Machine Learning Engineer Nanodegree Capstone
Donia Robinson
KDD Cup 2010 Educational Data Mining Challenge

Definition

Project Overview

Problem Domain

Mined data in the education domain has many possible uses. One such use is to predict future performance based on day-to-day tutoring methods. The 2010 KDD Cup uses the term "assistment", which means to assess performance while simultaneously assist learning. How might machine learning assess how similar or dissimilar problems are, and know which problems a student would benefit from seeing more of? It would be useful if a student could ultimately reach the same level of proficiency, but do so with less learning hours. Those hours can then be spent doing something else. If tutoring is being paid for, this saves money as well.

In this project, student answers to algebra problems in online tutoring programs were recorded. These answers were recorded in such a way that analysis can be done to determine which portions of a problem a student has mastered or needs help with. These are logged as "transactions," and will be discussed in the **Analysis: Data Exploration** section. This is similar to cognitive task analysis, which has been shown to result in improved methods of instruction (Clark, Feldon, van Merriënboer, Yates, & Early, 2007). Simply knowing if a student got a question right or wrong is a start, but knowing which parts of a question the student struggled on is much more valuable.

Project Origin

This project originated as an educational data mining challenge called the KDD Cup 2010: https://pslcdatashop.web.cmu.edu/KDDCup/ KDD Cup is the annual Data Mining and Knowledge Discovery competition organized by ACM Special Interest Group on Knowledge Discovery and Data Mining http://www.sigkdd.org/ (KDD), the leading professional organization of data miners. It was held as part of 16th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2010) in Washington, DC, July 25-28. The workshop schedule (available here: http://pslcdatashop.org/KDDCup/workshop/) included invited discussions, as well as presentations by the teams with the best scores on the challenge sets. The challenge was believed to be both of interest scientifically and useful practically.

Datasets and Inputs

Because this project was originally a competition, the necessary datasets are well-defined. There are 5 data sets. There are a minimum of 19 inputs per data set, which I will discuss in detail in the **Analysis: Data Exploration: Feature Discussion** section. Of note, there are a few groups of features that I anticipate being especially useful. The contents of the "Correct First Attempt" column becomes the labels for the data. Ultimately, this is what the algorithm is supposed to learn. "KC Traced Skills," which lists the skills used in that problem, will allow the problems to be connected via mathematical topic. "Step Name," if parsed out, may help with this as well. The "Hints" column will be interesting to look at. For example, does a student who requests a lot of hints generally go on to get the problem correct or incorrect? The time measurements may also provide insight.

Citation of data, per the KDD Cup website: "The project will use 5 data sets from 2 different tutoring systems. These data sets come from multiple schools over multiple school years. The systems include the the Carnegie Learning Algebra system, deployed 2005-2006 and 2006-2007, and the Bridge to Algebra system, deployed 2006-2007. The development data sets have previously been used in research and are available through the Pittsburgh Science of Learning Center DataShop (also: https://pslcdatashop.web.cmu.edu/KDDCup/rules_data_format.jsp)."

Problem Statement

The overall problem to be solved in this project is predicting whether or not a student got a step right (the first time) on a math problem. The prediction will be a numerical value of either 0 or 1.

The first step in solving this problem was to choose which items would make up the algorithm. Generally speaking, you need the following:

- an estimator (regressor or classifier)
- a parameter space
- a method for searching
- a cross-validation scheme
- a score function

So, for starters, an estimator needed to be chosen. Because a value between 0 and 1 should be returned, I looked into probabilistic classifiers. After some experimenting, I could see the Random Forest Classifier was providing the best results. Another consideration at this point, then, was what kind of pre-processing would need to be done to turn the data into a workable format for this classifier (and the upcoming search). Because this turned out to be sparse data, TruncatedSVD ended up working out as the best means for taking the data to a lower dimensional space. The text data would need to be converted into a vector representation, clustered, and relabeled. For additional information, see the **Methodology: Data Preprocessing** section below. (This portion here is intended to be a high level overview.)

Next, I chose GridSearchCV as my search method. Throughout previous coursework, I had felt like this search produced the best results. For the parameter space, I began with fairly common settings and did little tuning on them. Later, when refining things, GridSearchCV would prove to be a helpful tool in calculating the best hyper-parameters. One other benefit to using GridSearchCV was that Stratified KFold validation could be used. As you'll see below, the data is very imbalanced. A combination of Stratified KFold validation and weighted classes helped combat that.

The last item to set up was the scoring function, though that had been chosen by the competition. The scoring function to be used was RMSE. For more information, see the **Metrics** section below. Additional scoring items were used informally to make decisions about relationships and other smaller items.

After setting up the items above and doing the data preprocessing, the model was fit on the training data. From there, I tuned parameters to get it to perform as well as it possibly could. My goal was to score better than the benchmark prediction of all ones. (This actually turned out to be a fairly difficult mark to hit!) Large teams in the competition scored approximately 6 points better than the benchmark, but they had quite a bit more brainpower working on the problem!

Metrics

The evaluation metric that will be used to quantify the performance of the algorithm is Root Mean Squared Error. As mentioned in the **Datasets and Inputs** section above, one column of the data indicates if the student got the correct answer on the first try or not (0 or 1).

As previously mentioned, the prediction (y_pred) being made by the algorithm is a number between 0 and 1. Root Mean Squared Error will be calculated using the data from the "Correct First Attempt" column (y) and the prediction. RMSE = sqrt(mean((y-y_pred)^2))

Because this was a competition, a scoring method was needed that was quantitative. So, for example, a confusion matrix would not make for a good metric. It would not be immediately discernible what made for a better model. RMSE serves as a good evaluation metric for this problem because it gives a single measurement of how far off all predictions were. It provides what is essentially the magnitude of the error.

Even though the competition is over, the correct classifications for the test "challenge datasets" are not released by the competition organizers. However, by uploading a solution file (in a particular format) to the competition (https://pslcdatashop.web.cmu.edu/KDDCup/upload.jsp), it will calculate the RMSE score. I will include screenshots of the results of my submissions.

Analysis

Data Exploration

There are five datasets associated with this competition. I began by working on two datasets, but ended up focusing my efforts on one dataset. The first challenge dataset (known as overall dataset #4) is loaded here. After loading, several samples are selected and printed.

Feature Discussion

As provided by the competition organizers, each record will be a step that contains the following attributes:

- Row: for challenge data sets, rows are renumbered within each file. They are 1...n for the training file and 1...n for the test/submission file.
- Anon Student Id: unique, anonymous identifier for a student
- Problem Hierarchy: the hierarchy of curriculum levels containing the problem.
- Problem Name: unique identifier for a problem
- Problem View: the total number of times the student encountered the problem so far.
- Step Name: each problem consists of one or more steps (e.g., "find the area of rectangle ABCD" or "divide both sides of the equation by x"). The step name is unique within each problem, but there may be collisions between different problems, so the only unique identifier for a step is the pair of problem_name and step_name.
- Step Start Time: the starting time of the step. Can be null.
- First Transaction Time: the time of the first transaction toward the step.
- Correct Transaction Time: the time of the correct attempt toward the step, if there was one.
- Step End Time: the time of the last transaction toward the step.
- Step Duration (sec): the elapsed time of the step in seconds, calculated by adding all of the durations for transactions that were attributed to the step. Can be null (if step start time is null).
- Correct Step Duration (sec): the step duration if the first attempt for the step was correct.
- Error Step Duration (sec): the step duration if the first attempt for the step was an error (incorrect attempt or hint request).
- Correct First Attempt: the tutor's evaluation of the student's first attempt on the step—1 if correct, 0 if an error.
- Incorrects: total number of incorrect attempts by the student on the step.
- Hints: total number of hints requested by the student for the step.
- Corrects: total correct attempts by the student for the step. (Only increases if the step is encountered more than once.)

- KC(KC Model Name): the identified skills that are used in a problem, where available. A step can have multiple KCs assigned to it. Multiple KCs for a step are separated by

~~ (two tildes). Since opportunity describes practice by knowledge component, the corresponding opportunities are similarly separated by ~~.
- Opportunity(KC Model Name): a count that increases by one each time the student encounters a step with the listed knowledge component. Steps with multiple KCs will have multiple opportunity numbers separated by ~~.

In this data set, the KC models are: SubSkills, KTracedSkills, and Rules.

For the test portion of the challenge data sets, values will not be provided for the following columns:

- Step Start Time
- First Transaction Time
- Correct Transaction Time
- Step End Time
- Step Duration (sec)
- Correct Step Duration (sec)
- Error Step Duration (sec)
- Correct First Attempt
- Incorrects
- Hints
- Corrects

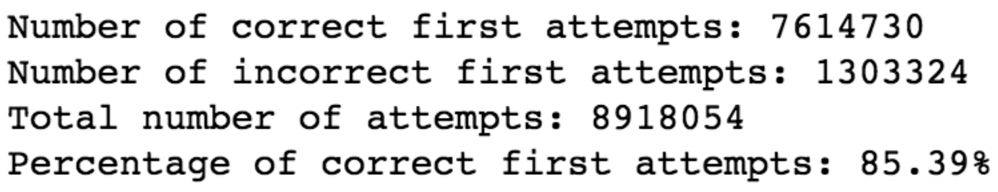
I will not be using any of the opportunity fields for my algorithm. They likely could be useful, but for the scope of the project, there will not be time to make use of all fields. I have also opted to not use the following features for analysis purposes: Row, Anon Student Id, Problem Hierarchy, Problem View.

I initially thought about determining if there was correlation between any of these and what I am trying to predict, and then determining if there was correlation between the fields that I would be given in the challenge data sets and these fields. As I progressed through my work, I had more than enough other relationships to explore, so I ended up simply omitting this data from my algorithm. See additional discussion in **Conclusion: Improvement** section.

Additional work to isolate which features are the most relevant will be done in the **Analysis: Data Exploration: Relevant Statistics** and **Analysis: Exploratory Visualization** sections below.

Relevant Statistics

Below, the percentage of correct first attempts is calculated. It is important to note that the data is quite imbalanced. The positive to negative ratio is approximately 85:15. These weights will be used later in the classification algorithm as a means to manage this imbalance. Other statistics will be mentioned and used in the **Methodology: Data Preprocessing** section.



```
Number of correct first attempts: 7614730
Number of incorrect first attempts: 1303324
Total number of attempts: 8918054
Percentage of correct first attempts: 85.39%
```

Data Abnormalities

The most glaring abnormality of the data is that a large percentage of rows have NaN for fields that will be relevant to classification. The percentage of NaN for three features are listed below the chart (which is shown again for convenience).

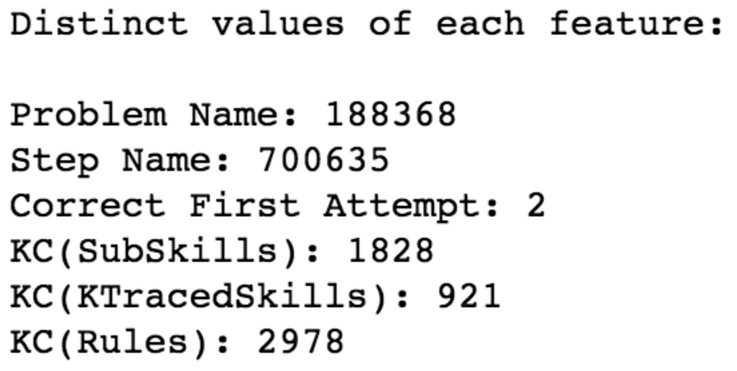
Another abnormality involves the ~~ characters used as dividers in some fields. There are multiple ways these could be handled; for example, the data could be split into additional features. I plan to keep the data together in the feature it came from, but remove the tildes so there is no unexpected behavior.

Abnormalities that need to be addressed will be taken care of in the **Methodology: Data Preprocessing** section below.

Methodology

Data Preprocessing

Almost all of the columns that will be used to train the algorithm are text. Classifiers cannot natively handle relationships between text. Thus, it will be necessary to vectorize the text, cluster it together, and assign a category number to each cluster. In order to figure out which ones would benefit from this, I'll start by listing the number of distinct values each feature takes.



```
Distinct values of each feature:

Problem Name: 188368
Step Name: 700635
Correct First Attempt: 2
KC(SubSkills): 1828
KC(KTracedSkills): 921
KC(Rules): 2978
```

Cluster and convert feature text values to numerical categories

Turning a column of data into a list is very resource intensive, and only needs to be done once and saved to a file. The code I used is contained in external files. I have included the files I generated from this code, in the folder called feature-files. I will outline the pre-processing steps used for each individual feature.

1. Replace NaN with string '' to be able to treat it like an actual value
2. Loop through feature column, appending value to new list
3. Save list to txt file (data is now safely saved, so that work won't have to be done again even if the notebook gets closed)
4. Load list from txt file
5. If applicable, replace ~~ with blank spaces
6. Convert text to a matrix of TF-IDF features
7. Apply Truncated SVD to sparse matrix, to narrow down to two dimensions
8. Cluster 2D values of "text" using Kmeans
9. Take labels assigned to clusters and save them into a new column
10. Save the entire dataframe to a file

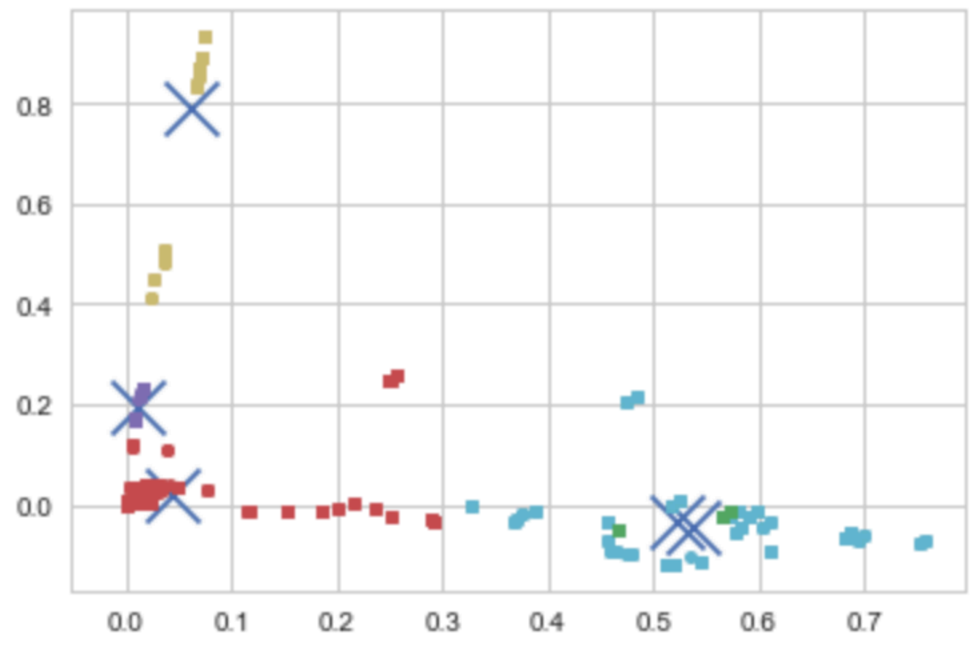


Figure 1. Quick visual proof that kmeans is working the way it is intended to

Analysis: Exploratory Visualization

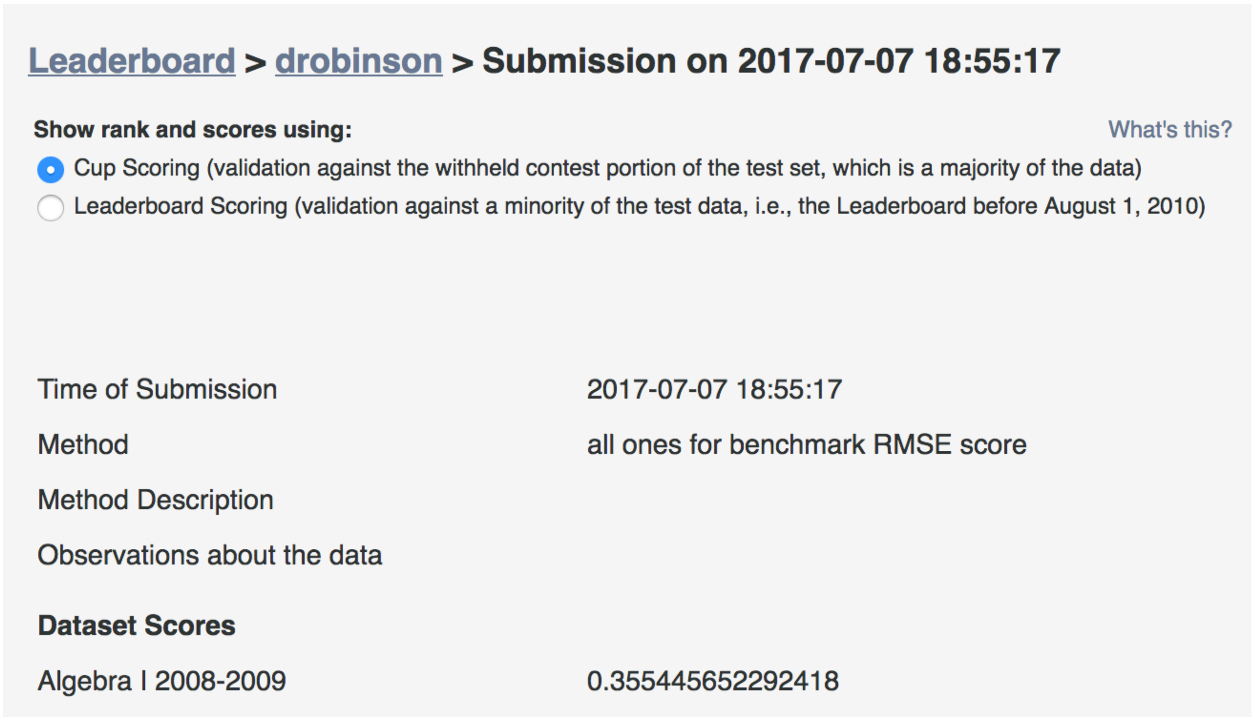
In the section above, I showed an example of kmeans graph I created for a small subset of data for one feature. This graph proved to be too resource intensive to generate for 8 million rows and 200 clusters. However, seeing the small graph was a huge light bulb moment. Working with text data proved to ve very challenging. It wasn't easy to discern how one line of text was related to another. It really showed me that I was on the right track. We had not worked with data like this at any point during the course, so this exploratory visualization, while not relating multiple features to each other, was extremely important.

To determine how features were correlated, a scatter plot would not do. Though I had finally gotten the data to a numerical state, the numbers were arbitrary. I experimented with different types of charts, trying to put my finger on what it would be helpful for me to see during development. This report essentially wrote itself because these truly were the steps I went through to understand the process and plan my next move. So, for the charts below, I used a package called seaborn. The description reads "Seaborn is a Python visualization library based on matplotlib. It provides a high-level interface for drawing attractive statistical graphics." The specific method I used is countplot, which: "Shows the counts of observations in each categorical bin using bars." Each feature was divided into k categories at this point. I plotted the count of "Correct First Attempt" (equal to 0 and 1) by category. What was interesting about these graphs was that there **were** a few categories where the number of 0's outweighed the number of 1's. I felt that these would prove to be the really important categories. With only 15% incorrect answers, anywhere where there were a lot of incorrect answers was highly significant. This was another point in which exploratory visualization was able to demonstrate what I didn't even realize I was looking for proof of!

Analysis: Benchmark

The benchmark I am using is the RMSE score if all 1's are predicted. (Because the data has an overall positive to negative ratio of approximately 85:15, a "1" is a pretty good guess!) Any algorithm I create should at least do better than this.

The code to create my file of all 1's is below. The RMSE score, as returned by that, is approximately 0.3554 (Fig. 2). This is my benchmark.


Leaderboard > drobinson > Submission on 2017-07-07 18:55:17

Show rank and scores using:

What's this?

[✓ Cup Scoring (validation against the withheld contest portion of the test set, which is a majority of the data)]

Leaderboard Scoring (validation against a minority of the test data, i.e., the Leaderboard before August 1, 2010)

Time of Submission	2017-07-07 18:55:17
Method	all ones for benchmark RMSE score
Method Description	
Observations about the data	
Dataset Scores	
Algebra I 2008-2009	0.355445652292418

Figure 2. Benchmark RMSE if all 1's are predicted

Analysis: Algorithm and Techniques

Methodology: Implementation

Reflection

When I wrote my capstone proposal, I had a general idea of how I would solve this problem. (The overall description of the algorithm written out in the **Problem Statement** section above is similar.) However, there are several details about this problem that revealed themselves as I began working on it. I will address these items in detail here, as I feel they are what contributed to the specific choices made.

Techniques

Pre-processing ended up taking a great deal of time. I have heard this is often the case with these projects, but it still surprised me! It was non-trivial to take text data, convert it to a numberical representation, operate on it in higher than a 3D plane, and then calculate relationships. There are many tools that exist in sklearn to do these things, but it took time to research these and learn how to use them. It was quite interesting!

I combined the testing and training sets (provided by the competition) together for the pre-processing portion. Because the text values of the features would not necessarily overlap completely in both files, I needed to ensure my algorithm took into account all text when deciding how to classify it.

I had anticipated using PCA to lower the dimensionality of the data. However, because the data was sparse, I had to look for other methods to do this. TruncatedSVD turned out to work well for this situation.

At points in the process, I saved data that I had manipulated to files. This saved the time of having to run those operations again and again, each time I opened a new kernel. I ended up adding on to these files when I decided to save additional information, but it was always nice knowing that a certain step wouldn't need to be repeated.

Algorithm

As mentioned previously, because a value between 0 and 1 should be returned, I looked into probabilistic classifiers. Random Forest Classifier was chosen because it tends to do well on problems with large datasets, gracefully handles situations where a large amount of data are missing, and generated forests can be saved for future uses.

Due to the nature of the data, the sets cannot be assigned randomly. As previously mentioned, they need to have approximately the same positive to negative ratio (85:15) as the overall dataset. Stratified KFolds was used to handle the data imbalance. I experimented with a package called "imbalanced-learn" but felt it did not give me the results I was looking for.

The overall concept of the project is to take the data provided, convert it into a format that can be input into an algorithm, determine which features are most important to predicting the "Correct First Attempt", build the algorithm, and then evaluate it. It turns out that each of these steps is fairly complicated in its own right. Also, none of these steps can be eliminated. For a more detailed explanation, please see the **Problem Statement** section.

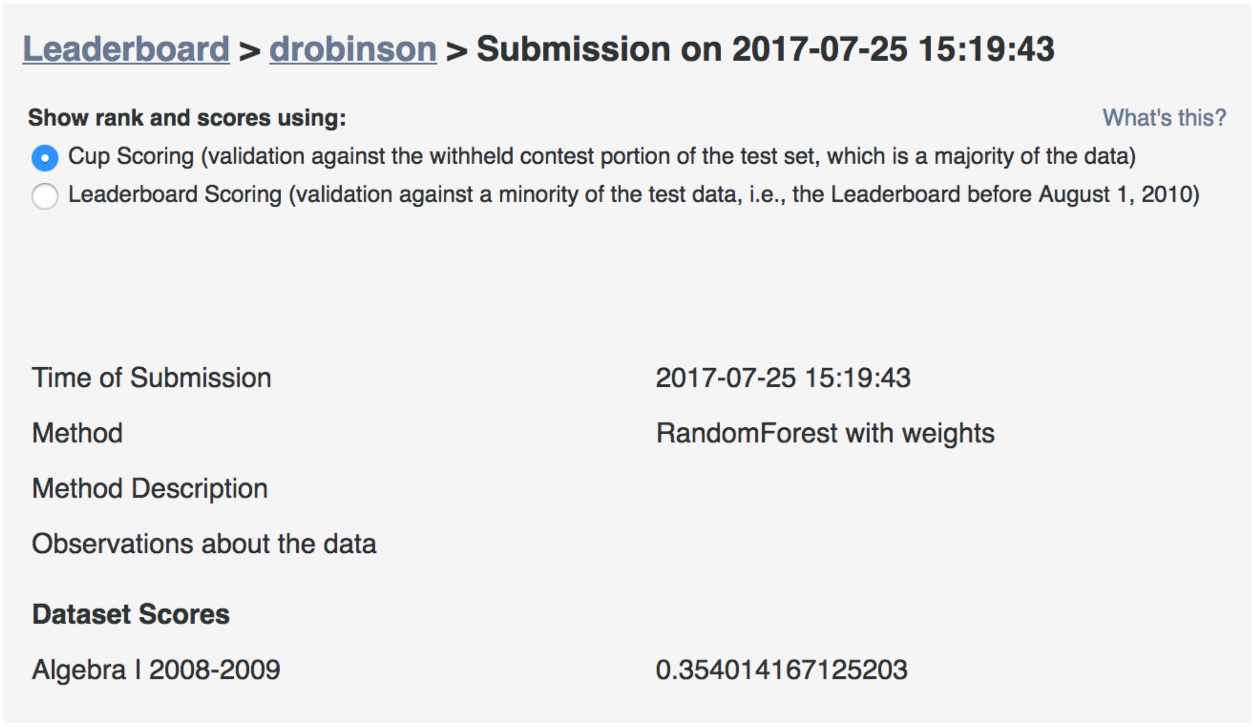
Metrics

While the use of RMSE for scoring the test set was dictated by the competition, I did find it useful to use other metrics at intermediate steps to give me an estimate of how my algorithm might perform. (It wasn't too time consuming to upload results to the competition website, but looking at intermediate results made it so that I didn't have to do this after every run.) I found that calculating the best RMSE and accuracy on the training data (broken into training and testing subsets) was a fairly good indicator of how the algorithm would perform on the competition data. This would indicate the algorithm was not over-fitting too terribly.

Complications

There were several items that complicated this problem. On complication was the sheer size of the data set. I have a computer with quite a bit of processing power, but running the final algorithm on all 8 million rows would take a while. I crashed the kernel many, many times. Another complication was that there were so many possibilities for ways to build and refine the algorithm... virtually an unlimited number. This is a blessing and a curse. For me, it was hard to decide when the RMSE number was "good enough."

Results: Initial



Leaderboard > drobinson > Submission on 2017-07-25 15:19:43

Show rank and scores using:

What's this?

● Cup Scoring (validation against the withheld contest portion of the test set, which is a majority of the data)

○ Leaderboard Scoring (validation against a minority of the test data, i.e., the Leaderboard before August 1, 2010)

Time of Submission	2017-07-25 15:19:43
Method	RandomForest with weights
Method Description	
Observations about the data	

Dataset Scores

Algebra I 2008-2009	0.354014167125203

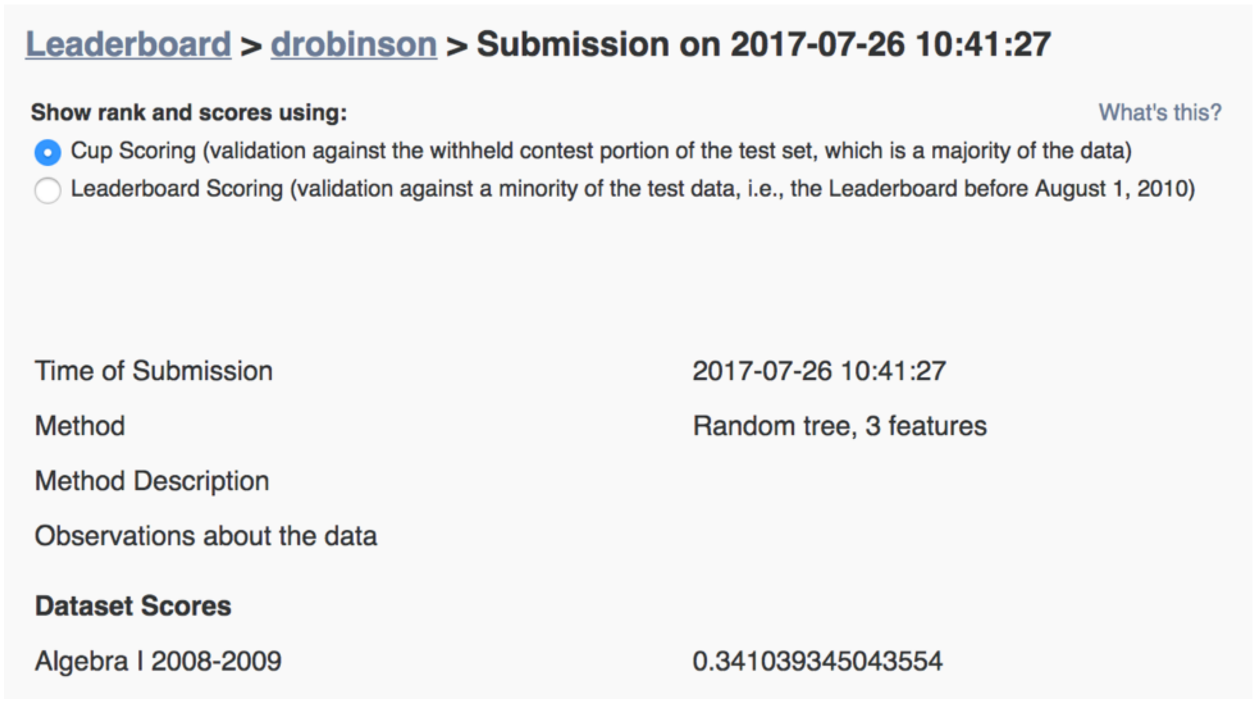
Figure 3. Initial results before any optimization takes place

Methodology: Refinement

After setting up the items above and doing the data preprocessing, the model was fit on the training data. From there, I tuned parameters to get it to perform as well as it possibly could. My goal was to score better than the benchmark prediction of all ones. (This actually turned out to be a fairly difficult mark to hit!) Large teams in the competition scored approximately 6 points better than the benchmark, but they had quite a bit more brainpower working on the problem! The initial solution was posted in the **Methodology: Implementation** section.

As I tuned parameters, I kept watch over the RMSE. Besides tuning the parameters of the algorithm, I experimented with how many clusters each feature got clustered into. Intermediate results are posted here.

Intermediate Solution #1:



Leaderboard > drobinson > Submission on 2017-07-26 10:41:27

Show rank and scores using:

What's this?

[✓ Cup Scoring (validation against the withheld contest portion of the test set, which is a majority of the data)]

Leaderboard Scoring (validation against a minority of the test data, i.e., the Leaderboard before August 1, 2010)

Time of Submission	2017-07-26 10:41:27
Method	Random tree, 3 features
Method Description	
Observations about the data	

Dataset Scores

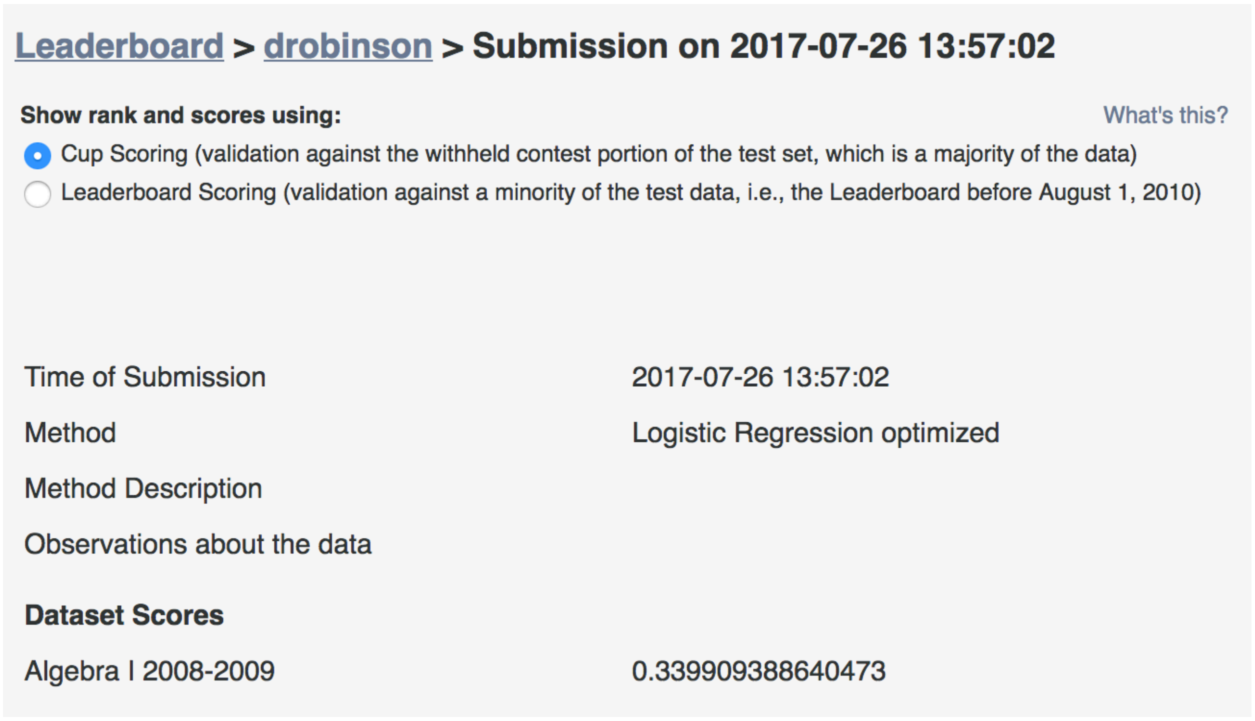
Algebra I 2008-2009	0.341039345043554

Figure 4. RMSE score of intermediate solution #1

```
beginning
batch size: 891805
start fitting model ['Problem Name', 'KC(KTracedSkills)', 'KC(Rules)'] 2017-07-26 08:29:39.29
6890
done fitting model 2017-07-26 09:08:54.416393
GridSearchCV(cv=10, error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True,
            class_weight={0: 0.15000000000000002, 1: 0.85},
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity_split=1e-07,
            min_samples_leaf=10, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=10, verbose=0, warm_start=False),
       fit_params={}, iid=True, n_jobs=1, param_grid={},
       pre_dispatch='2*n_jobs', refit=True,
       scoring=make_scorer(mean_squared_error, greater_is_better=False),
       verbose=0)
Best score -0.137216:
done with this section
```

Figure 5. Description of algorithm for intermediate solution #1

Intermediate Solution #2:



Leaderboard > drobinson > Submission on 2017-07-26 13:57:02

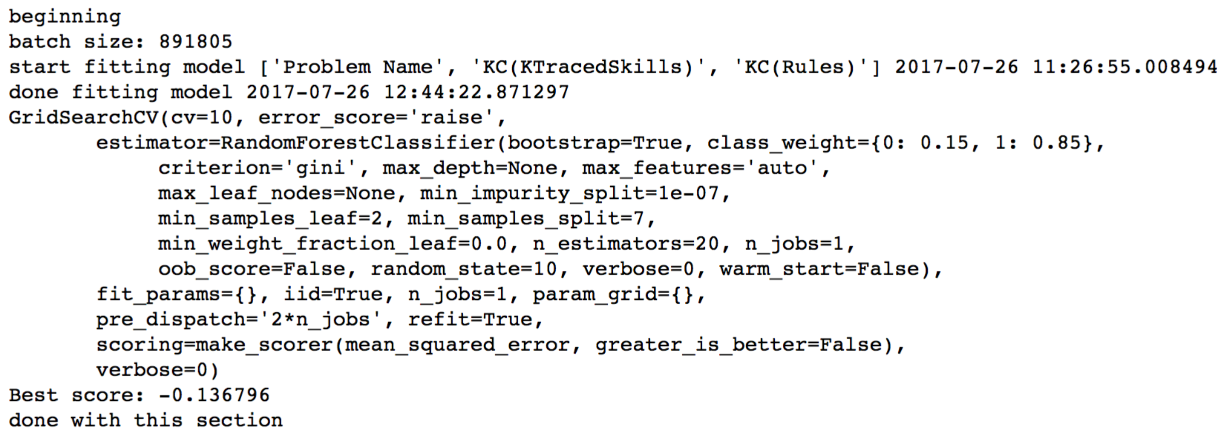
Show rank and scores using: What's this?

[✓ Cup Scoring (validation against the withheld contest portion of the test set, which is a majority of the data)]

Leaderboard Scoring (validation against a minority of the test data, i.e., the Leaderboard before August 1, 2010)

Time of Submission	2017-07-26 13:57:02
Method	Logistic Regression optimized
Method Description	
Observations about the data	
Dataset Scores	
Algebra I 2008-2009	0.339909388640473

Figure 6. RMSE score of intermediate solution #2



```
beginning
batch size: 891805
start fitting model ['Problem Name', 'KC(KTracedSkills)', 'KC(Rules)'] 2017-07-26 11:26:55.008494
done fitting model 2017-07-26 12:44:22.871297
GridSearchCV(cv=10, error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True, class_weight={0: 0.15, 1: 0.85},
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity_split=1e-07,
            min_samples_leaf=2, min_samples_split=7,
            min_weight_fraction_leaf=0.0, n_estimators=20, n_jobs=1,
            oob_score=False, random_state=10, verbose=0, warm_start=False),
       fit_params={}, iid=True, n_jobs=1, param_grid={},
       pre_dispatch='2*n_jobs', refit=True,
       scoring=make_scorer(mean_squared_error, greater_is_better=False),
       verbose=0)
Best score: -0.136796
done with this section
```

Figure 7. Description of algorithm for intermediate solution #2

The final solution is reported in the next section.

Results

Model Evaluation and Validation

I tried a variety of classifiers, but found Random Forest Classifier worked the best. As I began to tune the parameters, I looked back on my past projects to get initial ideas for which parameters to tune and what sorts of values to use. The large number of rows somewhat impaired my ability to experiment. Trying out any more than one or two combinations could mean a 2-hour process run, which made it hard to continue working!

I systematically went through and tried different values for parameters. The ones below, for KBest (“feature_selection”) and RandomForest (“classifier”), produced the best RMSE number. It also ran in a reasonable amount of time (under an hour).

```
parameters = dict(feature_selection__k=[4],
                  classifier__criterion=["entropy"],
                  classifier__n_estimators=[20],
                  classifier__min_samples_leaf=[2],
                  classifier__min_samples_split=[5],
                  classifier__min_impurity_split=[1e-07]
                 )
```

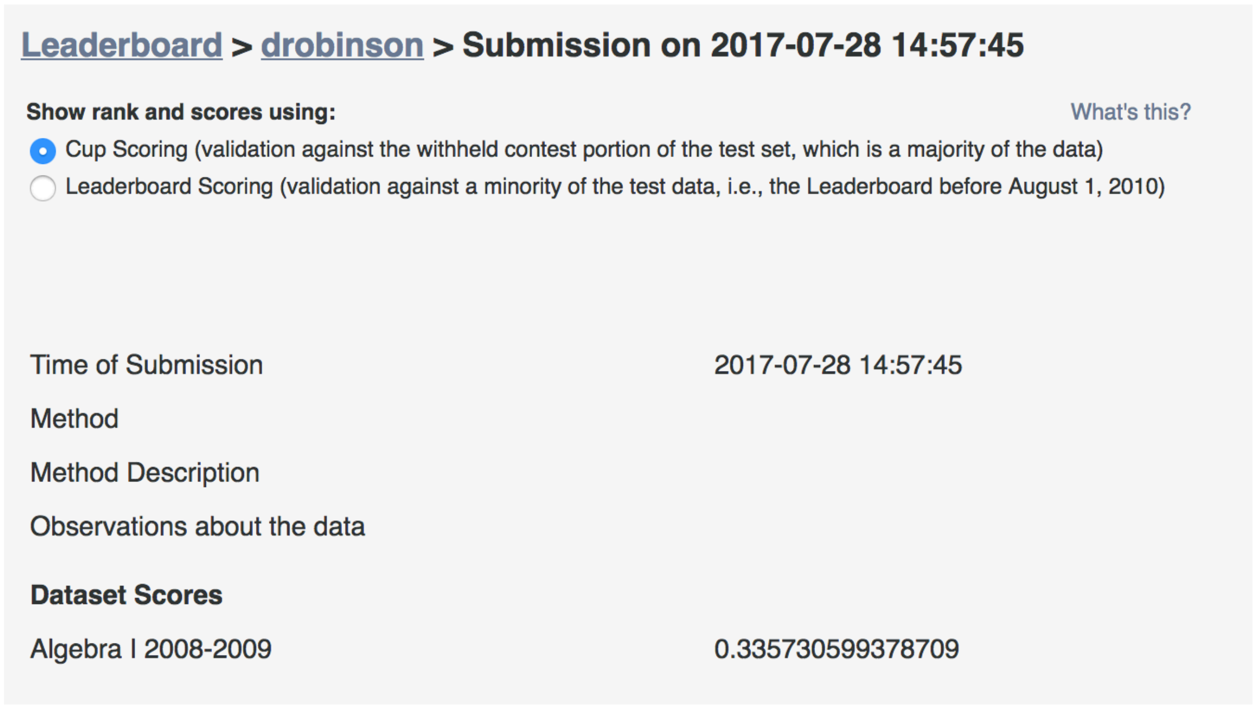
Robustness

There are a variety of ways to evaluate a model for robustness. One method is to use a variation of parameters. “In this approach, the model is run at a set of sample points (different combinations of parameters of concern) or with straightforward changes in model structure (e.g., in model resolution). Sensitivity measures that are appropriate for this type of analysis include the response from arbitrary parameter variation, normalized response and extrema.” (Isukapalli 1999, p.23). Figure 8 shows the results from the model being run with a variety of (reasonable) parameters. As you can see, the RMSE values for the training data remain quite similar. As outlined above, this would indicate a good level of robustness of the model.

	Comp Solution #1	**Comp Solution #2**	**Comp Solution #3**	**Final Solution**
classifier__criterion	gini	gini	entropy	entropy
classifier__n_estimators	10	20	5	20
classifier__min_samples_leaf	10	2	2	2
classifier__min_samples_split	2	7	5	5
classifier__min_impurity_split	1.00E-07	1.00E-07	1.00E-06	1.00E-07
RMSE score on training data	**-0.137216**	**-0.136796**	**-0.134482**	**-0.134427**

Figure 8. RMSE scores of training sets with various parameters

Final Solution Scores



Leaderboard > drobinson > Submission on 2017-07-28 14:57:45

Show rank and scores using: What's this?

[✓ Cup Scoring (validation against the withheld contest portion of the test set, which is a majority of the data)]

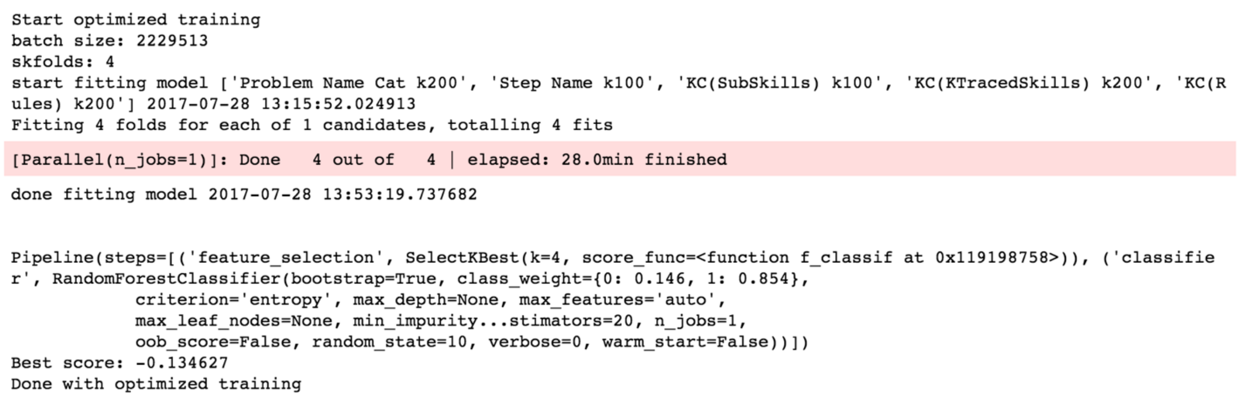
Leaderboard Scoring (validation against a minority of the test data, i.e., the Leaderboard before August 1, 2010)

Time of Submission	2017-07-28 14:57:45
Method	
Method Description	
Observations about the data	

Dataset Scores

Algebra I 2008-2009	0.335730599378709

Figure 9. RMSE score of final solution



```
Start optimized training
batch size: 2229513
skfolds: 4
start fitting model ['Problem Name Cat k200', 'Step Name k100', 'KC(SubSkills) k100', 'KC(KTracedSkills) k200', 'KC(R
ules) k200'] 2017-07-28 13:15:52.024913
Fitting 4 folds for each of 1 candidates, totalling 4 fits
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed: 28.0min finished
done fitting model 2017-07-28 13:53:19.737682


Pipeline(steps=[('feature_selection', SelectKBest(k=4, score_func=<function f_classif at 0x119198758>)), ('classifie
r', RandomForestClassifier(bootstrap=True, class_weight={0: 0.146, 1: 0.854},
            criterion='entropy', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity...stimators=20, n_jobs=1,
            oob_score=False, random_state=10, verbose=0, warm_start=False))])
Best score: -0.134627
Done with optimized training
```

Figure 10. Description of algorithm for final solution

Justification

As noted in the **Benchmark** section, the RMSE score of a submission of all 1's is considered the benchmark. It is approximately 0.3554 (Fig. 2). The teams who scored well in the competition, getting into the ballpark of 0.27, had 5-30 members. I was not aiming for a particular number, but instead, aimed to keep decreasing my RMSE amount as I went. I feel that for the scope of a Nanodegree capstone project, the score I achieved adequately solved the problem. I feel that if I had more processing power accessible, I could have lowered my final RMSE number more.

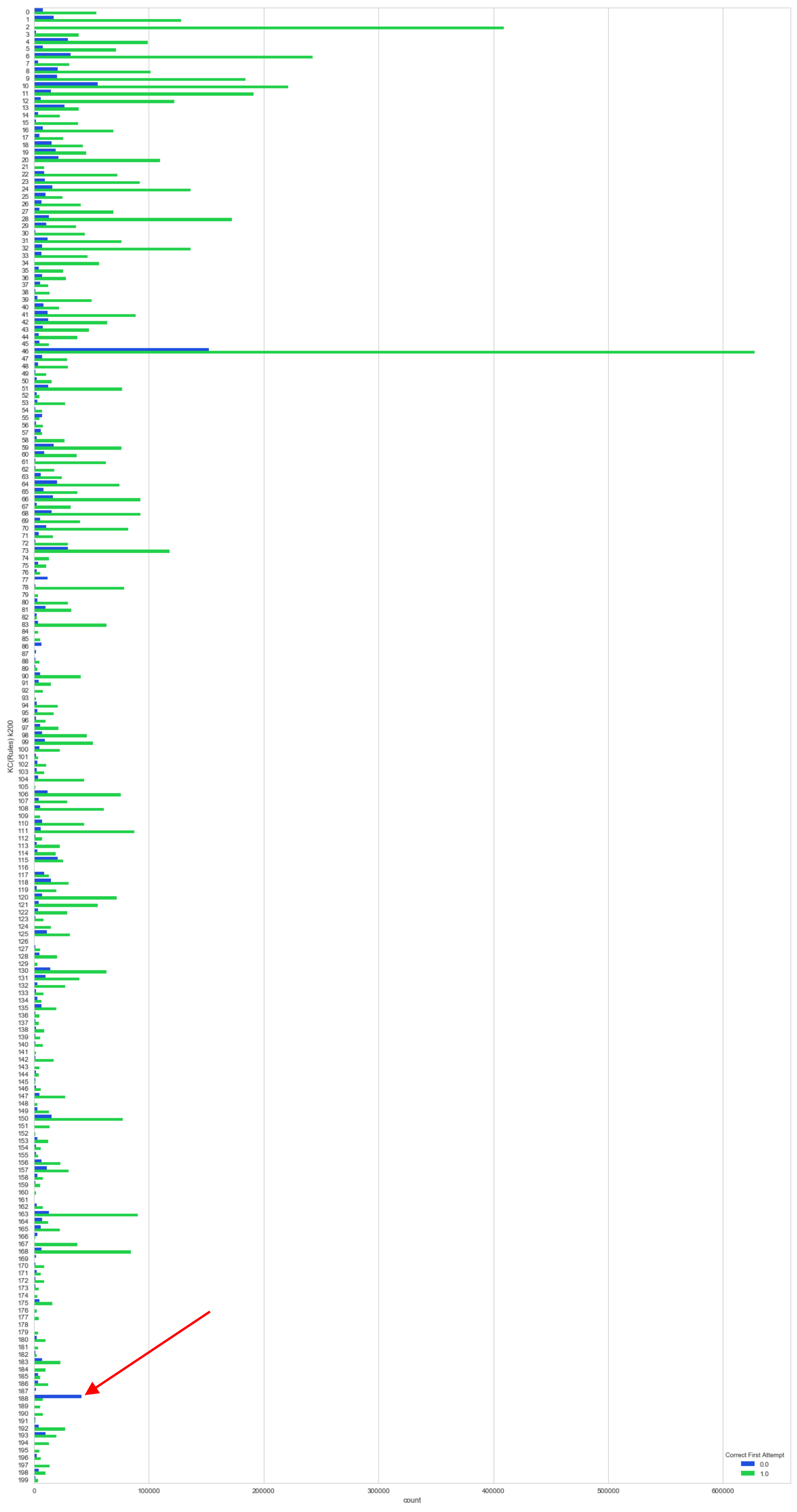
Conclusion

Free-Form Visualization

As mentioned earlier in the report, the data for this project is very imbalanced. For this particular dataset, students correctly answered a step on the first try 85% of the time. This means that just predicting “correct” for all items in the test set produces pretty good results! Thus, one particularly interesting item that I chose to visualize was number of correct vs. incorrect answers by feature category. Figure 11 (next page) shows a chart of this for the feature KC(Rules).

As a reminder, the text for the feature was clustered into 100 or 200 numerical categories. For each of these categories, we can see a count of how many times the student answered the question correctly on the first try and how many times they didn’t. For the majority of categories, students answered the question correctly. This makes sense given our statistical data from above; most of the time, a student answers correctly on the first. However, the rare category in which the count of incorrect answers on the first try is *higher* than the count of correct answers are the ones that are going to impact the model the most. Plotting this chart helped me focus on finding the places where the data was unusual, and therefore significant.

The blue bars indicate incorrect answer counts; green indicate correct answer counts. In nearly every category, the green bar is larger than the blue bar. I have placed a red arrow at a particularly significant category. The blue bar is far larger than the green bar. If the model saw that category of text in the testing set, I would expect it to predict “incorrect.”


KC(Rules) k200
count
0
100000
200000
300000
400000
500000
600000
Correct First Attempt
0.0
1.0

Reflection

For a description of my end-to-end solution, please see the **Problem Statement** section above. I used that section as a place to give a thorough description of the solution, and as a developer, strongly believe in the DRY (Don't Repeat Yourself) philosophy. I think there were two items that I found to be the most difficult. The first was considering topics for my capstone project. In looking at potential projects, there were a number of reasons I picked this one. However, there were so many aspects to it that I perhaps should have seen in advance! For example, processing text data when most of the projects in the course had been numerical data was a huge hurdle to get over. The second item was simply the sheer volume of the data. I'm sure there are much, much larger datasets out there as well, but having dealt with only smaller ones in the course, this large one was another hurdle to overcome.

Improvement

One place for improvement, or change, would be in handling the columns of text all together instead of separately. In some cases, the words in one column are similar to the words in another. If the text of all columns were vectorized and categorized together, would the algorithm be better, worse, or the same at predicting? I would have liked to have tried this, but it would have required significant computing time. This means of categorizing the text would also allow for more reports on correlation of variables. Because each column was assigned category numbers that are meaningless outside of the column, there was no way to determine how the columns might be related. Again, it would be interesting to know if this would have had an effect on the algorithm.

References

1. Clark, R. E., Feldon, D., van Merriënboer, J., Yates, K., & Early, S. (2007). Cognitive task analysis. In J. M. Spector, M. D. Merrill, J. J. G. van Merriënboer, & M. P. Driscoll (Eds.), Handbook of research on educational communications and technology (3rd ed., pp. 577-593). Mahwah, NJ: Lawrence Erlbaum Associates.
2. Stamper, J., Niculescu-Mizil, A., Ritter, S., Gordon, G.J., & Koedinger, K.R. (2010). Algebra I 2008-2009. Challenge data set from KDD Cup 2010 Educational Data Mining Challenge. Find it at http://pslcdatashop.web.cmu.edu/KDDCup/downloads.jsp
3. Isukapalli, S. S. (1999). Uncertainty Analysis of Sport-Transformation Models. Dissertation submitted to the Graduate School at Rutgers, The State University of New Jersey.

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.473.402&rep=rep1&type=pdf