CS838 Project Report – Stage 3

ENTITY MATCHING

GAURAV MISHRA (gmishra2@wisc.edu)
OM JADHAV (ojadhav@wisc.edu)
PALLAVI M KAKUNJE (kakunje@wisc.edu)

1. Overview:

The goal of this stage of the project is to perform entity matching: identifying entities referring to the same real-world entity.

Following are the specific details of the entity:
Entity Type: Mobile Game Application.
Few examples include games like Angry Birds, Real Boxing, Candy Crush Saga etc.

Data Sources: Two selected data sources are Google Play Store (google table) and Apple Store (apple table). The game data was extracted (in stage one) using scrapy tool and stored in csv format.
Number of tuples in google table: 3789
Number of tuples in apple table: 3697
Each tuple in both the tables describe a single game application. Table schema is of the form *(ID, name, category, developer, rating)*. Our application will use the best matcher to match the tuples from google and apple tables that refer to same game.

We used Magellan EM tool for most of the tasks of this project stage.

2. Blocking:

In this step, we use *Overlap Blocker* for blocking two tables and produce a smaller candidate set of tuple pairs. For the given tables (google & apple) we will assume that two game apps with no sufficient overlap between their name and developer name do not refer to the same game app. We tokenize name and developer name by word and check for overlap of at least two for game name and one word for developer name. This step blocks the tuple pairs with insufficient overlap.

Number of tuple pairs before blocking: 14007933
Number of tuple pairs after blocking: 4307

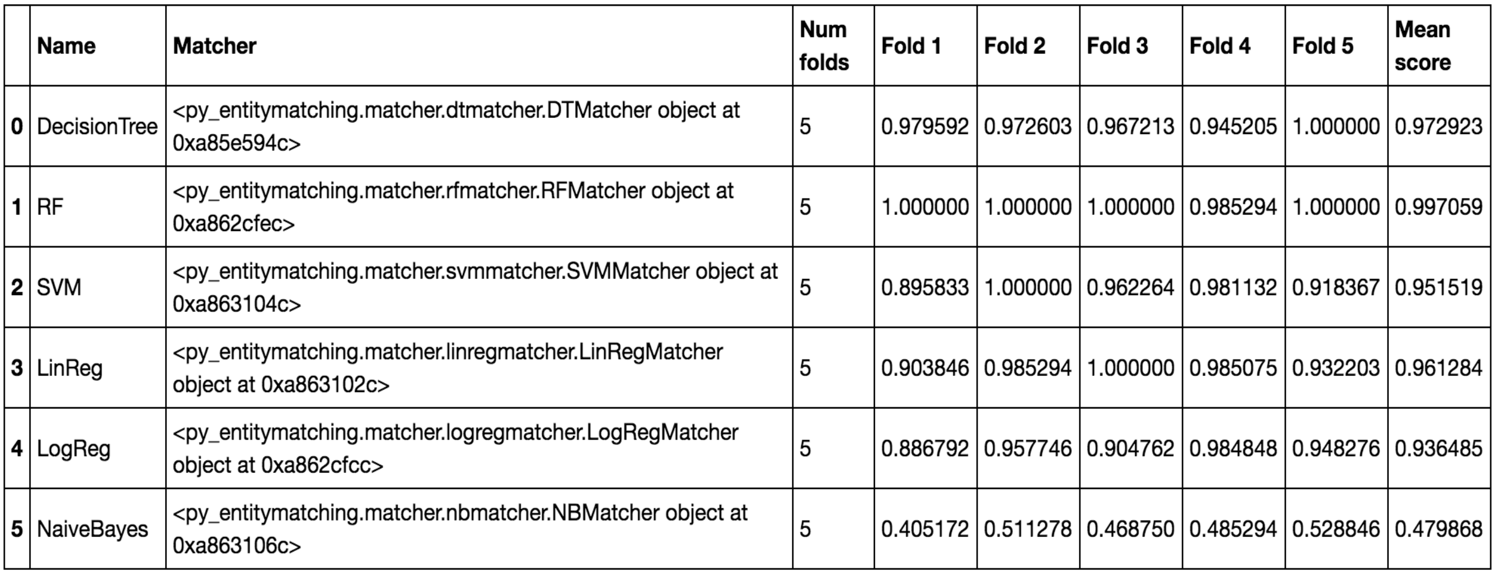
We label the candidate tuple pairs by adding a new table column “label” and populate it with 1 for a match and 0 for a non-match pair, using MS excel. We have 4307 labeled candidate tuple pairs after this step.

3. Matching:

In this step, we split the labeled data into two sets: development (I) and evaluation (J). Specifically, the development set is used to come up with the best learning-based matcher and the evaluation set used to evaluate the selected matcher on unseen data.
Selecting the best learning-based matcher involved, first creating a set of learning-based matchers (Decision Tree, Random Forest, Support Vector Machine, Linear Regression, Logistic Regression and Naive Bayes), creating features and converting the development set into feature vectors using the functions provided by Magellan. This is followed by a k-fold cross validation on the training set to compute precision, recall and F1.

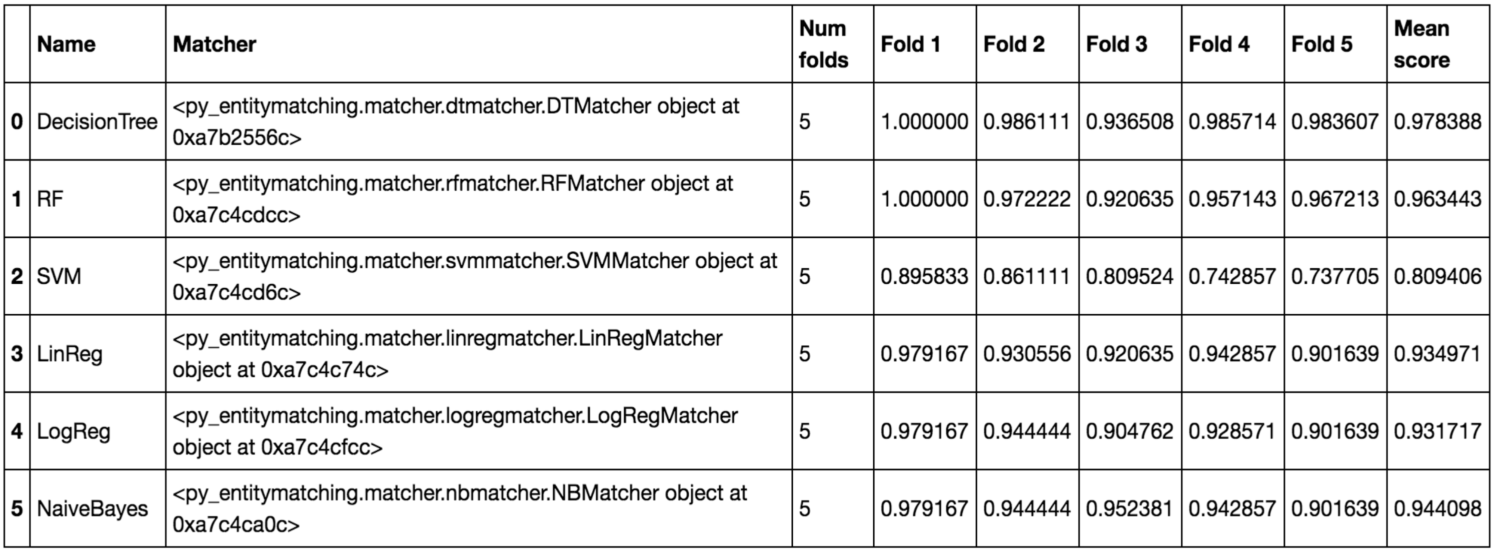
k-fold Cross Validation performed on the training set using multiple classifiers provided the following result:

3.1. Precision:



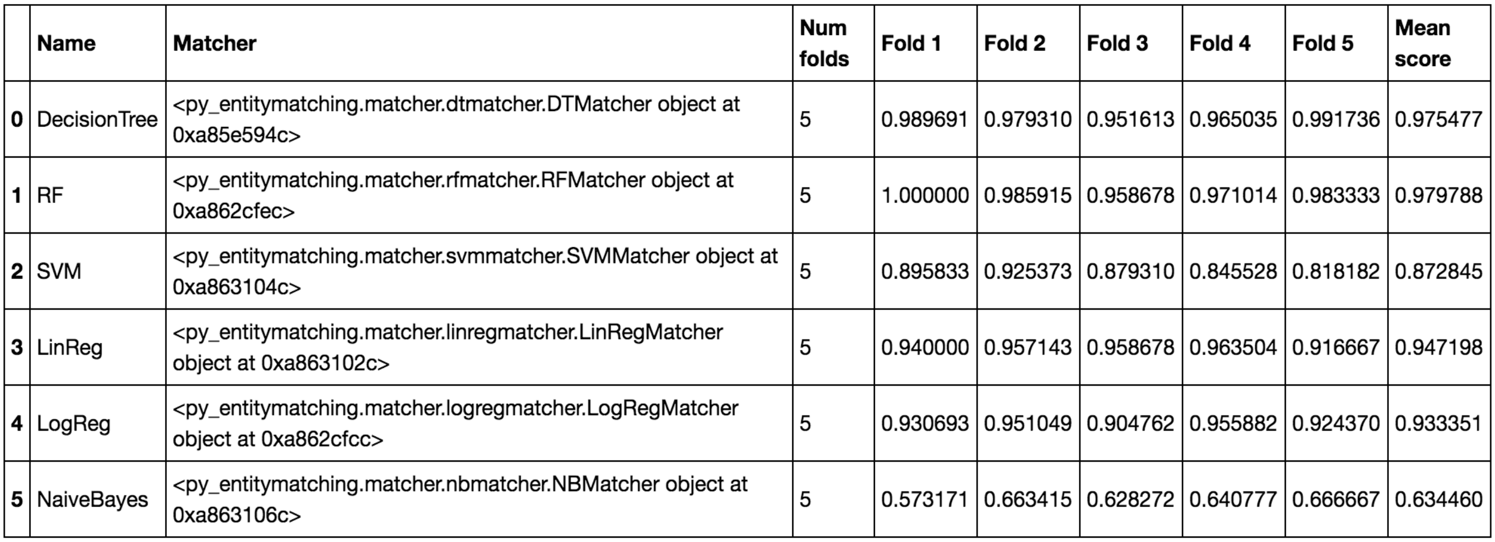
	Name	Matcher	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean score
0	DecisionTree	<py_entitymatching.matcher.dtmatcher.DTMatcher object at 0xa85e594c>	5	0.979592	0.972603	0.967213	0.945205	1.000000	0.972923
1	RF	<py_entitymatching.matcher.rfmatcher.RFMatcher object at 0xa862cfec>	5	1.000000	1.000000	1.000000	0.985294	1.000000	0.997059
2	SVM	<py_entitymatching.matcher.svmmatcher.SVMMatcher object at 0xa863104c>	5	0.895833	1.000000	0.962264	0.981132	0.918367	0.951519
3	LinReg	<py_entitymatching.matcher.linregmatcher.LinRegMatcher object at 0xa863102c>	5	0.903846	0.985294	1.000000	0.985075	0.932203	0.961284
4	LogReg	<py_entitymatching.matcher.logregmatcher.LogRegMatcher object at 0xa862cfcc>	5	0.886792	0.957746	0.904762	0.984848	0.948276	0.936485
5	NaiveBayes	<py_entitymatching.matcher.nbmatcher.NBMatcher object at 0xa863106c>	5	0.405172	0.511278	0.468750	0.485294	0.528846	0.479868

3.2. Recall:



	Name	Matcher	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean score
0	DecisionTree	<py_entitymatching.matcher.dtmatcher.DTMatcher object at 0xa7b2556c>	5	1.000000	0.986111	0.936508	0.985714	0.983607	0.978388
1	RF	<py_entitymatching.matcher.rfmatcher.RFMatcher object at 0xa7c4cdcc>	5	1.000000	0.972222	0.920635	0.957143	0.967213	0.963443
2	SVM	<py_entitymatching.matcher.svmmatcher.SVMMatcher object at 0xa7c4cd6c>	5	0.895833	0.861111	0.809524	0.742857	0.737705	0.809406
3	LinReg	<py_entitymatching.matcher.linregmatcher.LinRegMatcher object at 0xa7c4c74c>	5	0.979167	0.930556	0.920635	0.942857	0.901639	0.934971
4	LogReg	<py_entitymatching.matcher.logregmatcher.LogRegMatcher object at 0xa7c4cfcc>	5	0.979167	0.944444	0.904762	0.928571	0.901639	0.931717
5	NaiveBayes	<py_entitymatching.matcher.nbmatcher.NBMatcher object at 0xa7c4ca0c>	5	0.979167	0.944444	0.952381	0.942857	0.901639	0.944098

3.3. F1:



	Name	Matcher	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean score
0	DecisionTree	<py_entitymatching.matcher.dtmatcher.DTMatcher object at 0xa85e594c>	5	0.989691	0.979310	0.951613	0.965035	0.991736	0.975477
1	RF	<py_entitymatching.matcher.rfmatcher.RFMatcher object at 0xa862cfec>	5	1.000000	0.985915	0.958678	0.971014	0.983333	0.979788
2	SVM	<py_entitymatching.matcher.svmmatcher.SVMMatcher object at 0xa863104c>	5	0.895833	0.925373	0.879310	0.845528	0.818182	0.872845
3	LinReg	<py_entitymatching.matcher.linregmatcher.LinRegMatcher object at 0xa863102c>	5	0.940000	0.957143	0.958678	0.963504	0.916667	0.947198
4	LogReg	<py_entitymatching.matcher.logregmatcher.LogRegMatcher object at 0xa862cfcc>	5	0.930693	0.951049	0.904762	0.955882	0.924370	0.933351
5	NaiveBayes	<py_entitymatching.matcher.nbmatcher.NBMatcher object at 0xa863106c>	5	0.573171	0.663415	0.628272	0.640777	0.666667	0.634460

Based on the reading obtained above, we observe that Random Forest provides best precision (0.997059), recall (0.963443) and F1 score (0.979788). Therefore, ***Random Forest*** is the matcher that best suits our requirement.

We observe that most of the game application names that match have same names in both Google Play Store and Apple Store. As a result, we obtain the required scores (precision, recall and F1) in the first attempt. Hence, we do not go for the debugging step.

4. Testing

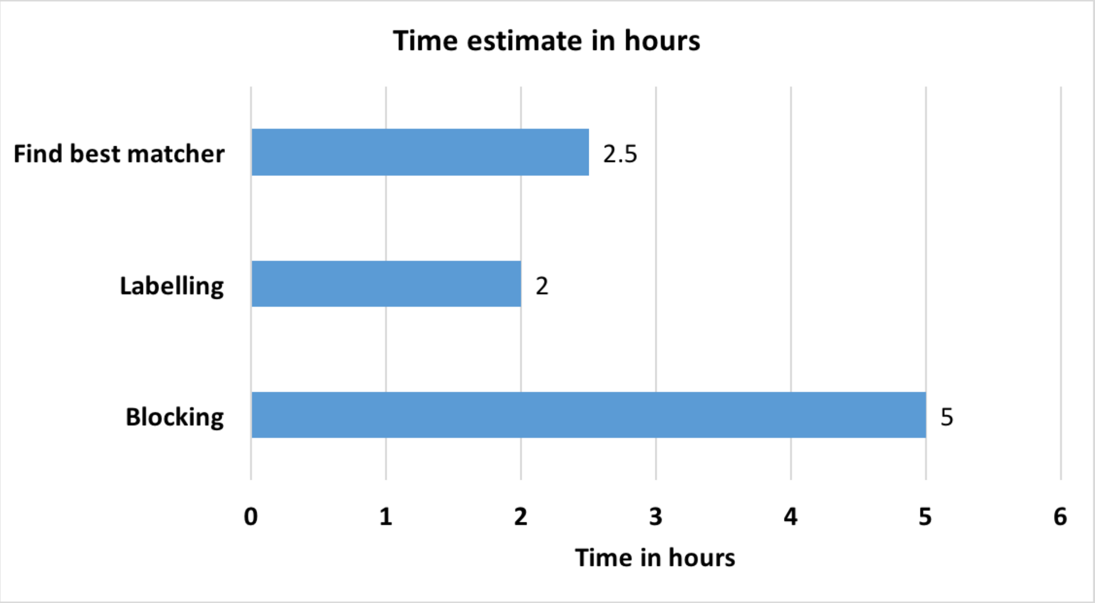
The six learning methods used to train the matcher based on I, and tested on set J gives the following result:

Matcher	**Precision (%)**	**Recall (%)**	**F1 (%)**
Decision Tree	98.87	98.31	98.59
Random Forest	99.43	98.31	98.87
Support Vector Machine	96.77	84.27	90.09
Linear Regression	97.08	93.26	95.13
Logistic Regression	91.76	93.82	92.78
Naive Bayes	58.68	94.94	72.53

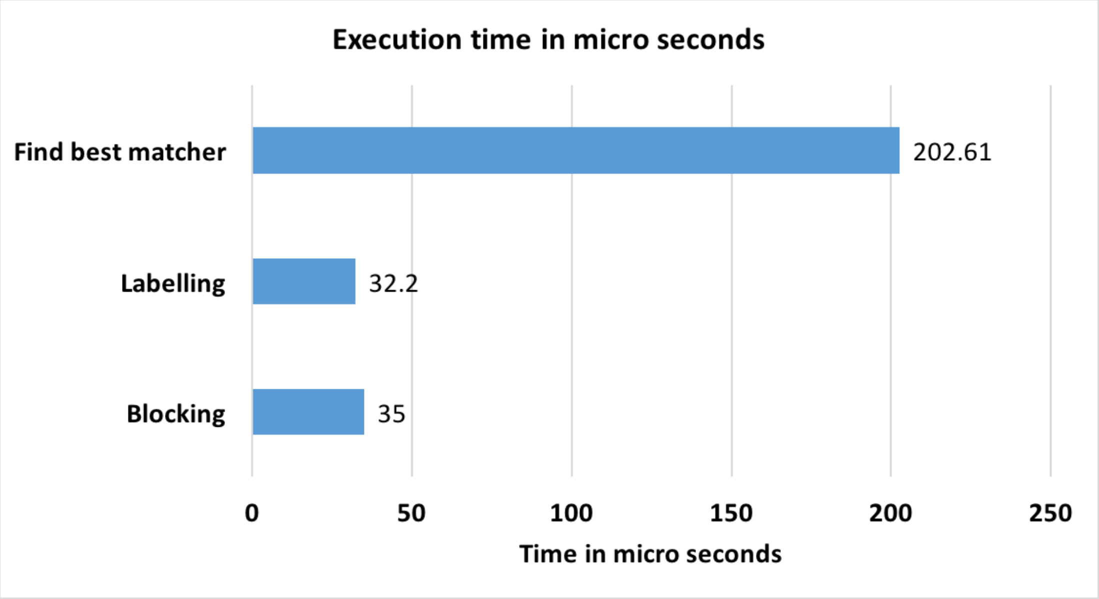
Finally, our best matcher when used to identify matches for the test set (J) gives the following result:
Precision:99.43% (175/176)
Recall: 98.31% (175/178)
F1: 98.87%
False positives: 1 (out of 176 positive predictions)
False negatives: 3 (out of 1117 negative predictions)

5. Time Estimates:

5.1. The graph below shows the approximate number of hours spent in completing different phases of this project stage.



5.2. The graph below shows the execution time for different phases of this project stage.



6. Comments on Magellan (Bonus section):

6.1. Contributions/highlights of Magellan:

- Magellan focuses on entire EM pipeline, not just matching and blocking.
- It provides a powerful scripting environment to facilitate interactive experimentation and allow users to quickly write code to "patch" the system.
- Detailed how-to-guide.
- Magellan is designed to be an open-world system and provides support to a rich set of tools such as pandas, scikit-learn, matplotlib.
- Blocking debugger.
- Matcher debugger.

6.2. Possible improvements:

- Magellan promotes itself as EM tool, but many DS scenarios which needs do data extraction as prior step before EM. Magellan doesn't have the data extraction step in its package, Magellan starts with loading two tables. For example, Magellan can also integrate the tool to scrape data from the web like Scrapy for python and provide the APIs to scrape, clean and convert data to CSV. This is because in most of the DS problems data sources are in the form of websites.
- Also, Magellan can provide some functions which can tell how much time it took for blocking, matching and other important steps. It can also be done by Python inbuilt timeit functions, but if Magellan can provide a detailed analysis of it, for example, how much data has been used for blocking step, and how complex are the blocking algorithms and how much time it takes for particular size of data. This analysis will be helpful for users to take decisions about how much resources to allocate, when run on distributed cluster (same applies for matching step).
- Magellan can also help user in blocking step, running blocking on potential columns with various blocking algorithms after first iteration of EM (using the result of first iteration), and providing the analysis to user and recommended choices of blockers, this way user would have an option to choose a blocker from the recommended list for the next iteration which would improve the accuracy and save the brainstorming time of user with the help of this automation. The same thing can be done for the matcher (or combinations of blockers and matchers - user will be provided with the pair options).
- Magellan can also provide support for tools for the production stage on top of the Python Big Data stack (e.g., Pydoop, mrjob, PySpark, etc.)