Machine Learning Engineer Nanodegree

Record Linkage Using Noisy Demographic Data

Improving Performance using Adaptive Boosting

Chad Acklin
March 3, 2018

I. Definition

Project Overview

In a perfect world, datasets that need to be combined would have a reliable identifier that crosses sources and matches perfectly. That is very rarely the case, though. One of the most important yet arduous tasks of data science is linking data sources that has no unique identifiers and does not match precisely. Linking these data sources involves making decisions that are relatively easy for a human, but difficult for a machine.

A human could easily see that the following two records refer to the same person:

First Name	Last Name	Address 1	City	State	Zip
John	Doe	123 Main St	Louisville	KY	40207
Jonathan	Doe	123 Main Street	St Matthews	KY	40207

Only the Last Name, State, and Zip match exactly, however, and it would be difficult to say with any certainty that the records refer to the same person. A SQL join on any or all of these factors will either miss potential matches or produce a large number of false matches.

This task has been noted and addressed in research literature for many years. Scholars as early as Dunn in 1946 have noted the needs for (among other things) a national system to reliably link records for birth and death to create what he refers to as the "Book of Life." [1] This task is critical in a variety of domains including healthcare[2], business[3], and counter-terrorism.[4]

---

[1] *Dunn, Halbert L. 1946. "Record linkage." American Journal of Public Health and the Nations Health 36(12):1412–1416.*

[2] Golden, Cordell, and National Center for Health Statistics (U.S.). *Linkage of Nchs Population Health Surveys to Administrative Records from Social Security Administration and Centers for Medicare & Medicaid Services, Programs and Collection Procedures*. Vital and Health Statistics, Series 1, Programs and Collection Procedure, No. 58. Hyattsville, Maryland: U.S. Department of Health and Human Services, Centers for Disease Control and Prevention, National Center for Health Statistics, 2015.

The author's specific interest in this project grows out of a real-world problem encountered in industry in which an existing database of customers needs to be integrated with data from OCR scans of paper forms. Due to privacy concerns, the forms do not have identifiers that would make the integration task much easier. A solution is sought to reliably connect the information from the paper form into the existing database using only the demographics collected.

Problem Statement

Given a dataset of original person demographics and a second dataset of matching persons with random noise introduced into the data simulating:

- OCR mistakes
- phonetic misspellings
- number transpositions
- missing data,

classify the records as either a match or non-match.

Dataset

Due to privacy concerns, it is difficult to obtain publicly available examples of real-world data. Because of this, the author will be using synthetically generated data for this project. The data will be generated using the Freely Extensible Biomedical Record Linkage Project.[5] This project provides a data generator that creates records and randomly introduces noise that would be expected from OCR errors, keying errors, and phonetic misspellings.

The data for this task will consist of 30,000 original records with 15,000 matching records. I have generated the data using the FEBRL package and prepared for the machine learning task.

Data Preparation

The data generated is not suitable in its raw form for use in a machine learning algorithm. The approach to building this dataset is to compare original records with potential matches and quantify the text differences between each feature.

---

[3] H. Kopcke, A. Thor and E. Rahm, "Learning-Based Approaches for Matching Web Data Entities," in *IEEE Internet Computing*, vol. 14, no. 4, pp. 23-31, July-Aug. 2010. doi: 10.1109/MIC.2010.58

[4] Gomatam, Shanti, and Michael D Larsen. "Record Linkage and Counterterrorism." *Chance* 17, no. 1 (2012): 25-29. doi:10.1080/09332480.2004.10554883.

[5] Christen, Peter, Churches, Tim and Markus Hegland "Febrl - A Parallel Open Source Data Linkage System: Proceedings of the 8th Pacific-Asia Conference", PAKDD 2004, Sydney, Australia, May 26-28, 2004. Pages 638 - 647. *Springer Lecture Notes in Artificial Intelligence, Volume 3056.*

Blocking

In order to limit the number of comparisons, blocking is usually employed to make a solution more scalable.[6] In our project, for example, the number of comparisons necessary if every record is compared to every potential match is 450M (30,000 * 15,000). As the size of a dataset grows, this quickly becomes impractical.

Traditional blocking involves matching one or more specific data elements in order to create smaller "blocks" within which to do the comparisons. While the data elements are not all equal, they usually share *some* data in common.

In order to generate the dataset for a machine learning model, I first attempted to block records based upon the following criteria:

- Phonetic Matches on First Name, Last Name, City
- Exact Matches, Zip Code, Phone Number, First letter of the last and first names

This resulted in approximately 4.2M potential matches and included 14,989 of our 15,000 True Positives.

Traditional blocking gave satisfactory results, but other blocking methods were explored. A concatenated string was created with the title, first name, last name, address, city, state, zip, and phone. Then, a string matching implementation of TF-IDF was used to find the 10 closest string matches for each original record.

TF-IDF

TF-IDF is a text mining method that weighs the relative frequency of a term (Term Frequency) against the commonality of that term in the entire corpus of text (Inverse Document Frequency). When reading a text document, the term frequency boosts the weight of words that are used often (TF), but reduces the weight of terms that are very common in the entire corpus (IDF).

While usually used with documents and groupings of words, TF-IDF can actually be leveraged to identify groupings of letters within a string of text. [7] In the case of record linkage blocking, the effect of this is that the most unique text strings are upweighted and common text strings are down-weighted. If, for example, 90% of your samples come from the state of Kentucky, then finding the text string "Kent" is actually not very valuable information. However, finding the text string "Cali" might be more important to the string matching process.

The common way to compare TF-IDF vectors is by using a cosine similarity calculation. A package from the data science team at ING bank was used to calculate the cosine similarities.

---

[6] Steorts R.C., Ventura S.L., Sadinle M., Fienberg S.E. (2014) A Comparison of Blocking Methods for Record Linkage. In: Domingo-Ferrer J. (eds) Privacy in Statistical Databases. PSD 2014. Lecture Notes in Computer Science, vol 8744. Springer, Cham

[7] https://bergvca.github.io/2017/10/14/super-fast-string-matching.html

This package introduces some efficiencies and improvements over traditional cosine similarity calculations. The 10 closest matches for each string are retained down to a threshold of 0.1 cosine similarity, effectively limiting the scope of calculations and memory required.

This blocking technique provided several benefits:

1. Speed - TF-IDF and cosine similarity calculations completed in roughly half the time of traditional blocking. On my linux machine, traditional blocking completed in 18 seconds compared to 9 seconds for TF-IDF.
2. Reduction in block size - TF-IDF blocking retained 175k potentially matching records, compared to 4.2M in traditional blocking. By reducing the number of comparisons needed, building and analyzing models can be more thorough and efficient.
3. Completeness of matches - traditional blocking resulted in 4.2M records and missed 11 true positive matches from the data. TF-IDF blocking resulted in 175k records and only missed 1 true positive.
4. Additional features - the cosine similarity becomes a valuable feature in the final dataset used to classify matches.

Feature Engineering

Each original record is compared to the blocked potential matching record and then features are engineered to determine the string similarity between the two datasets. A combination of exact matching and edit distance (Levenshtein distance) of each pair of data elements are employed. Edit distances are calculated using the jellyfish python package. The specific metric used is the Damerau-Levenshtein Distance, which measures the number of edits & transpositions required for two strings to be equal.[8] Edit distances were scaled and inverted so that a value of 1 represents a perfect match and values of <=0 are completely different.

Metrics

Results from the classification task (match/non-match) will be evaluated on precision, recall, and f1-score. Success is defined as a model that obtains a better f1-score than the baseline model on the test set of data.

Precision - The ratio of how many predicted true responses are actually true. Increasing precision is associated with the reduction of “False Positives”

Recall - The ratio of how many actual true responses are predicted as true. Increasing recall is associated with a reduction in “False Negatives”

F1-Score - the weighted average of precision and recall. The F1-score takes into account both False Positives and False Negatives and thus is a good metric to use when evaluating a classification model’s overall efficacy.

[8] http://jellyfish.readthedocs.io/en/latest/comparison.html#damerau-levenshtein-distance

In this case, models have been evaluated on the basis of F1 score because false positives are assumed to be neither better nor worse than false negatives.

II. Analysis

Data Exploration

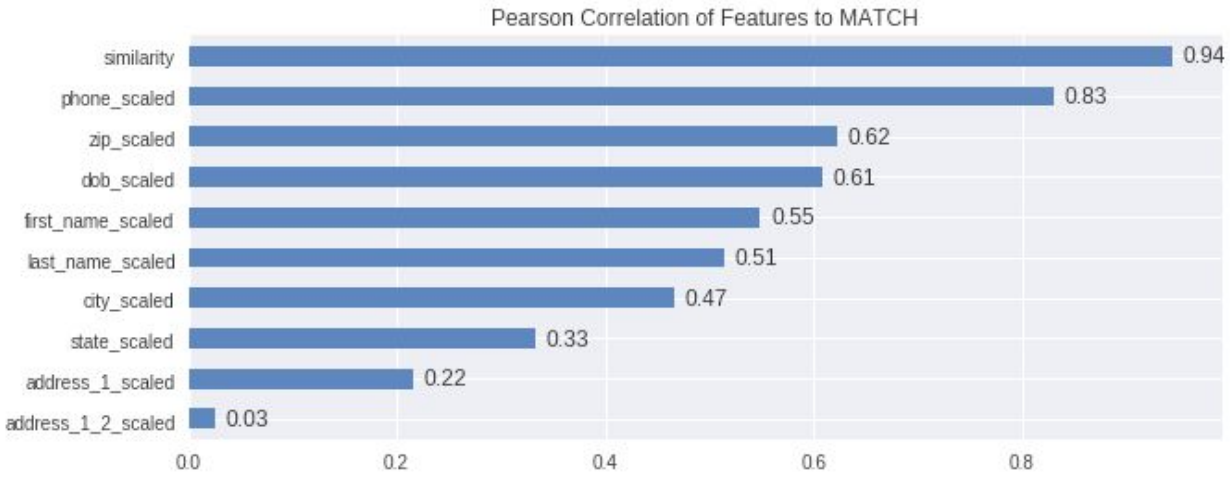
A visual examination of the raw data shows some of the types of variations present in the dataset. There appear to be missing characters, misspellings, and errant spacing that would limit the effectiveness of a deterministic matching strategy.

	rec-0-dup-0	rec-0-dup-1	rec-0-org
gender	f	f	f
age	28	2	28
dob	19901218	19901218	19901218
title			
first_name	caitlin	cait in	caitlin
last_name	chappel	chap pl	chappel
state			
city	pres	preston	preston
zip	4504	4504	4504
street_number			
address_1	g'lbbes street	gibbess reet	gibbes street
address_2			
phone	087631 3909	087631 909	08 76313909

After blocking and data preparation, the similarities between original records and potential duplicates can be quantified. A visual examination of the prepared data indicates that records where MATCH=1 exhibit higher values in almost all features. Of note is that several features of the non-match class could also exhibit high values for some features that are more general. This is intuitive in that a particular state or zip code could have multiple non-matching records.

id_org id_match	rec-0-org rec-0-dup-0	rec-0-org rec-0-dup-1	rec-0-org rec-4547-dup-0	rec-0-org rec-339-dup-1	rec-0-org rec-9649-dup-1
MATCH	1	1	0	0	0
phone_scaled	1	0.9	0.2	0	0.2
last_name_scaled	1	0.8	0.3	0.2	0.3
first_name_scaled	1	0.9	0.4	0.5	0.4
city_scaled	0.7	1	0	0.3	0
state_scaled	1	1	0.7	0.7	0.7
address_1_scaled	0.8	0.8	1	0	0.3
address_1_2_scaled	0	0	0	0	0
dob_scaled	1	1	0.4	0.6	0.4
zip_scaled	1	1	0.5	0.7	0.8
similarity	0.683051	0.652186	0.130793	0.109278	0.107823

Pearson's Correlations between the various features and the dependent variable indicate that the similarity and phone_scaled features are strongly correlated to the MATCH target, but neither are strictly predictive. For example, records with a similarity score between 0.3 and 0.4 are almost equally split between matches and non-matches. Also of note is that 793 perfect matches on phone are actually non-matching records.



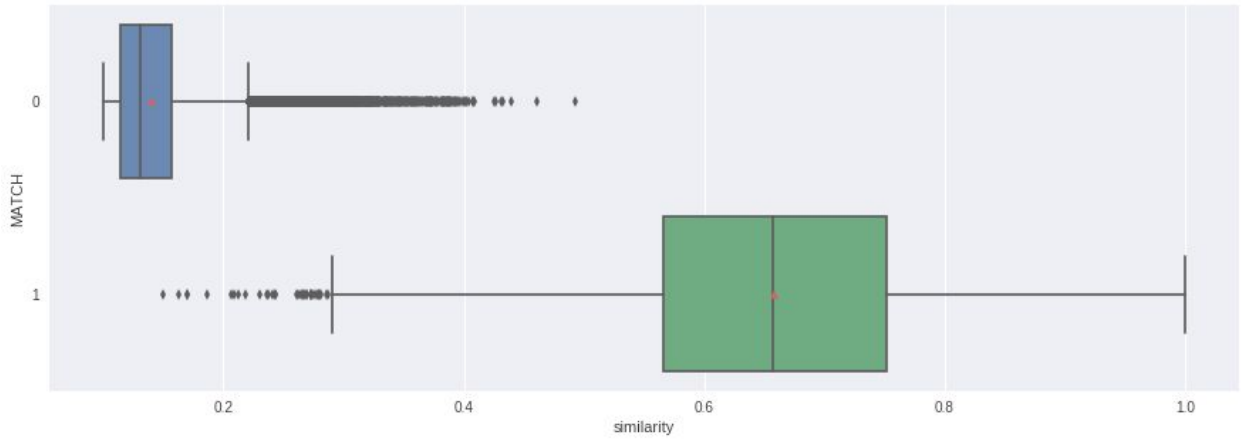
Value Counts by similarity and phone_scaled

Similarity	Non-Match	Match
(0.0991, 0.2]	150,868	5
(0.2, 0.3]	9,498	37
(0.3, 0.4]	333	316
(0.4, 0.5]	14	1,458
(0.5, 0.6]		3,215
(0.6, 0.7]		4,235
(0.7, 0.8]		3,540
(0.8, 0.9]		1,775
(0.9, 1]		418

phone_scaled	Non-Match	Match
0	32,385	167
0.1	43,932	181
0.2	43,490	158
0.3	26,959	84
0.4	9,705	26
0.5	2,469	5
0.6	659	13
0.7	249	81
0.8	67	443
0.9	5	2,044
1	793	11,797

Exploratory Visualization

When comparing the distributions of feature scores grouped by match/non-match, it is helpful to use boxplots and/or histograms to visualize the data.

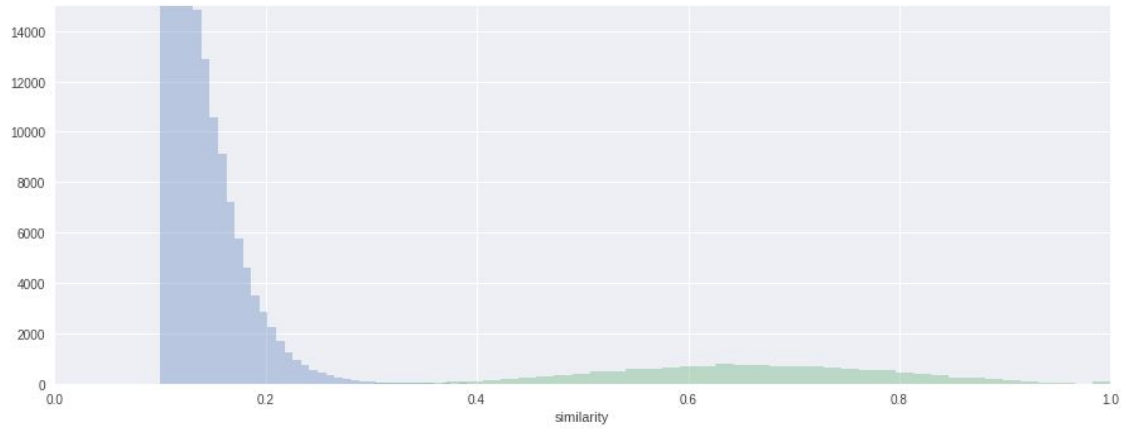


This boxplot tells us several things about the distribution of similarity scores between matches and non-matches. First, you will notices that matching records have a similarity score with a mean of approximately 0.65 and non-matching records have a similarity score with a mean of approximately 0.1.

Also, it is apparent that there is greater variation in the similarity scores of matches than non-matches. The distribution of non-matching records is much more tightly grouped and is also right-skewed. The “whiskers” of the boxplot extend further to the right than the left and the “box” of the plot is much smaller than that of the matches. The matches appear to be

distributed normally with whiskers roughly centered, but the box and whiskers are much longer. One could also think of these as having fatter tails as observed in the histogram below.

Also, the boxplot shows that there are a number of outliers that need to be considered. Matches have outliers in their similarity scores down to about 0.1 and non-matches have similarity score outliers extending as far up as 0.5.



III. Methodology

Implementation

After the data was processed and features engineered, several approaches to the classification problem were considered. Models were implemented in scikit-learn using default hyperparameters in order to determine an approach for a solution. Logistic Regression was considered as a possible generalized linear model. A Random Forest Classifier was evaluated as a potential tree-based model. A boosted model was also run using the Adaboost algorithm. Finally, several architectures were considered to build a Multi-layered Perceptron model.

Algorithms and Techniques

Benchmark - Gaussian Naive Bayes

Traditional approaches to record linkage have been demonstrated to be mathematically equivalent to classification using a Naive Bayes algorithm.[9] Bayes' theorem quantifies the conditional probability of a given event given certain prior knowledge. Mathematically, this can be represented as:

$$P(A|B) = P(B|A) * P(A)/P(B)$$

[9] D. R. Wilson, "Beyond probabilistic record linkage: Using neural networks and complex features to improve genealogical record linkage," *The 2011 International Joint Conference on Neural Networks*, San Jose, CA, 2011, pp. 9-14.

In our case, the probability of a match (A) given the agreement of the text components (B) equals the probability of those text components matching (B) given a match (A) times the probability of a match (A) divided by the probability of the text elements matching (B).

The oft-noted limitation of this strategy is that it assumes the independence of each variable. This is not the case in our problem, but despite this limitation, results have often been adequate for this approach to be used in practice quite often.[10]

The benchmark for this task is generated by building a simple out-of-the-box Gaussian Naive Bayes classifier. This performed surprisingly well considering the previously noted potential for independent variable interactions. At Gaussian Naive Bayes classifier implemented in scikit-learn achieved an f-1 score of 0.9987 and only resulted in 3 false positives and 10 false negatives in the test set.

Logistic Regression

Logistic Regression is a statistical learning approach to classification tasks (usually binary). Logistic regression estimates the probability that a given dependent variable belongs to a particular class based upon the the independent variables given. Logistic Regression models the coefficients, error term, and bias required to linearly separate the classes of data represented by the dependent variable (in our case, match or non-match).

In our case, Logistic regression is a potential solution to the record linkage problem since the data has been labeled with a ground-truth binary dependent variable (match/non-match) and the independent variables are continuous. The computation is efficient and easy to interpret. Logistic Regression does require a large dataset to train reliably, and our data samples should be adequate to build a robust model.

The implemented logistic regression model resulted in an f1-score of 0.9982, which is slightly lower than the benchmark model. Logistic regression resulted in no false positives, but 18 false negatives.

Decision Tree

Decision trees take multiple input variables and build an “if-this-then-that” decision structure that can be visualized in a manner similar to a flowchart. At each variable, a threshold is determined to maximize the amount of information gain. Decision trees are popular due to their simplicity to implement and interpret. Decision trees are considered for this problem due to their speed, simplicity, and interpretability.

Mathematically, decision trees measure the amount of impurity each node and split the data to maximize the homogeneity with regard to the dependent variable. The tree is built by

[10] Wilson, pp 9-14

considering the possible variables and then splitting the tree on the variable that results in the purest resulting groups.

The implemented decision tree model achieved an f1-score of 0.9990 on the testing dataset, which is slightly better than the benchmark model. The decision tree model resulted in 5 false positives and 5 false negatives in the testing set.

AdaBoost

Adaboost (short for Adaptive Boosting) is an ensemble learning method that takes weak learners (which do only slightly better than chance) and combine them iteratively to create a much stronger learner. This algorithm was proposed by Freund and Schapire in 1997,[11] and was one of the earliest and most important boosting algorithms developed.

Adaboost creates weak learners iteratively but trains each weak learner based on the performance of previous learners. Misclassified samples from an iteration are upweighted in subsequent iterations. In effect this allows the algorithm to pay more attention to samples that were classified incorrectly and train to get those right in future iterations. This combination of weak learners can often result in a much stronger overall model. Adaboost is implemented in the scikit-learn python package.

Adaboost is a potential choice for our problem because matches may not necessarily follow a linear decision boundary, we have plenty of data, and additional estimators could make the model more robust to difficult classifications. In record linkage, it is often the case that there are a large number of clear matches and a large number of clear non-matches, but classifying the more the challenging examples is where an algorithm is proven effective or ineffective.

The Adaboost model performed quite well, classifying only two samples incorrectly in the test set. The f1-score of the Adaboost model was 0.9998.

Multilayer Perceptron

Multilayer Perceptrons are a class of neural networks that make use of several layers of perceptrons. Each perceptron is similar to a neuron in the brain. The perceptron takes multiple inputs, assigns weights to each input, sums the weighted values and applies an activation function to produce an output. In a simple multilayer perceptron network, inputs are mapped to multiple perceptrons, one or more hidden layers, and an output layer to produce a classification result. MLP's are generally trained through a process of backpropagation in which a dataset with known labels is fed forward through the network and weights are adjusted iteratively to improve results.

---

[11] Freund, Yoav; Schapire, Robert E (1997). "A decision-theoretic generalization of on-line learning and an application to boosting". *Journal of Computer and System Sciences*. **55**: 119

Multilayered Perceptrons are a potential choice for this task because of their proven capability to correctly label data with complex interrelated features. The dataset is well suited for this task because of the discrete binary output (match or non-match) as well as scaled continuous independent variables. MLP models have been generated and trained using the Keras Python package with a Tensorflow backend. Multiple architectures have been considered using both deeper and wider network architectures to obtain optimal results.

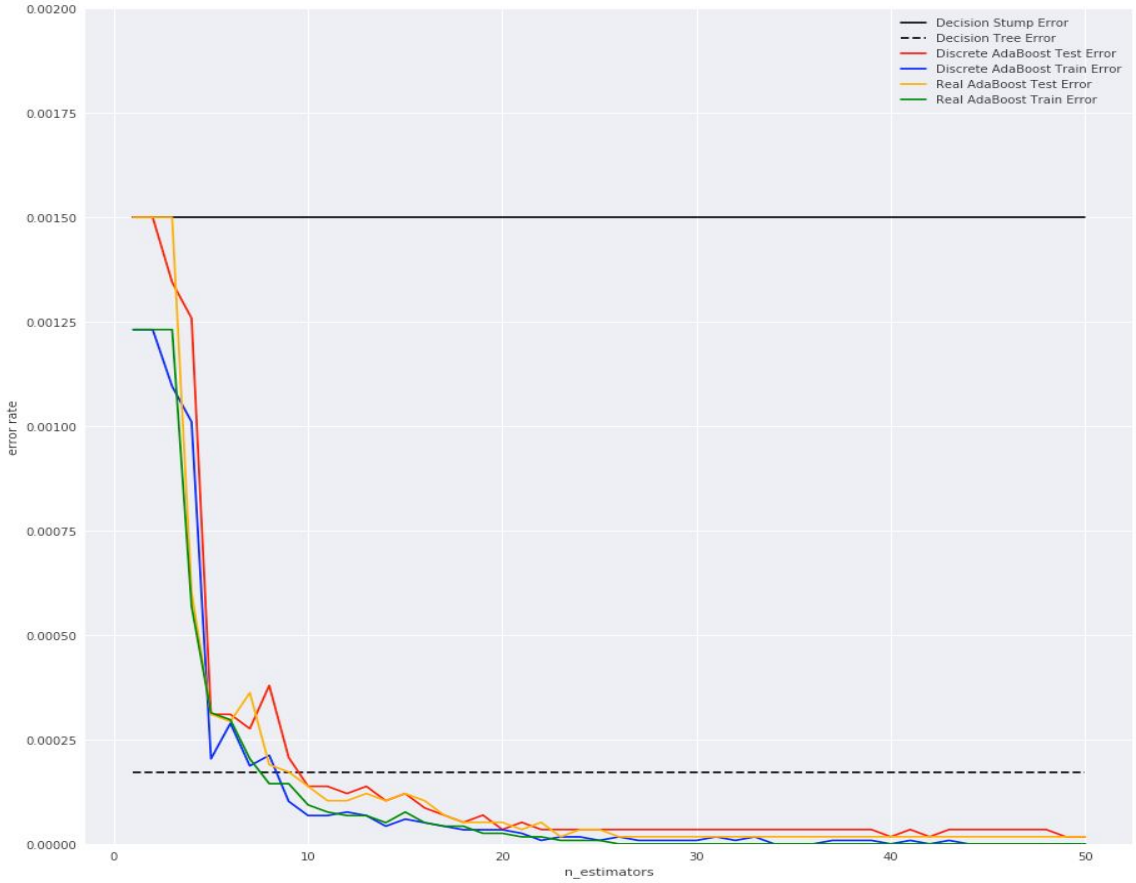
A simple architecture was tested with one hidden layer of 4 nodes. This model performed quite well, achieving an f1-score of 0.9995. Additional architectures were evaluated that did not significantly improve upon the smaller architecture. A deeper model with three hidden layers of 6, 12, and 6 nodes achieved an f1-score of 0.9994. A wider model with a single hidden layer of 24 notes achieved an f1-score of 0.9993.

Refinement

Because of its performance on the testing set, Adaboost algorithm was chosen for further refinement. A GridSearchCV method was tested with various hyperparameter tunings. Results were actually no better than the sklearn defaults, but a few parameter tunings allowed the prediction time to be reduced without sacrificing accuracy on the testing set.

	Training Time	Prediction Time	F1 Score	Precision	Recall
Gaussian Naive Bayes	0.023073	0.006533	0.99868	0.99939	0.99797
Logistic Regression	0.296882	0.001045	0.99817	1	0.996346
Decision Tree	0.187004	0.002389	0.998985	0.998985	0.998985
Adaboost	3.1456	0.329261	0.999797	1	0.999594
MLP Small	44.1738	0.383028	0.999492	0.999797	0.999188
MLP Deep	44.6797	0.435892	0.999492	0.999797	0.999188
MLP Wide	43.4112	0.394839	0.999594	0.999594	0.999594
Adaboost Tuned		0.202983	0.999797	1	0.999594

The number of estimators was reduced from the default of 50 to a tuned value of 25. The n_estimators parameter controls how many trees are generated before the model stops. The following chart shows the error rate as additional estimators are added. You can visually see that around the 20th iteration, the training and testing errors don't improve significantly with additional iterations.



The tuned model's max_features value set to 3 and the min_samples_split set to 4. Max features in the first model defaulted to the number of features(8) and tuned min_samples_split was set at 4 (up from the default of 2). The max features parameter controls how many features are used to build the trees and the min_samples_split features tell the model how many samples to consider when creating a new decision split. The GridSearchCV resulted in setting all other parameters to the default values in scikit-learn.

IV. Results

Model Evaluation and Validation

The final Adaboost model was tested against a separately generated validation dataset. In addition to the variations generated by the FEBRL package, another type of error was introduced to evaluate the model's robustness. In the real world, one type of error that is often observed is when a person will have a residential address in one system but put a Post Office Box or other mailing address in another system. Using Pandas' sample function and numpy's randint function, Address_1 was randomly replaced with a PO Box # for two percent of the records in the validation dataset.

The tuned Adaboost algorithm was tested to make predictions against the validation data. Results were similar to the original testing data and the model achieved an f1-score of 0.9993. Interestingly, though, the default Adaboost model performed slightly better than the tuned model, achieving an f1-score of 0.9996. The additional source of noise in the data was better handled by the model with more estimators and a smaller min_samples_split.

In the end, the default Adaboost model proved to be the most accurate and most robust. The increased training and prediction time is acceptable, given the modest improvement in accuracy.

The final parameters for the Adaboost model are as follows:

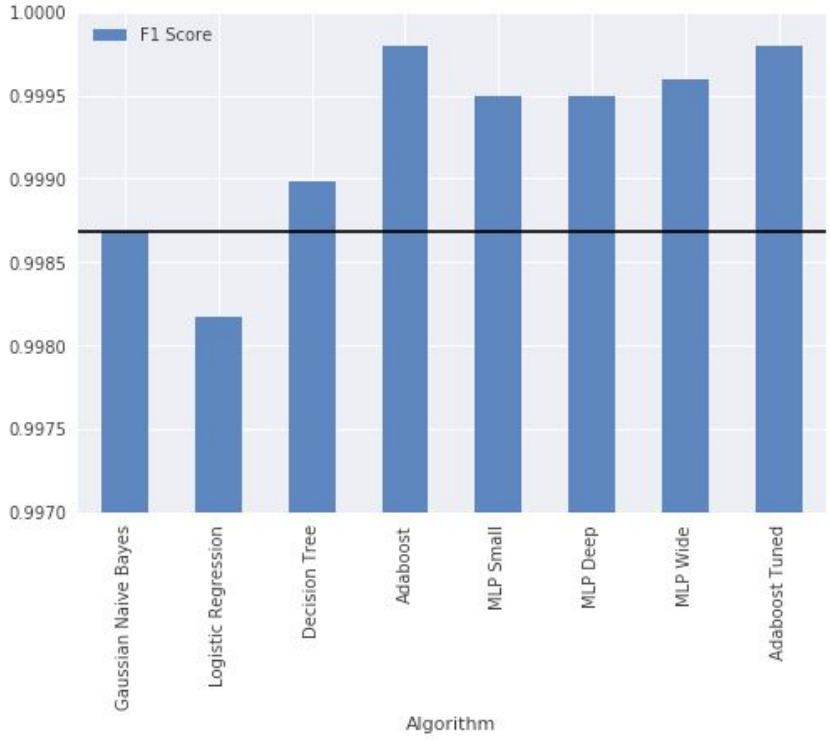
- Base Estimator: Decision Tree Classifier
    - Criterion = 'gini'. The optimal metric for determining the impurity of each node.
    - Max_features = None. This allows the algorithm to uses all features in building trees.
    - Splitter = best. This allows the algorithm to choose the best split as opposed to random splits
- Algorithm: 'SAMME.R'. This algorithm uses the class probability to build future estimators as opposed to 'SAMME' which uses the binary classifications only.
- Learning Rate. The default value of 1 gives acceptable results. The learning rate controls how aggressively the model trains on the residual errors. An error rate that is too high can cause the model to quickly fit (and overfit) the data in fewer iterations. Reducing this value can slow the model's learning and prevent overfitting, but in this case, this does not appear to be necessary.
- N_estimators: 50. The winning Adaboost model produced 50 estimators.

Justification

The final Adaboost model improved performance on the validation data from an f1-score of 0.9989 for the benchmark Gaussian Naive Bayes model to an f1-score of 0.9996. This seems like only a modest improvement, but in the real world this improvement could be a very important gain. Incorrect record linkage could cause lost sales opportunities in industry, incorrect risk categorization in homeland security applications, or incorrectly linked medical records in healthcare applications. The fact that a boosted tree model could perform better than the benchmark model is promising.

V. Conclusion

Free-Form Visualization



With the exception of the Logistic Regression model, all models performed better than the benchmark established by the Gaussian Naive Bayes model. The Adaboost models performed best and were surprisingly effective at classifying matching records. Even after a new type of noise was introduced into the validation data, the boosted models continued to perform well and outperformed the benchmark.

Improvement

This project is limited in scope by the source data available. Because of the previously noted privacy concerns, the availability of actual data is a limitation. Future work will include implementing this model with real-world proprietary data.

Also, future improvements will be considered in the feature engineering of edit distance metrics. It is possible that similarity metrics other than Damerau-Levenshtein could be more useful and/or efficient in preparing features for the algorithms. Cosine similarity may also be useful in engineering additional text comparison features.

Reflection

The task of record linkage based upon noisy demographic data is an important and challenging field of study. In a wide variety of applications, it is necessary to quickly and accurately link records from different sources based upon imperfect similarities. The Adaboost model proposed by this project shows promise as a technique for record linkage.

For this project:

1. Synthetic Record linkage data was generated using the FEBRL package
2. Data was “Blocked” using TF-IDF
3. Text similarities were quantified using the Damerau-Levenshtein distance metric
4. Multiple models were trained to identify a promising approach
5. The Adaboost model was tuned through a process of GridSearch
6. The model was tested against an unseen validation set with slightly modified noise characteristics.

The first challenge to overcome is to limit the scope of comparisons. Traditional blocking techniques proved to be useful but imperfect. This project demonstrates that techniques such as TF-IDF provide better, faster, and more complete results than traditional blocking.

This project also demonstrates the potential for non-linear and boosted models to solve the problem of record linkage. Bayesian and linear models may perform better than other deterministic methods, but tree-based, neural-network, and boosted models show greater promise in accurately identifying accurate matching records.