichever level is specified. In this case, we specified student-level features, resulting in a feature set that included, for example:

- *NUM_UNIQUE(rows.Observable WHERE AccessionNumber = VH098783)*
  - Count of how many different types of actions a student did during one specific item (problem ID VH098783)
- *AVG_TIME_BETWEEN(rows.EventTime WHERE AccessionNumber = VH098834)*
  - Average amount of time per action that a student spent on one specific item (problem ID VH098783)
- NUM_UNIQUE(rows.AccessionNumber)
  - Count of how many items a student viewed at least once

Given our specified columns and hierarchy, Featuretools extracted 2,980 features for each of the 10-, 20-, and 30-minute time lengths. We did not explore options to generate even more features (e.g., extracting features from only the last five minutes as we did with TSFRESH) because this was already a large number of features.

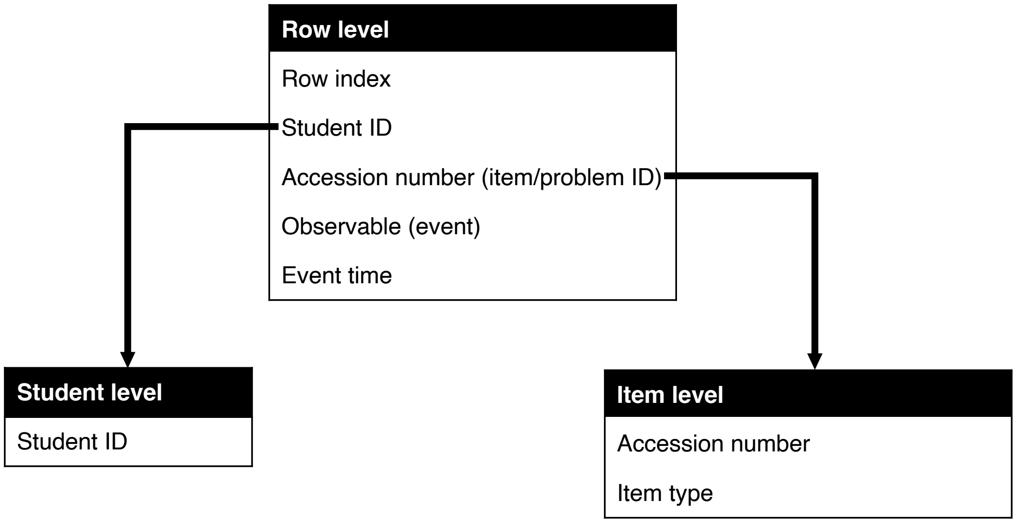

Figure 1: Illustration of the variables and relationships between variables that we specified for Featuretools. There is one table per hierarchical level, even when there is no meaningful data associated with a level (e.g., in the case of the student level) so that Featuretools automatically aggregates the other variables at that level.

3.2.3. Expert Feature Engineering

We engineered a variety of features via brainstorming and human effort, many of which we found were largely redundant with AutoML features. For example, we engineered features such as how long students spent on each item, the number of actions per item, the standard deviation of time spent per item, and other features that seemed likely to be related to the outcome label. Redundant features were removed during the feature selection process, as described in Section 3.3.2.

We also engineered answer popularity features, which were not redundant with AutoML features. The NAEP dataset does not contain information about whether a student's answer was correct, so we instead focused on how popular each answer was as a proxy for correctness and common misconceptions. Doing so was straightforward for multiple choice and similar question formats, but complicated for other questions where students could enter answers into a box with few restrictions. The NAEP system recorded such answers in LaTeX because students could enter fractions, math symbols, and special characters that are not easily represented without markup. We created a simple parser for LaTeX answers that standardized student responses by converting fractions to decimal numbers, standardizing leading and trailing digits, and deleting various other LaTeX commands. This method was imperfect, but served to convert most LaTeX answers to comparable decimal numbers.

After standardizing answers, we ranked them according to popularity and extracted several features consisting of the count of answers each student had in rank 1 (i.e., the most popular answer) through rank 5. We also calculated the standard deviation of their answer ranks, their answer rank for each item, and similar features intended to measure how often students were choosing popular (perhaps correct) versus less popular answers.

We also extracted features from the first 5 minutes of data separately. For the 30-minute time length data, we extracted features from the last 5 minutes of data as well. In total, we extracted 644 expert-engineered features from 10-minute data, 647 from 20-minute data, and 990 from 30-minute data. As we did for TSFRESH, we discarded features that resulted in calculation errors such as division by zero. Thus, there is variation in the number of features selected across time lengths.

We treated the expert-engineered features like the AutoML features during feature selection, as described next.

3.3. Feature Selection

Irrelevant features cause problems if they represent a large proportion of the data, even in decision-tree ensembles (Geurts et al., 2006). For example, if almost all features in a dataset are irrelevant, then a random forest type model will consist mostly of useless trees because each tree is built from a random subset of features (Breiman, 2001). Any trees that do happen to contain a relevant feature are also less likely to contain multiple features that may be needed to capture a meaningful interaction among features. Hence, some feature selection is necessary.

As mentioned above, TSFRESH's default feature selection was ineffective for these data. Additionally, we could not apply common wrapper forward feature selection methods because they are quite slow for large numbers of features (Sanyal et al., 2020). Forward selection and backward elimination wrapper methods have quadratic runtime complexity with respect to the number of features in a dataset; given the runtime for a single model in our experiments, we estimated forward feature selection would require approximately 30 years to complete even without optimizing hyperparameters described below (Section 3.4). We thus developed a feature selection method to deal with the large number of features in this study (over 4,000). The goal of this multi-phase method was to select predictive, unique features that generalized well to new data (i.e., to the competition holdout data).

3.3.1. Supervised Feature Selection

In the first phase of feature selection, we evaluated each feature individually by training one-feature models consisting of a single CART (Classification And Regression Trees; Breiman et al., 1984) decision tree. We set the minimum number of samples per tree leaf to 8 and left other hyperparameters at *scikit-learn* defaults. A single CART tree with one feature does not measure the value of features in interactions with other features, unlike wrapper feature selection methods that may better identify groups of features that work well together. However, a single tree is fast to train and does at least capture possible non-linear relationships between the feature and the outcome label. Moreover, the method we used to train our final models (see Section 3.4) is an ensemble of trees, so we expected that features which worked well in a single tree would likely also work well in an ensemble of trees.

We evaluated one-feature models via four-fold cross validation and the AUC (area under the receiver operating characteristic curve) metric. AUC makes up half of the NAEP competition evaluation metric; the other half is Cohen's kappa. We relied on AUC alone, however, because one-feature models were often quite weak predictors such that some of the predictions might be in the correct rank order (which AUC measures) but the predictions might all be above .5 probability or all below it, yielding a chance-level kappa. We wanted to include individual features even if they were only effective for high- or low-probability predictions, so measured only AUC during feature selection. We selected features where the minimum one-feature model AUC across all four cross-validation folds was above .5 (chance level). This approach was intended to favor selecting features that were consistently related to the outcome label across samples, even in relatively small samples. We expected that such features would be more likely to generalize with above-chance accuracy to the holdout competition dataset as well.

We applied further supervised selection to narrow down the list of selected features to those with consistent distributions between training and holdout datasets. To do so, we combined training (without labels) and holdout datasets, and added a new column denoting whether the instance was from the training or holdout dataset. We then applied the same one-feature model training process described above to discover if it was possible to predict whether an instance was in the training or holdout dataset based on each feature. We selected only features with a mean AUC < .55 across four cross-validation folds. This removed any features with notably different distributions of values in the holdout data, and which would thus likely not generalize to new data.

We also compared the Kolmogorov-Smirnov (K-S) distribution similarity statistic as an alternative to the AUC-based method, based on 30-minute data as an example. The K-S statistic tests for deviations from the null hypothesis that two variables come from the same distribution. A large value of the K-S statistic (and a small *p*-value) indicates that the two variables' distributions differ; thus, we expected K-S statistics to correlate with AUCs from our proposed method. Indeed, K-S statistics correlated $r = .437$ ($p < .001$) with AUCs across all features. Although this correlation was in the expected direction, there were clearly differences between the two methods. We also measured the correlation between K-S statistics and AUCs computed without cross-validation. These correlated more strongly ($r = .765$, $p < .001$), indicating that the K-S statistic produced results similar to an overfit AUC. We expected that the cross-validated AUC-based method would more likely identify generalizable distribution differences, and, specifically, distributions differences detectable by decision trees used in our final model. Thus, we proceeded with the AUC-based method rather than K-S tests.

In sum, our supervised feature selection phases yielded a set of features that were, in theory, related to the outcome label and generalizable to new data. Subsequent unsupervised feature selection dealt with redundancies between individual features and dependencies among features.

3.3.2. Unsupervised Feature Selection

We removed redundant features that were highly correlated with another feature (i.e., Spearman's *rho* > .9). We applied the correlated feature-removal process iteratively, starting with the highest-correlated features. If two features were perfectly correlated with each other, we removed one randomly. Otherwise, we chose which feature in a pair of highly-correlated feature to remove by measuring the mean correlation between each of the two features and all other remaining features in the dataset. We then removed whichever feature had the higher mean correlation, thus keeping the overall less-redundant feature.

For the sake of computational efficiency, we applied the unsupervised feature selection steps above to each set of features individually (TSFRESH, Featuretools, and expert-engineered features). We then performed one final round of removal of perfectly correlated (*rho* = 1) features from among the three feature sets combined.

At the end of the feature selection process, we were left with 413 features for 10-minute data, 517 for 20-minute data, and 530 for 30-minute data. Thus, the feature selection process reduced the feature space to approximately 10% of the original (over 4,000 combined) features. Future work is needed to determine whether other fast feature-selection methods such as RELIEF-F (Kononenko, 1994) might produce similar results when configured to select 10% of the original features. However, we focused primarily on feature extraction in this study, and thus applied only one feature selection method before model training.

3.4. Model Training

We trained Extra-Trees models, which are ensembles of trees similar to random forest (Geurts et al., 2006). Random forest creates heterogeneity among the individual tree models in the ensemble by randomly selecting features and instances to be included in the training data for each tree. Extra-Trees takes this idea one step further by also randomizing the tree training process, such that the split points for each decision in a tree are chosen randomly. Although each individual tree may not be as accurate because of this added randomness, the trees are much faster to train and thus the ensemble can be much larger (and, hopefully, more accurate) for the same computational cost. We also explored eXtreme Gradient Boosting (XGBoost; T. Chen & Guestrin, 2016) and random forest models, but without notable improvements in accuracy, and both were much slower to train than Extra-Trees.

The main goal of the NAEP competition was to produce predictions for the holdout dataset. Therefore, we trained one model for each time length using all training data, then made predictions on the holdout data. However, we also applied four-fold cross-validation on the training data only, to explore methodological choices like our feature selection approach. For each model, we selected hyperparameters with a Bayesian search process implemented in the *Scikit-Optimize* Python package (Head et al., 2018). This method randomly selected 20 initial points from the space of possible hyperparameter values we defined, then trained a model for each one. It then iteratively trained another 80 models with hyperparameters chosen via Bayes' theorem to optimize our accuracy metric (Section 3.5) given the evidence from all previous models. We defined the hyperparameter search space as consisting of three key hyperparameters:

1) Maximum proportion of features to consider when making a branch in a decision tree (smaller values introduce more randomness in each tree); search space: uniform distribution over the [.001, 1] range
2) Proportion of instances to randomly sample for building each tree (smaller samples introduce more randomness); search space: uniform distribution over [.001, .999]
3) Cost-complexity pruning alpha (higher values prune more branches from each tree to prevent over-fitting; Breiman et al., 1984); search space: uniform distribution over [0, .004]

The first two hyperparameters control the bias-variance tradeoff of the model by balancing the benefits of having highly accurate individual trees in the ensemble (but with little uniqueness among trees) versus having highly heterogeneous trees (but each of which may not be accurate by itself). We probed the entire range of possible values for these two hyperparameters because Bayesian hyperparameter search quickly narrows down the range for subsequent fine-tuning. Cost-complexity pruning, on the other hand, has a range with an upper bound that is entirely problem-specific — hence the seemingly arbitrary .004 bound. To determine a reasonable range of values for the pruning hyperparameter, we trained an Extra-Trees classifier with 500 trees and otherwise default hyperparameters, then found the minimum pruning alpha such that all 500 trees were pruned down to a single branch. We then set this value (.004, rounded up slightly to provide a margin for error) as the upper bound for the pruning hyperparameter.

Apart from the three hyperparameters we tuned, we set the number of trees to 500 and used bootstrap sampling of random instances for each tree (which is not done by default in Extra-Trees). We left all other hyperparameters at their *scikit-learn* default values.

3.5. MODEL SELECTION AND FINAL PREDICTIONS

Hyperparameter tuning and other methodological choices require an accuracy metric to determine which model is best. Initially, we evaluated models based on the competition's evaluation metric: the sum of Cohen's kappa and AUC, where AUC is linearly adjusted to put it on the same scale as kappa; i.e., [-1, 1] rather than [0, 1]. However, we noted that some model variations with similar cross-validated accuracies yielded dissimilar results on the public competition leaderboard. We compared models that generalized to the leaderboard well and those that did not, and observed that the better models appeared to be less sensitive to the decision threshold used (which was .5 in the NAEP competition). Thus, we switched to selecting models based on a custom metric: area under the kappa curve (AUK), restricted to an interval with .1 width centered on the decision threshold that yielded maximum kappa. Unlike a single threshold-optimized value of kappa, AUK favored models that had a high kappa regardless of small fluctuations in the probabilities that were predicted because it measures kappa across a range of thresholds around the ideal threshold. We relied on AUK for selecting the best hyperparameters during cross-validation, then retrained the best model on all training data to make final predictions for the competition holdout data.

Adjusting the decision threshold to the point that yielded maximum kappa required rescaling our final predictions because the NAEP competition used a fixed threshold of .5. We rescaled predictions for each time length individually because we trained separate models for each and the ideal thresholds could vary. Depending on the ideal threshold *t*, we linearly rescaled predictions either up or down so that the ideal threshold corresponded to .5 (Equation 1). The rescaling procedure did not affect AUC because the order of predictions was preserved.

$$\hat{y}_{adjusted} = \begin{cases} \frac{\hat{y}}{2t}, & t \geq .5 \\ 1 - \frac{1-\hat{y}}{2(1-t)}, & t < .5 \end{cases} \tag{1}$$

Finally, we concatenated predictions from different time lengths and saved them in the order required for the competition. We also saved lists of selected features and the accuracy of one-feature models determined during the feature selection process described previously, so that we could answer our research questions about the predictive value and interpretability of AutoML features.

3.6. Surveying Expert Opinions on Feature Interpretability

We measured the interpretability of features via a small-scale survey of six experts, approved by the University of Illinois' Institutional Review Board. We recruited individuals who had previously published papers in the *Journal of Educational Data Mining* or the *International Conference on Educational Data Mining*, and, specifically, who had experience extracting features from students' log files. The survey consisted of two parts. In the first part, we selected the top five most predictive features from each of the three feature types, according to one-feature AUC in 30-minute data. We provided a definition for each feature and asked two questions: *How much effort does this feature definition require to understand?* and *How much could you infer about a student's learning experience from knowing that they had a particular value of this feature? (e.g., a low value, a high value)*. Both questions were on a 1–5 response scale from "Very little" (1) to "A great deal" (5). The features were presented in a random order, and the names of the features were not given, so that respondents would be more likely to evaluate each definition fairly and individually. Respondents were told that the survey was related to feature interpretability, but were not told that there were three types of features or that any of the features were AutoML features.

In the second part of the survey, we presented the same features in the same order with the same definitions, but also provided the name of the feature. We then asked *How much of the feature definition could you have guessed based solely on its name?* and collected responses on a 1–5 scale corresponding to *None*, *A little bit*, *Approximately half*, *Most of it*, or *Basically all of it*.

The three questions were designed to address different aspects of what "interpretable" might mean. We expect that more interpretable features are easier to understand, provide more insight into learning, and are more straightforward to represent with a variable name. For the purposes of analyses in this paper, we focused on mean responses to each question for each feature type. However, the survey questions and unaggregated response data are available in the project repository to promote more extensive future work in this area.[3]

[3] https://github.com/pnb/naep/tree/master/interpretability_survey

4. RESULTS

We describe results in terms of the three RQs outline in Section 1. The first research question focuses on our NAEP competition result, whereas later RQs examine the interpretability and value of the two AutoML feature engineering processes we applied.

4.1. RQ1: MAIN ACCURACY RESULTS

In RQ1, we asked *are student models with AutoML features highly accurate (specifically, are they competitive in the NAEP data mining competition)?* Our approach, largely based on AutoML features, was indeed effective in the competition; our final predictions yielded a 3rd place finish on the final leaderboard, after finishing 1st place on the public leaderboard.

Results showed that models were well above chance level, but far from perfect. On the final leaderboard, our submission had kappa = .2116, AUC = .6653, and aggregate score (kappa + AUC scaled to the [-1, 1] range) = .5422. Note that the aggregate score actually ranges from 0 to 2, where chance level is 0, because all negative scores were adjusted to 0. Thus, a score of .5422 is slightly over a quarter of the way between random chance and perfect accuracy.

First and second place models on the final leaderboard had aggregate scores of .5657 and .5524, respectively, whereas fourth and fifth place aggregate scores were .5413 and .5097, respectively. The top four solutions were separated by just .0244, indicating quite similar accuracy given the [0, 2] range of the aggregate score metric. The competition website provides brief descriptions of the methods employed by the top five results, which (apart from our method) mention neither deep learning nor AutoML feature extraction.[4] Hence, it appears that expert-engineered features were a popular choice for this prediction task; the AutoML methods we explored were similarly effective.

On the public leaderboard, our submission had kappa = .2799, AUC = .6676, and aggregate score = .6351, which was notably higher than the final leaderboard (aggregate score = .6351 versus .5422, respectively). This result may indicate that our final model was over-fit to the public leaderboard, which is expected given that we made certain methodological choices based on leaderboard results, such as the choice of restricted-range AUK for model selection and the choice of Extra-Trees classification instead of random forest or XGBoost. In total, we made 59 submissions, many which were small tweaks to explore the factors that could lead to effective generalization from training to held-out data. A nearly complete record of the source code for each submission attempt is available in the git history of our code (see beginning of Section 3). Our submission also had cross-validated accuracy scores that were similar to, or even higher than, public leaderboard results (kappa = .2703, AUC = .6985, and aggregate score = .6672), despite evaluation via cross-validation and model selection performed with nested cross-validation. Overfitting during cross-validation is also not surprising, however, given that we often made decisions about what to submit to the public leaderboard based on the best of several different cross-validated experiments.

We explored the possible public leaderboard overfitting issue further by measuring the rank order correlation between submission order and accuracy metrics. Submission order correlated moderately (Spearman's *rho* = .555) with kappa, *rho* = .646 with AUC, and *rho* = .688 with

[4] https://sites.google.com/view/dataminingcompetition2019/winners; see also a brief description of sequence-related features extracted in the second-place method: https://medium.com/playpower-labs/lets-do-educational-process-mining-5dcfd1e606ba

aggregate score (all $p < .001$). Thus, our submissions certainly became more fit to the leaderboard over time, and may easily have become over-fit as well. However, because final leaderboard evaluates holdout data only once, we can be confident that the results were not over-fit to the final leaderboard.

4.2. RQ2: Accuracy of AutoML vs. Expert-engineered Features

RQ1 showed that our model, which used feature-level fusion including two kinds of AutoML features and expert-engineered features, was indeed competitively accurate. In RQ2, we explored which types of features contributed to that accuracy by asking *how do TSFRESH and Featuretools compare to each other and to expert-engineered features in terms of model accuracy?* We approached RQ2 by examining only the features from the 30-minute time length; this simplifies the analyses and focuses on the time length with the most features.

We examined the results of the one-feature models trained during feature selection (see Section 3.3.1) to assess the accuracy of every feature that we extracted. Specifically, we examined the mean AUC across the four cross-validation folds. We compared TSFRESH, Featuretools, and expert-engineered feature sets via three pairwise independent *t*-tests, which showed that TSFRESH features were most accurate (mean per-feature AUC = .530), followed by expert-engineered features (AUC = .512), followed by Featuretools (AUC = .502). In the pairwise comparisons, the TSFRESH vs. expert-engineered features effect size was $d = 0.649$ (a medium effect by some standards; Cohen, 1988); for TSFRESH versus Featuretools, $d = 1.522$ (a large effect); for expert-engineered versus Featuretools, $d = 0.587$ (a medium effect). All pairwise comparisons were significant ($p < .001$). Note, however, that many features were highly correlated with each other both within and between the three feature sets. Hence, statistical significance should be interpreted with caution.

We also performed comparisons between feature types with only the features that were selected, because many features were redundant or irrelevant (especially for Featuretools) and may have diminished the mean AUC. Unsurprisingly, mean AUC was higher for each feature type in this analysis, but the rank ordering of feature types did not change. TSFRESH mean AUC was .550, versus expert-engineered AUC = .538, versus Featuretools AUC = .529. In pairwise comparisons, TSFRESH versus expert-engineered $d = 0.497$ (small effect), TSFRESH versus Featuretools $d = 0.945$ (large effect), and expert-engineered versus Featuretools $d = 0.395$ (small effect). All pairwise comparisons were still significant ($p < .0001$), though features may still have been correlated highly per the feature selection method, so statistical significance should be interpreted cautiously.

Finally, we also examined feature importance in the cross-validated 30-minute model to verify that patterns in the one-feature AUCs held up in the full model with all features included. Results for the full model might differ, in theory, because some features could be more important when considered in interactions with other features; others might be redundant and thus have less importance. We measured feature importance via SHAP (SHapley Additive exPlanations), which provides the effect of each feature in the model on each prediction in the cross-validated test set (Lundberg & Lee, 2017). Specifically, we calculated the mean absolute SHAP value per feature as a measure of how important that feature was to the model's predictions. Overall, mean absolute SHAP correlated $r = .564$ ($p < .001$) with one-feature AUC, indicating that one-feature AUC provided a decent proxy for how important each feature would eventually be to the model. Results across feature types also matched patterns in the one-feature AUCs: TSFRESH features were best, followed by expert-engineered, followed by Featuretools.

4.3. RQ3: AutoML Feature Interpretability

Analyses for RQ2 established that AutoML features were useful for prediction in this context, especially the TSFRESH features. In RQ3, we asked *how interpretable are the most important AutoML features in this use case?* Interpretability is valuable and sometimes even essential in student modeling (Kay, 2000; Rosé et al., 2019), so it is critical to understand the extent to which AutoML features contribute or detract from interpretability. As with RQ2, we focus on the 30-minute time length as a representative example from among the three time lengths.

We first examined the best five features from each feature set to get a sense of what features were effective, and whether there were any exceptionally good features. Table 1 shows that the best two features overall were highly similar, expert-engineered features that were directly engineered from the definition of ground truth — i.e., whether students would spend sufficient time in every problem. After those two features, however, the next best three features were all TSFRESH features related to the amount of time spent per problem. In fact, all five of the most predictive TSFRESH features were extracted from the time series of time spent per problem, perhaps unsurprisingly because time per problem is an indirect proxy for the ground truth definition. Most of the best Featuretools features were also time related.

Table 1 provides human-readable descriptions of the meaning of each AutoML feature (e.g., "Per problem seconds, mean of all sets of 3 consecutive values multiplied together" rather than the internal TSFRESH name "per_problem_sec__c3__lag_1".) We created these descriptions based on TSFRESH and Featuretools documentation along with research literature that those documents cited. Such interpretation is a laborious process, and even with documentation several AutoML features were difficult to interpret for two reasons. First, it was not clear what some of the features meant without extensive domain knowledge. For example, TSFRESH's continuous wavelet transform features are common in seismic activity modeling (Ricker, 1953), but perhaps less well known in student modeling. Second, it was not clear what all features actually represented or why they might have been related to the outcome. For example, for most students, the number of events in problem ID VH098740 was a single number. Thus, the Featuretools feature capturing the skew of that distribution may seem irrelevant. The feature captured cases where a student visited that problem more than once (in which case they have a distribution of event counts), and, if so, whether they (a) skimmed through quickly at the beginning and revisited the problem longer later — a left skew — or (b) spent some time on the problem initially but may have reviewed it more briefly later.

Our expert survey results confirmed that even human-readable descriptions of AutoML features required additional effort to understand, and would yield less insight into learning experiences. Figure 2 illustrates these expert responses, showing that expert-engineered features required much less effort to understand and provided much more insight into learning. Experts were also least able to guess at feature definitions from the names for TSFRESH features, and most for expert-engineered features. However, Featuretools feature definitions were nearly as straightforward to guess from names as expert-engineered features. One survey respondent commented during a post-survey debrief that Featuretools feature names are verbose and thus offer much of the definition, but that the names themselves require substantial effort to parse.

Despite limited interpretability, AutoML features also lent some unexpected insights. We examined Featuretools features with above-chance AUC scores and found examples of features that were interpretable (with some effort) and which captured aspects of students' NAEP experiences that we had not thought of during expert feature engineering. For instance, two such features were:

- *MODE(rows.WEEKDAY(EventTime) WHERE items.ItemType = TimeLeftMessage) = 0.0*
    - Whether the assessment was on a Monday
- *MODE(rows.Observable WHERE AccessionNumber = VH098740) = Eliminate Choice*
    - Whether students used the process of elimination functionality in the user interface, for one particular multiple choice problem (ID VH098740), rather than simply selecting their final answer

Such features highlight that AutoML feature engineering methods can capture unexpected patterns that experts might not have considered. However, they also highlight the possibility that AutoML features might capture patterns that experts might rather *not* capture. For example, it is questionable whether a student's test-taking time management abilities should be predicted based on whether it is Monday. We discuss some of the implications of using these methods next.

Table 1: Five most predictive features from each feature set, based on one-feature model accuracy.

Feature	**AUC**
TSFRESH	
Per-problem seconds, absolute energy	.629
Per-problem seconds, mean of all sets of 3 consecutive values multiplied together	.628
Per-problem seconds, 30th percentile	.618
Per-problem seconds, 7th coefficient from a continuous wavelet transform with width 20	.614
Per-problem seconds, 20th percentile	.597
Featuretools	
Linear slope of time elapsed for each "Math Keypress" event (working on a fill-in-the-blank question)	.604
Linear slope of time elapsed over all events	.584
Standard deviation of timestamps for events in problem ID VH098812	.584
Linear slope of time elapsed for events in fill-in-the-blank questions	.576
Skew of the distribution of the number of events in problem ID VH098740	.575
Expert-engineered	
Count of items with >= 5th percentile time spent	.657
Count of problems (not including other items) with >= 5th percentile time spent	.654
Count of "Receive focus" events (starting a fill-in-the-blank question) in the last 5 minutes	.602
Total seconds spent in the NAEP software	.595
Count of less than 5th-most popular (i.e., rank > 5) answers in the last 5 minutes	.594

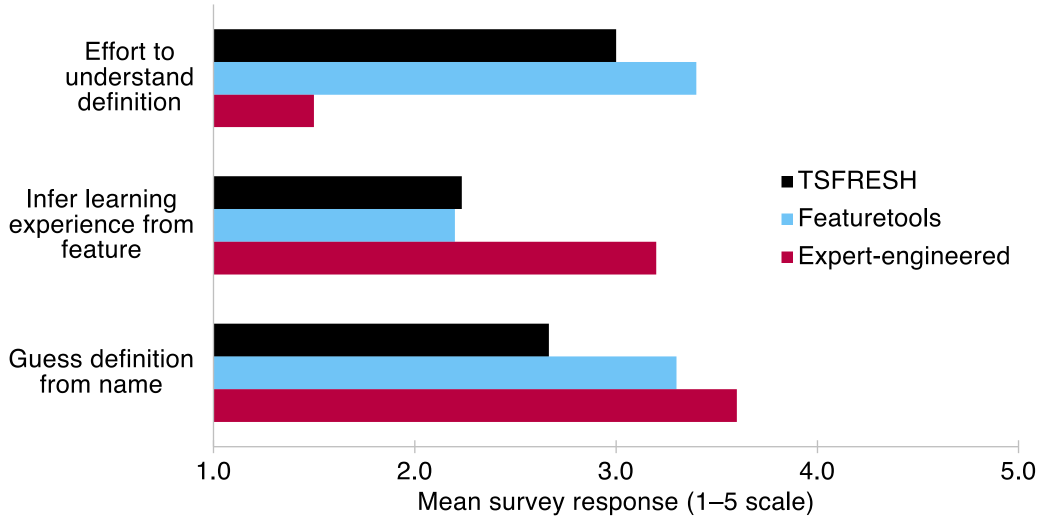

Figure 2: Aggregated responses from six experts for the three types of survey questions regarding feature interpretability. Higher values on the response scale indicate “more” (effort) or “better” (inference about learning, guess about the definition).

5. Discussion

Our results indicate that AutoML feature engineering approaches have promise for predicting educational outcomes from log files with little effort. However, our results also highlight issues with AutoML features. In this section, we discuss the implications of our results, highlight areas for future related work to address some of the limitations of this paper, and offer concluding remarks.

5.1. Implications for Accuracy, Effort, and Interpretability

AutoML features were effective with relatively little effort needed for brainstorming and extraction, though we did need to apply extensive feature selection to remove the many irrelevant features that were automatically extracted. In fact, expert-engineered features may have been almost entirely redundant in this use case, apart from answer ranking features (which were not possible to extract without expert knowledge) and features extracted specifically to match the ground truth definition. Two of the five most predictive expert-engineered features in Table 1 were also extracted by one or both AutoML methods, as were most of the other expert-engineered features. Hence, AutoML could perhaps relieve much of the feature engineering burden, at least for features that do not require domain knowledge to calculate.

We also noted differences in the two AutoML feature engineering methods used in this study. On average, TSFRESH features yielded superior accuracy compared to Featuretools features, and even compared to expert-engineered features. This finding is likely problem-specific, however: TSFRESH features are almost entirely time-based, which aligns well with the time management skills ground truth definition in the NAEP competition. Four of the five best Featuretools features were also time-related. Thus, in another student modeling task where time is less important, the rankings of feature types may have differed. This conclusion aligns with previous research noting that certain model types (e.g., deep learning versus Bayesian knowledge tracing) are more or less accurate depending on whether time is a crucial component of the data mining task (Gervet et al., 2020).

Regardless of accuracy, interpretability considerations may also guide selection of model and feature types (Kay, 2000; Rosé et al., 2019). The current study speaks to interpretability considerations across a variety of student modeling tasks (unlike accuracy results), given that the features extracted by AutoML were not specific to the task. Additionally, our survey questioned experts about learning experiences and student modeling in general, rather than about the specific prediction task in the NAEP competition. That AutoML features were difficult to interpret indicates that even with the simplest of models, such as a single small decision tree or a logistic regression, a model may be uninterpretable simply because of the features that it includes. Featuretools refers to its extraction method as "deep feature synthesis" (Kanter & Veeramachaneni, 2015), highlighting that the method considers highly complex interactions between variables at many hierarchical levels, much like the features implicitly extracted by a deep neural network. As shown in Section 4.3, such features are possible to interpret to a certain extent with sufficient effort, though this may simply be shifting researchers' effort from feature engineering to interpretation (Khajah et al., 2016). Future work is needed to explore methods for improving the interpretability of models built with AutoML features, as has been done with deep neural networks in education applications (Pardos et al., 2019).

5.2. Limitations and Future Work

There are a few opportunities for future work to address the limits of our study. One of the largest limitations of our approach was its focus on AutoML feature engineering specifically, when there are several other areas of the student modeling pipeline that AutoML approaches can address. Perhaps the most common of these is the model training step (the CASH problem discussed in Section 2.1). In future work, we plan to explore the relative benefits (mainly accuracy) for feature extraction versus CASH components of modeling as well as the costs (mainly interpretability). More empirical work is needed to compare such approaches to the often-high costs of expert involvement at every modeling step (Hollands & Bakir, 2015). Our focus on the feature engineering step also leaves future work to be done comparing fast feature selection methods beyond the method proposed in this paper (which could also be made more useful for future research by providing a generalized *scikit-learn* compatible interface).

It is also yet unknown whether the low-level features extracted by AutoML methods in this study will generalize broadly to the same extent as high-level, expert-engineered features. Paquette et al. (2015) found that student models trained on high-level features could generalize across entire educational software platforms to a certain extent, whereas the features automatically engineered by methods in our study might be quite specific to the NAEP competition dataset and software (e.g., features that included specific problem IDs). We did not explore this aspect of AutoML features in depth in our study, but the difference between public and final leaderboard results (RQ1) suggest a degree of overfitting and a lack of generalizability to new students. Whether this was due to the large, complex feature space inherent to AutoML features or because of the repeated submissions to the public leaderboard remains to be determined. At the very least, the results highlight the importance of evaluating data mining research on truly unseen data, where results may differ from the cross-validation results obtained during method development. Future work in this area is needed to characterize the generalizability of models trained on AutoML features across time, student populations, and even software platforms.

Finally, our survey of experts' perceptions of feature interpretability was exploratory in nature, with just six expert participants and thus limited ability to make statistically verified claims. In future work, we hope to extend this survey to include more experts, more types of AutoML features, expert-engineered features from more projects, and more aspects of interpretability.

5.3. CONCLUDING REMARKS

Manual feature engineering can be expensive and time-consuming (Hollands & Bakir, 2015). We explored whether out-of-the-box AutoML feature-engineering methods show promise for reducing the time and labor needed for feature engineering. We found that models based on AutoML features were indeed competitively accurate in the context of the NAEP data mining competition. However, there are serious concerns regarding model interpretability with these methods that need to be addressed in applications where it is important for researchers, teachers, or students (or all three) to be able to quickly understand why a model makes a particular decision. Thus, our results contribute to the understanding of methods for cost-effective educational data mining, but future work is needed to discover how well stakeholders are able to interpret the predictions made with these methods. Methodological work is also needed to improve the interpretability of existing AutoML features or to research AutoML feature engineering methods that incorporate interpretability constraints.

REFERENCES

ABADI, M., BARHAM, P., CHEN, J., CHEN, Z., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., IRVING, G., ISARD, M., KUDLUR, M., LEVENBERG, J., MONGA, R., MOORE, S., MURRAY, D. G., STEINER, B., TUCKER, P., VASUDEVAN, V., WARDEN, P., … ZHENG, X. (2016). TensorFlow: A system for large-scale machine learning. *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 265–283.

ALYUZ, N., OKUR, E., GENC, U., ASLAN, S., TANRIOVER, C., & ESME, A. A. (2017). An unobtrusive and multimodal approach for behavioral engagement detection of students. *Proceedings of the 1st ACM SIGCHI International Workshop on Multimodal Interaction for Education*, 26–32. https://doi.org/10.1145/3139513.3139521

BAKER, B., GUPTA, O., NAIK, N., & RASKAR, R. (2017). Designing neural network architectures using reinforcement learning. *ArXiv:1611.02167 [Cs]*. http://arxiv.org/abs/1611.02167

BENJAMINI, Y., & HOCHBERG, Y. (1995). Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B (Methodological)*, *57*(1), 289–300.

BREIMAN, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324

BREIMAN, L., FRIEDMAN, J., STONE, C. J., & OLSHEN, R. A. (1984). *Classification and regression trees*. CRC press.

CHEN, F., & CUI, Y. (2020). Utilizing student time series behaviour in learning management systems for early prediction of course performance. *Journal of Learning Analytics*, *7*(2), 1–17. https://doi.org/10.18608/jla.2020.72.1

CHEN, T., & GUESTRIN, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794. https://doi.org/10.1145/2939672.2939785

CHRIST, M., BRAUN, N., NEUFFER, J., & KEMPA-LIEHR, A. W. (2018). Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package). *Neurocomputing*, *307*, 72–77. https://doi.org/10.1016/j.neucom.2018.03.067

COHEN, J. (1988). *Statistical Power Analysis for the Behavioral Sciences* (2nd ed.). Lawrence Erlbaum.

DANG, S. C., & KOEDINGER, K. R. (2020). Opportunities for human-AI collaborative tools to advance development of motivation analytics. *Companion Proceedings of the 10thInternational Conference on Learning Analytics & Knowledge (LAK20)*, 322–329.

EYBEN, F., WÖLLMER, M., & SCHULLER, B. (2010). OpenSMILE: The Munich versatile and fast open-source audio feature extractor. *Proceedings of the 18th ACM International Conference on Multimedia*, 1459–1462. https://doi.org/10.1145/1873951.1874246

FEI, M., & YEUNG, D.-Y. (2015). Temporal models for predicting student dropout in massive open online courses. *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, 256–263. https://doi.org/10.1109/ICDMW.2015.174

FEURER, M., EGGENSPERGER, K., FALKNER, S., LINDAUER, M., & HUTTER, F. (2020). Auto-sklearn 2.0: The next generation. *ArXiv:2007.04074 [Cs, Stat]*. http://arxiv.org/abs/2007.04074

FISCHER, C., PARDOS, Z. A., BAKER, R. S., WILLIAMS, J. J., SMYTH, P., YU, R., SLATER, S., BAKER, R., & WARSCHAUER, M. (2020). Mining big data in education: Affordances and challenges. *Review of Research in Education*, *44*(1), 130–160. https://doi.org/10.3102/0091732X20903304

FULCHER, B. D., & JONES, N. S. (2017). hctsa: A computational framework for automated time-series phenotyping using massive feature extraction. *Cell Systems*, *5*(5), 527-531.e3. https://doi.org/10.1016/j.cels.2017.10.001

GERVET, T., KOEDINGER, K., SCHNEIDER, J., & MITCHELL, T. (2020). When is deep learning the best approach to knowledge tracing? *Journal of Educational Data Mining*, *12*(3), 31–54. https://doi.org/10.5281/zenodo.4143614

GEURTS, P., ERNST, D., & WEHENKEL, L. (2006). Extremely randomized trees. *Machine Learning*, *63*(1), 3–42. https://doi.org/10.1007/s10994-006-6226-1

GOSWAMI, M., MANUJA, M., & LEEKHA, M. (2020). Towards social & engaging peer learning: Predicting backchanneling and disengagement in children. *ArXiv:2007.11346 [Cs]*. http://arxiv.org/abs/2007.11346

HEAD, T., MECHCODER, LOUPPE, G., SHCHERBATYI, I., FCHARRAS, VINÍCIUS, Z., CMMALONE, SCHRÖDER, C., NEL215, CAMPOS, N., YOUNG, T., CEREDA, S., FAN, T., RENE-REX, SHI, K., SCHWABEDAL, J., CARLOSDANIELCSANTOS, HVASS-LABS, PAK, M., … FABISCH, A. (2018). *scikit-optimize/scikit-optimize: V0.5.2*. https://doi.org/10.5281/zenodo.1207017

HOLLANDS, F., & BAKIR, I. (2015). Efficiency of automated detectors of learner engagement and affect compared with traditional observation methods. *New York, NY: Center for Benefit-Cost Studies of Education, Teachers College, Columbia University*.

HORN, F., PACK, R., & RIEGER, M. (2020). The autofeat Python library for automated feature engineering and selection. In P. Cellier & K. Driessens (Eds.), *Machine Learning and*

Knowledge Discovery in Databases (pp. 111–120). Springer International Publishing. https://doi.org/10.1007/978-3-030-43823-4_10

Hur, P., Bosch, N., Paquette, L., & Mercier, E. (2020). Harbingers of collaboration? The role of early-class behaviors in predicting collaborative problem solving. *Proceedings of the 13th International Conference on Educational Data Mining (EDM 2020)*, 104–114.

Hutter, F., Kotthoff, L., & Vanschoren, J. (2019). *Automated Machine Learning: Methods, Systems, Challenges*. Springer Nature. https://library.oapen.org/handle/20.500.12657/23012

Jiang, Y., Bosch, N., Baker, R. S., Paquette, L., Ocumpaugh, J., Andres, J. Ma. A. L., Moore, A. L., & Biswas, G. (2018). Expert feature-engineering vs. Deep neural networks: Which is better for sensor-free affect detection? In C. P. Rosé, R. Martínez-Maldonado, H. U. Hoppe, R. Luckin, M. Mavrikis, K. Porayska-Pomsta, B. McLaren, & B. du Boulay (Eds.), *Proceedings of the 19th International Conference on Artificial Intelligence in Education (AIED 2018)* (pp. 198–211). Springer. https://doi.org/10.1007/978-3-319-93843-1_15

Kanter, J. M., & Veeramachaneni, K. (2015). Deep feature synthesis: Towards automating data science endeavors. *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 1–10. https://doi.org/10.1109/DSAA.2015.7344858

Karumbaiah, S., Ocumpaugh, J., Labrum, M., & Baker, R. S. (2019). Temporally rich features capture variable performance associated with elementary students' lower math self-concept. *Companion Proceedings of the 9th International Learning Analytics and Knowledge Conference (LAK'19)*, 384–388.

Kay, J. (2000). Stereotypes, student models and scrutability. In G. Gauthier, C. Frasson, & K. VanLehn (Eds.), *Proceedings of the 5th International Conference on Intelligent Tutoring Systems* (pp. 19–30). Springer. https://doi.org/10.1007/3-540-45108-0_5

Khajah, M., Lindsey, R. V., & Mozer, M. C. (2016). How deep is knowledge tracing? In T. Barnes, M. Chi, & M. Feng (Eds.), *Proceedings of the 9th International Conference on Educational Data Mining (EDM 2016)* (pp. 94–101). International Educational Data Mining Society.

Kononenko, I. (1994). Estimating attributes: Analysis and extensions of RELIEF. In F. Bergadano & L. D. Raedt (Eds.), *European Conference on Machine Learning (ECML 94)* (pp. 171–182). Berlin Heidelberg: Springer.

Kuhn, M. (2008). Building predictive models in R using the caret package. *Journal of Statistical Software*, *28*(5), 1–26.

Lang, M., Binder, M., Richter, J., Schratz, P., Pfisterer, F., Coors, S., Au, Q., Casalicchio, G., Kotthoff, L., & Bischl, B. (2019). mlr3: A modern object-oriented machine learning framework in R. *Journal of Open Source Software*, *4*(44), 1903. https://doi.org/10.21105/joss.01903

Le, T. T., Fu, W., & Moore, J. H. (2020). Scaling tree-based automated machine learning to biomedical big data with a feature set selector. *Bioinformatics*, *36*(1), 250–256. https://doi.org/10.1093/bioinformatics/btz470

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

LUNDBERG, S. M., & LEE, S.-I. (2017). A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 30* (pp. 4765–4774). Curran Associates, Inc.

MOHAMAD, N., AHMAD, N. B., JAWAWI, D. N. A., & HASHIM, S. Z. M. (2020). Feature engineering for predicting MOOC performance. *IOP Conference Series: Materials Science and Engineering*, *884*, 012070. https://doi.org/10.1088/1757-899X/884/1/012070

OLSON, R. S., URBANOWICZ, R. J., ANDREWS, P. C., LAVENDER, N. A., KIDD, L. C., & MOORE, J. H. (2016). Automating biomedical data science through tree-based pipeline pptimization. In G. Squillero & P. Burelli (Eds.), *Applications of Evolutionary Computation* (pp. 123–137). Springer International Publishing. https://doi.org/10.1007/978-3-319-31204-0_9

PAQUETTE, L., BAKER, R. S., DE CARVALHO, A., & OCUMPAUGH, J. (2015). Cross-system transfer of machine learned and knowledge engineered models of gaming the system. In F. Ricci, K. Bontcheva, O. Conlan, & S. Lawless (Eds.), *Proceedings of the 23rd International Conference on User Modeling, Adaptation and Personalization (UMAP 2015)* (pp. 183–194). Springer International Publishing. https://doi.org/10.1007/978-3-319-20267-9_15

PAQUETTE, L., DE CARVAHLO, A. M. J. A., BAKER, R. S., & OCUMPAUGH, J. (2014). Reengineering the feature distillation process: A case study in detection of gaming the system. *Proceedings of the 7th International Conference on Educational Data Mining (EDM 2014)*, 284–287.

PARDOS, Z. A., FAN, Z., & JIANG, W. (2019). Connectionist recommendation in the wild: On the utility and scrutability of neural networks for personalized course guidance. *User Modeling and User-Adapted Interaction*, *29*(2), 487–525. https://doi.org/10.1007/s11257-019-09218-7

PARDOS, Z. A., TANG, S., DAVIS, D., & LE, C. V. (2017). Enabling real-time adaptivity in MOOCs with a personalized next-step recommendation framework. *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale*, 23–32. https://doi.org/10.1145/3051457.3051471

PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., VANDERPLAS, J., PASSOS, A., COURNAPEAU, D., BRUCHER, M., PERROT, M., & DUCHESNAY, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

PIECH, C., BASSEN, J., HUANG, J., GANGULI, S., SAHAMI, M., GUIBAS, L. J., & SOHL-DICKSTEIN, J. (2015). Deep knowledge tracing. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 28 (NIPS 2015)* (pp. 505–513). Curran Associates, Inc.

RICKER, N. (1953). The form and laws of propagation of seismic wavelets. *Geophysics*, *18*(1), 10–40. https://doi.org/10.1190/1.1437843

ROSÉ, C. P., MCLAUGHLIN, E. A., LIU, R., & KOEDINGER, K. R. (2019). Explanatory learner models: Why machine learning (alone) is not the answer. *British Journal of Educational Technology*, *50*(6), 2943–2958. https://doi.org/10.1111/bjet.12858

SANYAL, D., BOSCH, N., & PAQUETTE, L. (2020). Feature selection metrics: Similarities, differences, and characteristics of the selected models. *Proceedings of the 13th International Conference on Educational Data Mining (EDM 2020)*, 212–223.

SEGEDY, J. R., KINNEBREW, J. S., & BISWAS, G. (2015). Using coherence analysis to characterize self-regulated learning behaviours in open-ended learning environments. *Journal of Learning Analytics*, *2*(1), 13-48-13–48. https://doi.org/10.18608/jla.2015.21.3

SEN, A., PATEL, P., RAU, M. A., MASON, B., NOWAK, R., ROGERS, T. T., & ZHU, X. (2018). Machine beats human at sequencing visuals for perceptual-fluency practice. In K. E. Boyer & M. Yudelson (Eds.), *Proceedings of the 11th International Conference on Educational Data Mining (EDM 2018)*. International Educational Data Mining Society. https://eric.ed.gov/?id=ED593113

SHAHROKHIAN GHAHFAROKHI, B., SIVARAMAN, A., & VANLEHN, K. (2020). Toward an automatic speech classifier for the teacher. In I. I. Bittencourt, M. Cukurova, K. Muldner, R. Luckin, & E. Millán (Eds.), *Proceedings of the 21st International Conference on Artificial Intelligence in Education (AIED 2020)* (pp. 279–284). Springer International Publishing. https://doi.org/10.1007/978-3-030-52240-7_51

SIMARD, P. Y., AMERSHI, S., CHICKERING, D. M., PELTON, A. E., GHORASHI, S., MEEK, C., RAMOS, G., SUH, J., VERWEY, J., WANG, M., & WERNSING, J. (2017). Machine teaching: A new paradigm for building machine learning systems. *ArXiv:1707.06742 [Cs, Stat]*. http://arxiv.org/abs/1707.06742

STANDEN, P. J., BROWN, D. J., TAHERI, M., TRIGO, M. J. G., BOULTON, H., BURTON, A., HALLEWELL, M. J., LATHE, J. G., SHOPLAND, N., GONZALEZ, M. A. B., KWIATKOWSKA, G. M., MILLI, E., COBELLO, S., MAZZUCATO, A., TRAVERSI, M., & HORTAL, E. (2020). An evaluation of an adaptive learning system based on multimodal affect recognition for learners with intellectual disabilities. *British Journal of Educational Technology*, *51*(5), 1748–1765. https://doi.org/10.1111/bjet.13010

THORNTON, C., HUTTER, F., HOOS, H. H., & LEYTON-BROWN, K. (2013). Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 847–855. https://doi.org/10.1145/2487575.2487629

TSIAKMAKI, M., KOSTOPOULOS, G., KOTSIANTIS, S., & RAGOS, O. (2020). Implementing AutoML in educational data mining for prediction tasks. *Applied Sciences*, *10*(1), 90. https://doi.org/10.3390/app10010090

VISWANATHAN, S. A., & VANLEHN, K. (2019). Collaboration detection that preserves privacy of students' speech. In S. Isotani, E. Millán, A. Ogan, P. Hastings, B. McLaren, & R. Luckin (Eds.), *Proceedings of the 20th International Conference on Artificial Intelligence in Education (AIED 2019)* (pp. 507–517). Springer International Publishing. https://doi.org/10.1007/978-3-030-23204-7_42

XIONG, X., ZHAO, S., VAN INWEGEN, E. G., & BECK, J. E. (2016). Going deeper with deep knowledge tracing. *Proceedings of the 9th International Conference on Educational Data Mining (EDM 2016)*, 545–550.

ZEHNER, F., HARRISON, S., EICHMANN, B., DERIBO, T., BENGS, D., ANDERSON, N., & HAHNEL, C. (2020). The NAEP EDM competition: On the value of theory-driven psychometrics and machine learning for predictions based on log data. In A. N. Rafferty, J. Whitehill, V. Cavalli-Sforza, & C. Romero (Eds.), *Proceedings of The 13th International Conference on*

Educational Data Mining (EDM 2020) (pp. 302–312). International Educational Data Mining Society.

ZOPH, B., & LE, Q. V. (2017). Neural architecture search with reinforcement learning. *ArXiv:1611.01578 [Cs]*. http://arxiv.org/abs/1611.01578