Project Stage 3

Group Members:
Phil Martinkus and Abhijeet Kamble

Entity:
We are matching two tables that contain information about laptops. The data for one table was retrieved in stage 2 from Amazon.com and the other was retrieved in stage 2 from Walmart's website. The amazon data set contains 3102 tuples and the walmart data set contains 3034 tuples. EAch table has the same schema and contains the following fields: name, price, brand, screen size, RAM, hard drive capacity, processor type, processor speed, operation system, and battery life.

Blocking:
We combined several blockers together. To begin with, we used an attribute equivalence blocker on the Brand column. We knew that this would be a good starting place because the brand columns were very clean and no two laptops could match if they are made by different companies. After this step we still had over a million candidate pairs of tuples.

Then, we wanted to run a blocker on the Name column to check if the two laptops have similar names. However, we ran into a problem because the name column for the laptops contains lots of extra information. For example, the name will include information about the memory, hard drive capacity, offers for windows 10 for a year, the processor and lots of other extra information that would match up across lots of laptops. As a solution, we created a new column called 'Clean Name' that removed these common words from the name. We then were able to run a rule based blocker that compared the Clean Name columns using a jaccard score with three-grams. After this step we were able to significantly reduce the candidate set size to just under 70 thousand tuples.

Next, we still thought we could reduce the size of the candidate set further without removing real matches. We then ran a black box blocker that compares the RAM, hard drive capacity, and screen size of a potential pair of laptops. We did some cleaning in these columns to ensure that real matches would not slip through the cracks because of some small errors. Our blocker only lets tuples survive when all three of these columns are equal. We handled missing values here by considering the columns equal for the pair of laptops if either was missing the value. After this last step we reduced the

candidate set to about twenty thousand tuples. We then debugged the blocker and found that the top candidates to be matches were not actual matches and we were convinced that the blocking step was not removing any actual matches.

Labeling:
We then used the sampling and labeling commands in Magellan to label 300 tuples. We found that it could be difficult to tell if some of the laptops were actual matches, but we settled on what constitutes an actual match and completed the labeling stage. For example, we determined that when two tuples were describing the same laptop, but one was refurbished, we would not consider the laptops a match

Selecting a Matcher to Debug:
After the first round of cross validation on the development set, we received the following results:

	Matcher	Average precision	Average recall	Average f1
0	DecisionTree	0.917500	0.919048	0.910598
1	RF	0.940714	0.882937	0.910717
2	SVM	0.757738	0.671825	0.693077
3	NB	0.570000	0.576190	0.544761
4	LogReg	0.670000	0.529365	0.569893
5	LinReg	0.842381	0.630556	0.711829

The Decision Tree and Random Forest perform extremely well while the other models all struggled to make correct predictions.

Debugging a Matcher:
Since it had the highest F1 score, we decided to take a look at the Random Forest to see why it was making the mistakes it was making. We found a common theme of small differences in the Name column determining if the pair was a correct match. For example, as we discussed earlier, we don't consider two laptops to be a pair if one is refurbished and the other isn't even if they are the same laptop. This particular case ended up being a common cause for mistakes for the random forest model so we added a new feature that checks if the work refurbished is in both laptop names in the candidate pair. The feature returns 1.0 if both laptops are (not) refurbished, 0.5 if either laptops is missing the Name, and 0.0 is one laptop is refurbished and the other is not.

In the end, we only needed one round of debugging. This was the case because we spent much of our time in the cleaning and blocking steps. We found that cleaning the data was a necessity in order to get a decent blocking scheme, particularly because the Name field was so dirty and contained information from the other features. By adding in the Clean Name column, cleaning up the Brand, RAM, and Hard Drive Capacity fields, we were able to slim down the candidate set significantly and leave our machine learning models with a far simpler task. Here are the final average cross validation results for each model after debugging:

	Matcher	Average precision	Average recall	Average f1
0	DecisionTree	0.913929	0.919048	0.910598
1	RF	0.938492	0.927778	0.927661
2	SVM	0.784545	0.671825	0.706125
3	NB	0.714405	0.688889	0.682251
4	LogReg	0.728095	0.558532	0.616865
5	LinReg	0.812381	0.655952	0.721434

We would like to note that our new refurbished feature was able to help increase the recall of the Random Forest, Naive Bayes, and Linear Regression models.

Evaluation:
After debugging and settling on a matcher with the development set, we were able to evaluate our models with the evaluation set. After training each model on the entire development set and then testing on the entire evaluation set, we obtained the following results:

Model	**Precision**	**Recall**	**F1**
Decision Tree	92.86	96.3	94.55
Random Forest	100.0	88.89	94.12
SVM	66.67	59.26	62.75
Naive Bayes	56.67	62.96	59.65
Logistic Regression	69.23	33.33	45.0
Linear Regression	89.47	62.96	73.91

Time Estimates:
The majority of our time for this project took place in the blocking stage, mostly because we had to clean our data as well during this stage. Blocking took about a week for us to complete. After that, the project was relatively smooth sailing. Labeling took about an hour, and selecting and debugging the matchers took a few hours to finish.

Recall:
We believe that incorporating some natural language processing tools could be a great way to improve our precision and recall in the future. For example, many laptops feature important information buried in the Name column. This could include the color of the laptop, the model of the laptop, the name of this series of laptops, the processor name and type, and more. We believe that we could use NLP tools to extract this information from the name column to create even more features to compare laptops. When we were debugging the Random Forest, we found that often it was small features like these in the Name column that were the cause of mistakes and we think that using NLP tools would be the most effective way to tackle this problem.

Suggestions for Magellan:
To begin with, it must be noted that one of our team members, Phil Martinkus, is actually on the development team for Magellan so we didn't really go through the same growing pains of learning to use Magellan for the first time as other groups might have.

With that in mind, we were easily able to all get up to speed on how to use Magellan and we found no trouble when using it to complete the assignment. The API guide and Jupyter notebooks were great tools whenever Phil did not remember the exact syntax for a given command.

Overall, we would say that the biggest pain point for us was the blocking stage. It is very difficult to know when to stop blocking, especially when you don't really know how many matches are actually present in the data set. Due to the dirty nature of our data, especially in the Name column, we really didn't know how many matches were in the data. We were forced to iteratively try slightly more aggressive blocking schemes, check if we were killing matches with the debugger, and then take a small sample to ensure we were getting enough actual matches after blocking. Perhaps a useful tool in the future would be something that could give the user a rough estimate of the number of matches in the data. This would have helped us to better figure out when we are done with the blocking step and how aggressive we need to be while creating blocking rules.

Although we did not have time to use it, we feel that the deep learning matcher would be perfect for our data set. Our data fits perfectly into the project's description of a dirty data set because the Name column contains lots of extraneous information that belongs in other columns, even when those other columns are sometimes missing. While we regret that we did not have time to try it ourselves, we thought we should note that it would have been interesting to try on our data.