(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
10 May 2012 (10.05.2012)

PCT

(10) International Publication Number
WO 2012/061162 A1

(51) International Patent Classification:
G06F 17/30 (2006.01) ***G06F 17/00*** (2006.01)
G06F 17/40 (2006.01)

(21) International Application Number:
PCT/US2011/057757

(22) International Filing Date:
25 October 2011 (25.10.2011)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

61/406,264	25 October 2010 (25.10.2010)	US
61/409,908	3 November 2010 (03.11.2010)	US
61/466,608	23 March 2011 (23.03.2011)	US
61/527,926	26 August 2011 (26.08.2011)	US

(71) Applicant *(for all designated States except US)*: **INTELIUS INC.** [US/US]; 500 108th Avenue NE, Bellevue, WA 98004-5586 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **BORTHWICK, Andrew** [US/US]; 121 18th Avenue, Kirkland, WA 98033 (US). **CHEN, Sheng** [CN/US]; 665B Shaler Blvd., Ridgefield, New Jersey 07657 (US).

(74) Agent: FARIS, Robert, W.; Nixon & Vanderhye P.C., 901 North Glebe Road, 11th Floor, Arlington, VA 22203-1808 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:
— *of inventorship (Rule 4.17(iv))*

Published:
— *with international search report (Art. 21(3))*

[Continued on next page]

(54) Title: COST-SENSITIVE ALTERNATING DECISION TREES FOR RECORD LINKAGE

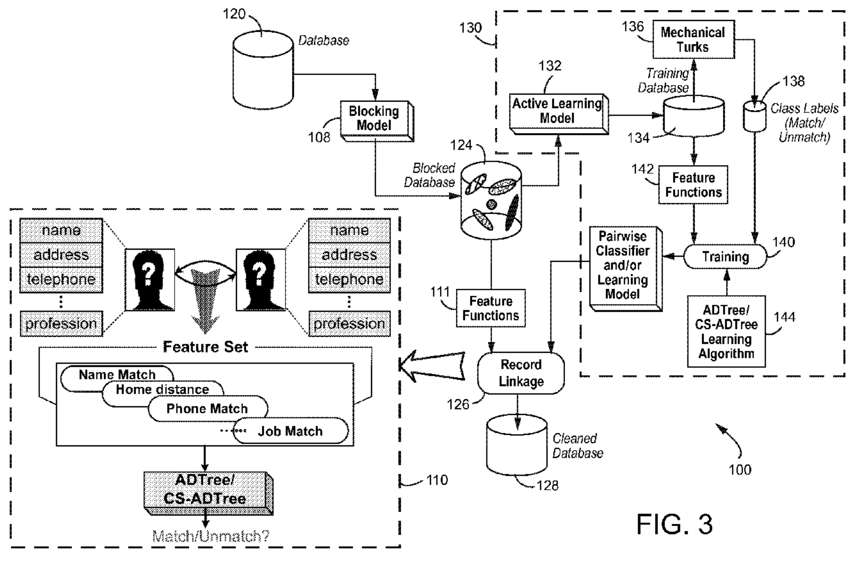


FIG. 3

(57) Abstract: Record Linkage (RL) is the task of identifying two or more records referring to the same entity (e.g., a person, a company, etc.). RL models can be based on Cost Sensitive Alternating Decision Trees (ADTree), an algorithm that uniquely combines boosting and decision trees algorithms to create shorter and easier-to-interpret linking rules. These models can be naturally trained to operate at industrial precision/recall operating points, and the shorter output rules are so clear that it can effectively explain its decisions to non¬ technical users via score aggregation or visualization. The models significantly outperform other baselines on the desired industrial operating points, and the improved understanding of the model's decisions led to faster debugging and feature development cycles.

WO 2012/061162 A1

COST-SENSITIVE ALTERNATING DECISION TREES FOR RECORD LINKAGE

CROSS-REFERENCES TO RELATED APPLICATIONS

[0001] Priority is claimed from US provisional patent applications 61406264 filed on 25-OCT-2010; 61409908 filed on 03-NOV-2010; 61466608 filed on 23-MAR-2011; and 61527926 filed 26-AUG-2011, each incorporated herein by reference for all purposes (specifically including all drawings, tables, equations and expressions) as if expressly set forth.

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

[0002] None

FIELD

[0003] The technology herein relates to databases and record linkage, and more particularly to cost-sensitive alternating decision trees for record linkage.

BACKGROUND AND SUMMARY

[0004] Many consumers want to research the background of childcare providers, coaches, healthcare workers and home contractors. Many of us want to reconnect with family and friends. Business users often have good reason to check and monitor the background of potential employees and ensure they are the best person for the job. Criminal checks can enable consumers to make informed choices about the people they trust by delivering appropriate and accurate public records information. Identity verification can assist businesses and consumers to confirm that someone is who they say they are.

[0005] Given enough time to search, you may be able to discover if your prospective employee has been charged with DUI, if your son's soccer coach has ever been accused of domestic violence, and where your old college roommate or high school classmate is now living. But a significant challenge is to be sure the information you are seeing pertains to the right person or corporation. We are all concerned about detection and prevention of identity theft, America's fastest-growing crime. Many of us once carried our social security cards in our wallets, but the risk of identity theft has cautioned us to now reveal our social security number to no one other than our bank, our employer and our taxing authority on a confidential need-to-know basis. Without a unique national identification number that could serve as an index into a variety of records from many different sources, it is challenging to accurately link such records without making mistakes that could provide incorrect information, damage someone's reputation or destroy confidence in the information source.

[0006] Figure 1 shows an example of the scope of the problem. There are over 300 million people living in the U.S. alone, and billions of records pertaining to all those people. Take the example of trying to find out accurate personal information about Jim Adler, age 68 of Houston Texas. By analyzing available records, it may be possible to find 213 records pertaining to “Jim Adler” but it turns out that those 213 records may pertain to 37 different Jim Adlers living all over the country. It is desirable to link available records to determine which ones pertain to the right Jim Adler as opposed to Jim Adler age 57 of McKinney Texas, or Jim Adler age 32 of Hastings Nebraska, or Jim Adler age 48 of Denver Colorado or any of the other 33 Jim Adlers for whom there are records. It is further desirable to avoid incorrectly linking Jim Adler the First Selectman of Canaan New Hampshire with Jim Adler serving time in the Forrest City Arkansas Federal Correctional Institute.

[0007] Some have spent significant effort to build comprehensive databases that link related records to provide background, criminal, identity and other checks for use by employers and consumers. When consolidating information from multiple data sources, it is often desirable to create an error-free database through locating and merging duplicate records belonging to the same entity. These duplicate records could have many deleterious effects, such as preventing discoveries of important regularities, and erroneously inflating estimates of the number of entities. Unfortunately, this cleaning operation is frequently quite challenging due to the lack of a universal identifier that would safely but uniquely identify each entity.

[0008] The study of quickly and accurately identifying duplicates from one/multiple data source(s) is generally recognized as Record Linkage ("RL"). Synonyms in the database community include record matching, merge-purge, duplicate detection, and reference reconciliation. RL has been successfully applied in census databases, biomedical databases, and web applications such as reference disambiguation of the scholarly digital library CiteSeerX and online comparison shopping.

[0009] One example non-limiting general approach to record linkage is to first estimate the similarity between corresponding fields to reduce or eliminate the confusion brought by typographical errors or abbreviations. A straightforward implementation of similarity function could be based on edit distance, such as the Levenshtein distance. After that, a strategy for combining these similarity estimates across multiple fields between two records is applied to determine whether the two records are a match or not. The strategy could be rule-based, which generally relies on domain knowledge or on generic distance metrics to match records. However, a common practice is to use Machine Learning (ML) techniques, to treat the similarity across multiple fields as a vector of features and "learn" how to map them into a match/unmatch binary decision. ML techniques that have been tried for RL include Support Vector Machines, decision trees, maximum entropy, or composite ML classifiers tied together by boosting or bagging.

[0010] Due to the importance of feature representation, similarity function design is at the core of many record linkage studies. As noted above, perhaps the most straightforward one is the Levenshtein distance which counts the number of *insert*, *remove*, and *replace* operations when mapping string A into B. Considering the unbalanced cost of applying different operations in practice, it is possible to modify the definition of edit distance to explicitly allow for cost customization by designers. In recent years similarity function design is increasingly focused on adaptive methods. Some have proposed similar stochastic models to learn the cost factors of different operations for edit distance. Rooted in the spirit of fuzzy match, some consider the text string at the tuple level and proposes a probabilistic model to retrieve the *K* nearest tuples with respect to an input tuple received in streamed format. Exploiting the similarity relation hidden under a big umbrella of linked pairs, some have iteratively extracted useful information from the pairs to progressively refine a set of similarity functions. Others introduce the similarity functions from probabilistic information retrieval and empirically study their accuracy for record linkage.

[0011] Whatever ingenious methods may be used for similarity functions, it is desirable to integrate all of these field-level similarity judgments into an overall match/no-match decision. Various learning methods have been proposed for this task. Some have proposed stacked SVMs to learn and classify pairs of records into match/unmatch, in which the second layer of SVMs is trained on a vector of similarity values that are output by the first layer of SVMs. Others consider the records in a database as nodes of a graph, and apply a clustering approach to divide the graph into an adaptively determined number of subsets, in which inconsistencies among paired records are expected to be minimized. Some instead consider features of records as nodes of a graph. Matched records would excite links connecting corresponding fields, which could be used to facilitate other record comparisons.

[0012] A well-performing pairwise classifier depends on the representativeness of the record pairs selected for training, which calls for an active learning approach to efficiently pick informative paired records from a data pool. Some have described committee-based active learning approaches for record linkage. Considering the efficiency concern of applying an active learning model on a data pool with quadratically large size, others propose a scalable active learning method that is integrated with *blocking* to alleviate this dilemma.

[0013] Despite its importance in producing accurate estimation of duplicates in databases, insufficient attention has been given to tailoring ML techniques to optimize the performance of industrial RL systems.

[0014] Example illustrative non-limiting embodiments herein provide cost sensitive extensions of the Alternating Decision Tree (ADTree) algorithm to address these and other problems. Cost Sensitive ADTrees (CS-ADTree) improve the ADTree algorithm which is well-suited to handle business requirements to deploy a system with extremely different minimum false-positive and false-negative error rates. One exemplary illustrative method assigns biased misclassification costs for positive class examples and negative class examples.

[0015] Exemplary illustrative non-limiting implementations provide record linkage of databases by ADTree. Considering that a problem of record linkage is that the business costs of misclassifying a matched pair and an unmatched pair can be extremely biased, we further propose CS-ADT which assigns a higher or lower misclassification cost for matched pairs than for non-matched pairs in the process of training ADTree. Experiments show CS-ADTree and ADTree perform extremely well on a clean database and exhibit superior performance on a noisy database compared with alternative ML techniques. We also demonstrate how the run-time representation of ADTree/CS-ADTree can facilitate human understandability of learned knowledge by the classifier and yield a compact and efficient run-time classifier.

[0016] Because ADTrees output a single tree with shorter and easy-to-read rules, the exemplary illustrative non-limiting technologies herein can effectively explain its decisions, even to non-technical users, using simple score aggregation and/or tree visualization. Even for very large models with hundreds of features, score aggregation can be straightforwardly applied to perform feature blame assignment — i.e., consistently calculate the importance of each feature on the final score of any decision. Improved understanding of these models can lead to faster debugging and feature development cycles.

[0017] Other non-limiting advantages features and advantages include:

- Human-understandability
- Confidence measure
- Runtime efficiency from lazy evaluation
- Capture intuitive feature interactions
- Very competitive F-Measure
- Bake preference for precision into algorithm
- Better recall at high levels of precision
- Other.

BRIEF DESCRIPTION OF THE DRAWINGS

[0018] These and other features and advantages will be better and more completely understood by referring to the following detailed description of exemplary non-limiting illustrative embodiments in conjunction with the drawings of which:

[0019] Figure 1 illustrates the record linking problem for personal records;

[0020] Figure 2 shows an example non-limiting overall execution-time data processing system 100;

[0021] Figure 3 shows a further illustrative non-limiting data processing system 100 encompassing both execution-time components and training-time components;

[0022] Figure 4 shows an example non-limiting illustrative “match/unmatch” test using an ADTree/CS-ADTree approach;

[0023] Figure 5 shows example non-limiting illustrative person record linkage features;

[0024] Figure 6 shows an example non-limiting illustrative Record Linkage Development Cycle;

[0025] Figure 7 shows an example illustrative comparison of machine learning and record linkage approaches;

[0026] Figure 8 shows an example alternating decision tree and an example conventional decision tree;

[0027] Figure 9 is an example non-limiting cost matrix for cost sensitive learning;

[0028] Figure 10 is a non-limiting illustrative example of a pair of person profile records;

[0029] Figure 11 is a non-limiting illustrative example of feature values of phone_match;

[0030] Figure 12 is a non-limiting illustrative example of a tiny ADTree for person linkage;

[0031] Figure 13 is a non-limiting illustrative example of a confusion matrix for binary classification;

[0032] Figure 14 is a non-limiting illustrative comparison of average’s for ADTree/CS-ADTree;

[0033] Figure 15 is a non-limiting illustrative example of Precision-Recall curves of ADTree/CS-ADTree;

[0034] Figure 16 is a non-limiting illustrative example of threshold range of ADTree/CS-ADTree with respect to high range;

[0035] Figure 17A-17W are together an extended drawing showing an example CS-ADTree (if name_suffix=differ, we get a strong negative score and don't query additional features) (to assemble this drawing, place Figure 17A on the left, Figure 17B to the right of Figure 17A, Figure 17C to the right of Figure 17B, and so on);

[0036] Figure 17X shows a subset of certain relevant nodes from the CS-ADTree shown collectively in the extended drawing of Figure 17A-17W which illustrates the effect of the *name_suffix* feature;

[0037] Figure 18 is an example partial CS-ADTree. Note how we assign different scores to different birthday distances and query additional features if birthday difference is unknown (infinite);

[0038] Figure 19 is an example non-limiting illustrative set of P-R curves of 3 active learning approaches;

[0039] Figure 20 shows example non-limiting illustrative P-R curves of ADTree and CS-ADTree on an example illustrative evaluation dataset $\mathcal{E}$; *and*

[0040] Figure 21 is an exemplary non-limiting illustrative histogram of f-measure's for ADTree, CS-ADTree, and alternative ML techniques on $\mathcal{E}$.

DETAILED DESCRIPTION

[0041] Figure 2 shows a non-limiting illustrative example overall data processing system 100 providing record linkage using alternating trees and cost-sensitive learning. In the example shown, data is acquired from a variety of different sources including for example address history and real estate tax records, telephone records, criminal and civil court records, licenses, professional records, social graphs and social networking sites. This information can be acquired electronically (e.g., through use of web crawlers or other information harvesting technology), manually (e.g., through visual record inspection, telephone calls, personal visits, interviews, etc.) or both. In one example non-limiting implementation, the data acquisition block 102 may comprise one or more computers connected to the Internet or other electronic network that generates and transmits queries over the network, receives responses and stores the responses in non-transitory storage device(s).

[0042] In the example shown, the data acquired by data acquisition 102 is cleaned, validated and standardized using one or more data exchange computer processors 104 to provide outputs in a standardized data model 106. One exemplary non-limiting implementation uses one or more Hadoop clusters having sufficient numbers of nodes to provide necessary processing resources.

[0043] The standardized data 106 is then provided to a conventional blocking engine 108 that attempts to find groups of records that are similar in some way. The blocked records are then provided a “Machine Learning” (ML) processor 110 that applies alternate trees and cost-sensitivity to accurately link or match records. The resulting record linkages are further processed by a clustering process 112 to generate search indices 114 that are stored in a non-transitory document store 116. In the illustrative implementation, the blocking 108’s job is to avoid recall loss in proposing pairwise judgments whereas the clustering 112’s job is to turn pairwise judgments into correct partitioning of graph into entities. Truth is derived from human judgments on record pairs, whereas precision is based on True Positive/(True Positive + False Positive) and Recall is based on True Positive/(True Positive + False Negative).

[0044] Local and/or remote users can search the document store 116 via web browser based or other displays 118 and associated input devices (e.g., touch screens, keyboards, pointing devices, voice recognition input, smart phones, tablet computers, etc.) to access (e.g., via the Internet) the stored information (e.g., a records check on a son’s soccer coach, a prospective employee, an investment advisor, etc.) on a free, trial or subscription basis.

[0045] The automated steps described above may be performed by computer under control of software instructions stored in non-transitory storage devices such as disk drives, flash memory devices or any other conventional program store(s). Accuracy can be measured on pairwise classification as match/no-match.

[0046] Figure 3 shows a further illustrative example implementation of system 100 encompassing both execution-time components and training-time components. System 100 includes a database 120 that is processed by blocking model 108 to provide a blocked database for further processing by feature functions and record linkage 126 for storage in a cleared database 128. Record linkage 126 in turn is based on a learning model 110 obtained from a machine learning process 130 using an active learning model that operates on the blocked database 124 to develop a training database 134. The training database may use a crowdsourcing methodology which in this illustrative non-limiting example is implemented via Amazon's Mechanical Turk platform, 136, to develop class labels 138 for input into training 140 used to develop the learning model 110. Training 140 may also accept input from additional feature functions 142 and ADTree/CS-ADTree learning algorithm(s) to be described below.

[0047] An output of record linkage 126 is a match/unmatch block 146 that determines whether two records should be linked. Figure 4 shows in more detail how two records containing a variety of fields (e.g., name, address, telephone number, profession, etc.) are applied to a feature set 148 through an ADTree/CS-ADTree process 150 to make a "match/unmatch?" determination as to whether the records should be linked or not.

[0048] Figure 5 shows an example of illustrative records that are candidates for linkage based on an illustrative feature set. The illustrative non-limiting fields shown are “Name”, “Birthday”, Street address”, “City/State” and “Phone number.” Each record contains the name “Robert Jones” who lives in Seattle Washington and was born in March of 1966. Note however that the phone numbers are different (same area code, but different numbers), the street addresses are different and the birthdays don’t match. Is this the same Robert Jones or a different one? Figure 5 thus illustrates different person record linkage features. There can be a number of (e.g., over 40) different features that can be a mix of real-valued, integer, Boolean, and categorical (e.g., phone match could be on area code, area code + exchange, or exact match).

[0049] Figure 6 shows an exemplary illustrative non-limiting record linkage development cycle. In the non-limiting example shown, we identify pairs we are unsure of (200), send pairs to human annotators (“mechanical turk”) (202), have the annotators label the pairs as match/no-match (204), write features to cover identified edge cases (206) and retrain the model with new data and features (208). In addition to the feature set, a focus in our exemplary illustrative non-limiting implementation was to choose an optimal machine learning algorithm for record linkage.

Example Machine Learning Algorithms

[0050] Figure 7 shows an example non-limiting comparison of different machine learning and record linkage techniques, and Figure 8 shows an example alternating decision tree and conventional decision tree. ADTrees were described by Freund and Mason, The Alternating Decision Tree Learning Algorithm, *Proceedings of the 16th International Conference on Machine Learning*, 124-133 (1999). They can be characterized as "Decision Trees meet Boosting". A standard boosting algorithm is used, but all features learned are of a certain form. There are two kinds of nodes in illustrative implementations: splitter nodes and prediction nodes. This does not provide binary splits like with a DT – one can have multiple splitter nodes (multiple questions) hanging off of each prediction node. Note that decision trees and alternating decision trees provide good human interpretability, capture feature interactions and handle diverse feature types. However, they do not provide cost sensitivity. On the other hand, cost-sensitive alternating decision trees provide all of these advantages as well as cost-sensitivity.

[0051] Cost-Sensitive Learning

[0052] One common motivation for cost sensitive learning is the scenario of training a classifier on a data set which contains a significantly unequal distribution among classes. This sort of problem usually consists of relative imbalances and absolute imbalances. Absolute imbalances arise in data sets where minority class examples are definitely scarce and under-represented, whereas relative imbalances are indicative of data sets in which minority examples are well represented but remain severely outnumbered by majority class examples.

[0053] A second motivation for cost sensitive learning is when there are different "business costs" (real-world consequences) between false positive and false negative errors. Cost sensitive learning for the binary classification problem can be best illustrated by a cost matrix adhering to data that consists of two classes: positive class P and negative class N. For the sake of convenience, in the rest of this paper we refer to examples/instances belonging to the positive and negative classes as positive examples/instances and negative examples/instances, respectively. In the person record linkage context, a positive example is a pair of records which represent the same person. A negative example is when the pair of records represent two different people.

[0054] The cost matrix in Fig. 9 demonstrates the cost of four different scenarios of classifying an instance into either positive class P or negative class N. The correct classifications reside on the diagonal line of the cost matrix and have a cost of 0, i.e., C(P, P)=C(N, N)=0. Traditional reports on Record Linkage work often assign equal costs to misclassifying a positive instance into a negative instance and misclassifying a negative instance into a positive instance, i.e., C(P, N)=C(N, P). This works perfectly fine when the positive class and negative class are of equal interest. Nevertheless, this is rarely true in the real business world. For instance, failure to identify a credit fraud case (a “false negative” error) would bring a much higher expense than the reverse case. On the other hand, in a RL industrial setting, there are also many applications which are more averse to false positives than false negatives. For instance, one illustrative industrial database may drive an application which displays the history of addresses a person had resided at and jobs they had held, among other things. In this business, a false negative would generally mean that we would fail to list a true address or true job title that a person had had. However, it is considered far worse to make a false positive error whereby we would say that a person had lived at an address or held a job that pertained to someone else. Similar business tradeoffs are known where a false positive in record linkage on a children's immunization database can lead to a child not being vaccinated for a dangerous disease, whereas a false negative leads to a child receiving a redundant, but harmless, vaccination.

[0055] In other terms, cost sensitivity asks "Do you care more about precision or recall error?" For an airline, a recall error is worse – the airline must identify all possible terrorists who are getting on the plane. For a search engine in contrast, a precision error is worse than a recall error. It is undesirable to falsely mix two people's profiles. Therefore, for search engines in our non-limiting examples, recall at very high precision (99%+) is what we care about. It may be difficult to obtain very high precision without cost sensitivity.

[0056] A classifier created for record linkage has the task of classifying a pair of records in a database into match or unmatch. Although academic systems typically target f-measure (the harmonic mean of precision and recall), which weights false positive and false negative errors equally, as discussed previously, some industrial applications typically consider false positives to be much more expensive than false negatives (albeit with exceptions, such as with the airline example cited earlier). Hence industrial systems targeting high precision will frequently seek to maximize recall while ensuring that precision is at least π for some $\pi \in [0,1]$. As an example, in one example non-limiting database, there may be a requirement of $\pi \geq 0.985$, where $\pi = 0.996$ is a typical value. This implies that misclassifying no-match cases, and thus making a false positive error, should be much more expensive than misclassifying match cases (yielding a false negative) in terms of cost, which gives rise to the study of applying cost sensitive learning to tackle record linkage problems.

Illustrative Feature Design

[0057] To build a strong record linkage system, coming up with a proper feature representation is highly useful. To accomplish that goal, most existing work concentrates on designing similarity functions estimating the similarity levels between corresponding fields in a pair of records. Although this kind of method can efficiently capture some of the semantic similarity between a pair of records despite various levels of distortions of textual strings, it leaves out some crucial signals. For instance, a rare name match in records should be considered more likely to be a duplicate than a common name match (e.g. two “Hannah Philomene’s” are more likely to the same than two “John Smith’s”). More generally, our experience is that achieving the highest levels of accuracy requires the design of at least dozens of heterogenous features so that the system can attempt to exploit every possible signal that is detectable by the feature designer. Given these requirements, working with a machine-learning algorithm which produces a human-understandable run-time model greatly facilitates feature development and debugging.

[0058] An illustration of one illustrative feature design starts with the example of a pair of person profile records shown in Fig. 10. Although the pair only differs by one character across all fields, it can be clearly identified as a non-match by anyone familiar with American naming conventions, since it is apparent that these two individuals are likely father and son. However, this example would be predicted as a match by many record linkage platforms because of the degree of textual similarity. Our first feature is thus name_suffix, which is a categorical feature with three possible values: match if the name suffixes match (i.e. John Smith Jr. vs. John Smith Jr.), differ if the name suffixes differ, and none if this feature does not apply (e.g. if one or both records do not contain a name suffix).

[0059] The second feature is related to name frequency, which is global_name_frequency. It is a numeric feature characterizing frequency of the name in the population. Note that if the names do not match, we consider the feature value in that case to be positive infinite. This feature is negatively correlated to the record linkage decision, i.e., a large value of global_name_frequency would decrease the likelihood that the two records match.

[0060] The third feature is telephone number match, i.e., phone _match, which is a categorical feature. US phone numbers can be segmented into three parts. Records matching on different parts or different conjunctions of parts of phone number should be considered duplicates with varied likelihood. Besides a no_match where phone numbers are different in all three parts, Fig. 11 illustrates the other four feature values phone_match can take. The advantage of this feature is that in lieu of determining the cost of variation in different part of phone numbers either manually or adaptively, we directly push the different match scenarios into the ML algorithm as feature values to make the algorithm figure out the appropriate strategy for “ranking" different match cases.

[0061] A partial list of other features used in this system include the following:

Feature Name	Explanation
street_address_match	returns “true" if street name and house number match
birthday_difference	returns the number of days separating the birthdays
regional_population	returns the population of the region that the two records have in common or positive infinity if not in the same region. (rationale: two records known to share a New York City address are less likely to match than two addresses sharing a Topeka, Kansas address)
name_matches	returns number of matching names (rationale: particularly useful for criminal matching, where criminals may match on multiple aliases)

Table 1: Other Non-Limiting Features

[0062] As can be seen from these feature examples, the construction of a high-precision, industrial-strength record linkage system requires a lengthy list of complex features which are the result of extensive feature engineering. It is thus useful that the illustrative machine learning algorithm supports the feature development process.

Exemplary Illustrative Non-Limiting ML Algorithms

Alternating Decision Trees (ADTrees)

[0063] The ADTree algorithm is a combination in a peculiar way of the decision tree and boosting algorithms. There are two kinds of nodes in an ADTree. One is the splitter node which specifies the condition that should be tested for the instance. The other is the prediction node, which assigns a real-valued score to instances satisfying conditions at different points. An ADTree can be split multiple times at any point, i.e., it can attach more than one splitter node at any single prediction node. This is different from a decision tree since (1) generally, a decision tree can only be split once at each point, and (2) the split can only be performed at the bottom of the tree, i.e., the prediction leaves, in the progress of tree generation. Upon determining the class label of an instance, ADTree sums up the score of the prediction nodes of all paths on which the condition specified by the splitter nodes are all satisfied by the instance. The sign of the summation is the class label of the instance. Note that a conventional decision tree also decides the class label of an instance by going through a path in the tree hierarchy. The difference is that there is just one path and only the prediction leaf at the bottom determines the class label of the instance.

[0064] A toy example of a person-linking ADTree, shown in Fig. 12, was built from a database of over 50,000 labeled examples consisting of a mixture of criminal and non-criminal person records. There are five features illustrated in this model which were described above. Let's suppose the model is considering the pair of records shown in Figure 5.

[0065] For this example, we first note that the root prediction node starts us off with a score of 0.140. The relatively small positive score indicates a slight predominance of positive ("match") examples in the training corpus. We then see that Feature 1, street_address_match, returns "none" (yes:-0.291), so we test Feature 2, which returns 3.3 million, since that is the population of the Seattle Metropolitan Area. Since feature 2 returned "yes" (+0.399), we now query features 4 and 6. Feature 4 returns 1000, the frequency of "Robert Jones" and thus answers "no" (-0.667), while Feature 6 tests the "regional population" feature again and decides "no" (-0.206), this time since the population is not less than 75,917.5. Feature 3 returns 14, which is less than 417.5, giving a "yes" decision (1.325). Feature 5 returns 1 since there is only one matching name, so the decision is also "yes" (-0.101).

[0066] Summing these values, the class label is thus determined through calculating $0.140-0.291+0.399-0.667-0.206+1.325-0.101=0.599$, i.e. a relatively low-confidence match. Note also that we can determine the amount of "blame" to assign to each feature by summing the prediction nodes of each feature. Consequently, we can see that "regional_population" contributes 0.399 - 0.206 = 0.193 to the final score, which is very helpful in feature engineering.

[0067] Finally, note the following two benefits of ADTrees. Firstly, if Feature 1 had returned "no" because street_address_match did not return "none", we would not have tested features 2, 4, and 6, thus reaping a major gain in run-time performance. Secondly, note how the ADTree seamlessly mixed real-valued features with a boolean feature (street_address_match), a property which simplifies development and facilitates the incorporation of heterogeneous features in a record linkage model.

[0068] As a second example, consider the name_suffix feature described above. Due to the importance of this feature, ADTree generally puts it somewhere near the root node. The example in Fig. 5 illustrates this. "name_suffix=differ" is a child of the root node and is the eighth feature chosen. If the name suffix differs, we decrease the score (making a match less likely) by −2.89 and don't query any additional features. On the other hand, if the suffixes don't differ, the score is basically unaffected (adding 0.039 is trivial in the context of the other, much larger values) and we query many other features.

Note that one additional feature is whether “name_suffix=match", which gets a strong positive value of 1.032 if true.

	Record 1	Record 2	Comment
Name	Robert Jones	Robert Jones	Name Frequency = 1000
Birthday	3/1/1966	3/15/1966	Birthday difference = 14
Address	121 Walnut St.	234 Chestnut St.	
City/State	Seattle, WA	Seattle, WA	Population of Seattle region = c. 3.3 million

Table 2: An example comparison of two records

Cost Sensitive Alternating Decision Tree

[0069] We formulate a novel implementation of ADTree equipped with cost sensitive learning, called *Cost Sensitive Alternating Decision Tree* (CS-ADTree). This example non-limiting adaptation of ADTree provides a framework of cost sensitive learning and also applies a cost sensitive ML technique to the problem of record linkage.

[0070] Cost sensitive learning additionally assigns a cost factor $c_i \in (0, \infty)$ to each example x_i, y_i in the database to quantify the cost of misclassifying x_i into a class label other than y_i. In the standard boosting algorithm, the weight distribution over the data space is revised in an iterative manner to reduce the total error rate. Cost sensitive learning operates in a similar manner, but it operates over a space of examples in which the weight distribution has been updated in a biased manner towards examples with higher costs. According to the “translation theorem", the classifier generated in this way will be conceptually the same as the one that explicitly seeks to reduce the accumulated misclassification cost over the data space. We will examine three different methods of biasing the training data to account for the business preference for false negatives vs. false positives. The baseline weight update rule of ADTree is,

$$w_i^{t+1} \leftarrow w_i^t \cdot e^{-r_t(x_i)y_i}$$

[0071] In the illustrative methodsherein, AdaC2, the weight update rule for ADTree is modified as,

$$w_i^{t+1} \leftarrow c(i) \cdot w_i^t \cdot e^{-r_t(x_i)y_i}$$

where

$$c(i) = c_+ \cdot \mathrm{I}(y_i = +1) + c_- \cdot \mathrm{I}(y_i = -1)$$

[0072] The intuition here is straightforward. We hope weights of the examples with higher costs are increased *faster* than those with lower costs. According to Lemma 1 (listed below), the learning focus of ADTree will be biased towards examples with higher costs.

[0073]

Lemma 1. The creation of a splitter node is approximately optimized if:
$Z = 2(\sqrt{\mathcal{W}_+(d_1 \cap d_2)\mathcal{W}_-(d_1 \cap d_2)} + \sqrt{\mathcal{W}_+(d_1 \cap \neg d_2)\mathcal{W}_-(d_1 \cap \neg d_2)}) + \mathcal{W}(\neg d_1)$ is minimized subject to $d_1 \in \mathcal{P}$ and $d_2 \in \mathcal{D}$.

PROOF:
$$Z = 2(\sqrt{\mathcal{W}_+(d_1 \cap d_2)\mathcal{W}_-(d_1 \cap d_2)} + \sqrt{\mathcal{W}_+(d_1 \cap \neg d_2)\mathcal{W}_-(d_1 \cap \neg d_2)}) + \mathcal{W}(\mathrm{T}) - \mathcal{W}(d_1)$$

Since $\mathcal{W}(\mathrm{T})$ is the same for pair-wise combination of d_1 and d_2, it can be omitted from the Z equation.
$\rightarrow$
$$Z = 2(\sqrt{\mathcal{W}_+(d_1 \cap d_2)\mathcal{W}_-(d_1 \cap d_2)} + \sqrt{\mathcal{W}_+(d_1 \cap \neg d_2)\mathcal{W}_-(d_1 \cap \neg d_2)}) - (\mathcal{W}_+(d_1 \cap d_2) + \mathcal{W}_+(d_1 \cap \neg d_2)) + \mathcal{W}_-(d_1 \cap d_2) + \mathcal{W}_-(d_1 \cap \neg d_2)$$
$\rightarrow$
$$Z = -[(\sqrt{\mathcal{W}_+(d_1 \cap d_2)} - \sqrt{\mathcal{W}_-(d_1 \cap d_2)})^2 + (\sqrt{\mathcal{W}_+(d_1 \cap \neg d_2)} - \sqrt{\mathcal{W}_-(d_1 \cap \neg d_2)})^2]$$

[0074]

[0075] Since $\sqrt{\mathcal{W}_+(d_1 \cap d_2)}$, $\sqrt{\mathcal{W}_-(d_1 \cap d_2)}$, $\sqrt{\mathcal{W}_+(d_1 \cap \neg d_2)}$, $\sqrt{\mathcal{W}_-(d_1 \cap \neg d_2)}$ are all greater than 0, *Z* is minimized when the following two conditions are *both* satisfied:

[0076] 1. Absolute difference between $\sqrt{\mathcal{W}_+(d_1 \cap d_2)}$ and $\sqrt{\mathcal{W}_-(d_1 \cap d_2)}$ is maximized.

[0077] 2. Absolute difference between $\sqrt{\mathcal{W}_+(d_1 \cap \neg d_2)}$ and $\sqrt{\mathcal{W}_-(d_1 \cap \neg d_2)}$ is maximized.

[0078] This means examples reaching conditions $d_1 \cap d_2$ or $d_1 \cap \neg d_2$ are largely from just one class. In an extreme case, if all weights are exactly the same, like in first iterative round, and the number of examples reaching $d_1 \cap d_2$ or $d_1 \cap \neg d_2$ is fixed, minimizing Z would result in a perfect split. Q.E.D.

[0079] *Corollary 1.* Precondition d_1 and condition d_2 that *maximize* the Z create a suboptimal split.

[0080] PROOF:

[0081] Following Lemma 1, Z can be maximized when the following two conditions are *both* satisfied:

1. $\sqrt{W_+(d_1 \cap d_2)} = \sqrt{W_-(d_1 \cap d_2)}$
2. $\sqrt{W_+(d_1 \cap \neg d_2)} = \sqrt{W_-(d_1 \cap \neg d_2)}$

[0082] This means the number of positive examples and negative examples reaching conditions $d_1 \cap d_2$ or $d_1 \cap \neg d_2$ should be close to each other to get Z to approach 0. An observer at this point can conclude little knowledge regarding the class label information of the instance reaching this point, which means the split is suboptimal. In an extreme case, when weights of all examples are the same, like in first iterative round, the number of positive and negative examples reaching $d_1 \cap d_2$ or $d_1 \cap \neg d_2$ are exactly matched. The split concludes *no* knowledge and thus is a worst one. Q.E.D.

[0083] Given the modified weight update rule, the original equation for calculating scores α_1 and α_2 at each iterative round no longer guarantees the error rate could be decreased fastest. Inspired by the induction of optimized α for AdaC2, α_1 and α_2 can be modified to $\alpha_1 = \frac{1}{2} \ln \frac{c_+ W_+ (d_1 \cap d_2)}{c_- W_- (d_1 \cap d_2)}$ and

$$\alpha_2 = \frac{1}{2} \ln \frac{c_+ W_+ (d_1 \cap \neg d_2)}{c_- W_- (d_1 \cap \neg d_2)}$$

[0084] Now that we have modified the weight update rule and equations for calculating prediction scores α_1 and α_2, we formulate CS-ADTree in Algorithm 2 below.

Algorithm 2 Cost Sensitive Alternating Decision Tree

```
Inputs:
 1: database $\mathcal{S} = \{(\boldsymbol{x}_1, y_1, c_1), \ldots, (\boldsymbol{x}_n, y_n, c_n)\}$, where $\boldsymbol{x}_i \in \mathcal{X}$, $y_i \in$
    $\{-1, +1\}$, and $c(i) = c_+ \cdot \mathrm{I}(y_i = +1) + c_- \cdot \mathrm{I}(y_i = -1)$.
 2: weights $\boldsymbol{W} = \{w_1^0, \ldots, w_n^0\}$, where $w_i^0 = 1$.
    /* uniform distribution initially */
 3: $\mathcal{D} \leftarrow \{$all possible conditions$\}$.
 4: $N \leftarrow$ Number of iterative rounds.
 5: $\eta \leftarrow$ smooth factor.

Procedure:
 6: $r_0 \leftarrow \mathrm{T}$ with score $\frac{1}{2}\ln\frac{c_+ \cdot \mathcal{W}_+(\mathrm{T}) + \eta}{c_- \cdot \mathcal{W}_-(\mathrm{T}) + \eta}$
    /* T: precondition is true for all examples. */
 7: $\mathcal{P}_0 \leftarrow r_0$ /* precondition set */
 8: for $t : 1 \rightarrow N$ do
 9:     $d_1, d_2 = \underset{d_1, d_2}{\mathrm{argmin}}(\mathrm{Z})$
        s.t. $\mathrm{Z} = [2(\sqrt{\mathcal{W}_+(d_1 \cap d_2)\mathcal{W}_-(d_1 \cap d_2)}$
        $+ \sqrt{\mathcal{W}_+(d_1 \cap \neg d_2)\mathcal{W}_-(d_1 \cap \neg d_2)}) + \mathcal{W}(\neg d_1)]$
        $d_1 \in \mathcal{P}_t,\ d_2 \in \mathcal{D}$
10:     $\alpha_1 = \frac{1}{2}\ln\frac{c_+ \cdot \mathcal{W}_+(d_1 \cap d_2) + \eta}{c_- \cdot \mathcal{W}_-(d_1 \cap d_2) + \eta} \rightarrow d_1 \cap d_2$
        /* $\alpha_1$ is the score associated with $d_1 \cap d_2$ */
11:     $\alpha_2 = \frac{1}{2}\ln\frac{c_+ \cdot \mathcal{W}_+(d_1 \cap \neg d_2) + \eta}{c_- \cdot \mathcal{W}_-(d_1 \cap \neg d_2) + \eta} \rightarrow d_1 \cap \neg d_2$
        /* $\alpha_2$ is the score associated with $d_1 \cap \neg d_2$ */
12:     $r_t \leftarrow d_1 \cap d_2 \bigoplus d_1 \cap \neg d_2$
        /* $r_t(\boldsymbol{x})$ is a new splitter node with two associated prediction
        nodes. */
13:     $\mathcal{P}_{t+1} \leftarrow \{\mathcal{P}_t, r_t\}$
14:     $w_i^{t+1} \leftarrow c(i) \cdot w_i^t \cdot e^{-r_t(\boldsymbol{x}_i)y_i}$
        /* update example weights */
15: end for
16: return Classifier for unseen unlabeled instances:
    $\mathcal{H}(\boldsymbol{x}) = \mathrm{sgn}(\sum_{t=0}^{N} r_t(\boldsymbol{x}))$
```

Adaptive Boosting

[0085] Since ADTree shares many similarity with boosting, we would like to firstly introduce Ada.Boost, the most popular boosting algorithm, in brief for better understandability of AD Tree. Algorithm I gives the procedure of applying Ada.Boost to derive the classifier H(x) from database S. A weight distribution W is maintained to sample database S into training data set S_t at each round. It is initially

a uniform distribution, which means all examples are equally likely to be selected into the first sampled data set S_1. At each round $t_{1,}$ the weak hypothesis h_t i.e., trained classifier, is generated by training the base learner L, on the sampled data set S_t. Then, the weighted error ε_t is calculated through summing the weights of the instances in S that are misclassified by h_t. Provided the weak classifier h_t, performs at least slightly better than a random guess on classifying *S*, i.e., $\varepsilon_t > 0.5$, it is guaranteed that $\alpha_t > 0$. Based on this, the weight distribution W is updated for the next iterative round by decreasing the weights of the correctly classified examples and increasing the weights of the misclassified examples both with a factor of ε^{α_t}. Since hard-to-learn examples are prone to be misclassified, their weights are therefore effectively increased with the progress of iterative weights updating. In other words. the proportion of difficult examples in the training data set for creating the weak hypothesis is guaranteed to be increasing. After AdaBoost exhausts N iterative rounds, it outputs the final classifier which is basically a set of N generated hypotheses. The prediction of final hypothesis on an unlabeled instance is a majority voting of the classes predicted by the individual hypotheses weighted by α_t. α_t is monotonically increased with the decrease of ε_t, therefore it is also considered as a *margin* of the hypothesis h_t and can be used to weigh h_t in voting for classifying
instances $x \in \mathcal{X}$.

[0086] It is known that AdaBoost can achieve arbitrary low error rate on training data set as long as enough iterative rounds can be guaranteed. Furthermore it was empirically discovered that AdaBoost has no issue of overfitting to the training data set even with an extremely large number of iterative rounds. Schapire et al, *Boosting the margin: A new explanation for the effectiveness of voting methods*, The Annals of Statistics, 26(5):1651 1685, 1998) later introduced the margin theory to explain this amazing yet a bit counter-intuitive property of AdaBoost. From the practical point of view, the beauty of the illustrative non-limiting AdaBoost technique is that the classification performance of a weak classifier can be significantly improved by creating and combining an ensemble set of such weak classifiers through the procedure of AdaBoost. Boosting a decision tree can yield very good classifiers. A problem is, however, the classification rules concluded by boosting decision trees through AdaBoost may lack the intuitive interpretability by human beings, which is a most desirable but non-limiting property of decision tree learning algorithms.

Alternating decision tree

[0087] *ADTree* is a new combination of decision tree with boosting. It inherits the learning capability of boosting, and is easy and intuitive to be interpreted for its structural representation of base classification rules in a visualized fashion similar to decision tree. There are two kinds of nodes in ADTree. One is the splitter node which specifies the condition that should be tested for the instance. The other is the prediction node which assigns a real-valued *score* to instances satisfying conditions at different points. An ADTree can be split multiple times at any part, i.e attach more than one splitter node at any single prediction node. This is different from decision tree since firstly, decision tree can only be split at most once at each part and secondly, the split can be only performed at bottom of the tree, i.e., the prediction leaves. Upon determining the class label of an instance, the illustrative ADTree sums up the score of the prediction nodes of *all* paths on which the condition specified by the splitter nodes are all satisfied by the instance. The sign of the summation is the class label of the instance. Note that a conventional decision tree also decides the class label of an instance by going through a path in the tree's hierarchy. The difference is that there is just one path and the only the prediction leaf at the bottom determines the class label of the instance.

Algorithm 1 AdaBoost

```
Inputs:
 1: database $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, where $\boldsymbol{x}_i \in \mathcal{X}$,
    $y_i \in \{-1, +1\}$.
 2: weights $\boldsymbol{W} = \{w_1^0, \ldots, w_n^0\}$, where $w_i^0 = \frac{1}{n}$.
    ▷ uniform distribution initially
 3: Base Learner $\mathcal{L}$.
 4: $N \leftarrow$ Number of iterative rounds.
Procedure:
 5: for $t : 1 \rightarrow N$ do
 6:     $S_t \leftarrow S \times \boldsymbol{W}$ ▷ sample from $S$ using $\boldsymbol{W}$
 7:     $h_t = \mathcal{L}(S_t)$, s.t. $h_t : \mathcal{X} \rightarrow \{-1, +1\}$.
        ▷ training hypothesis $h_t$ using $S_t$
 8:     $\epsilon_t = \sum_{i: h_t(\boldsymbol{x}_i) \neq y_i} w_i^t$
        ▷ weighted error of $h_t$ on classifying $S$
 9:     $\alpha_t = \frac{1}{2} \ln(\frac{\epsilon_t}{1 - \epsilon_t})$
10:     $w_i^{t+1} = w_i^t \cdot \frac{\exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))}{Z_t}$, s.t. $\sum_{i=1}^{n} w_i^{t+1} = 1$
11: end for
12: return classifier for unseen unlabeled instances:
    $\mathcal{H}(\boldsymbol{x}) = \mathrm{sgn}(\sum_{t=1}^{N} \alpha_t h_t(\boldsymbol{x}))$
```

[0088] The cost factor $c(i)$ can also be put in other spots of the weight update equation. For instance, it can be put inside the exponential term $e^{-r_t(x_i)y_i}$. When example x is misclassified by r_t, we have $\mathrm{sgn}-r_t(x_i)y_i>0$. A high cost inside the exponential term would thus increase the weight of the example in exponential order. This is the idea of illustrative AdaC1. Its weight update rule and prediction score equations of AdaC1 can be adapted for ADTree as,

$$w_i^{t+1} \leftarrow w_i^t \cdot e^{-c(i)\cdot r_t(\boldsymbol{x}_i) y_i}$$

$$\alpha_1 = \frac{1}{2} \ln \frac{\mathcal{W}(d_1 \cap d_2) + c_+ \cdot \mathcal{W}_+(d_1 \cap d_2) - c_- \cdot \mathcal{W}_-(d_1 \cap d_2)}{\mathcal{W}(d_1 \cap d_2) - c_+ \cdot \mathcal{W}_+(d_1 \cap d_2) + c_- \cdot \mathcal{W}_-(d_1 \cap d_2)}$$

$$\alpha_2 = \frac{1}{2} \ln \frac{\mathcal{W}(d_1 \cap \neg d_2) + c_+ \cdot \mathcal{W}_+(d_1 \cap \neg d_2) - c_- \cdot \mathcal{W}_-(d_1 \cap \neg d_2)}{\mathcal{W}(d_1 \cap \neg d_2) - c_+ \cdot \mathcal{W}_+(d_1 \cap \neg d_2) + c_- \cdot \mathcal{W}_-(d_1 \cap \neg d_2)}$$

[0089] The cost factor can also be put both inside and outside the exponential term in the weight update equation, which gives rise to illustrative AdaC3. Its weight update rule and prediction score equations can be adapted for ADTree as,

$$w_i^{t+1} \leftarrow c(i) \cdot w_i^t \cdot e^{-c(i)\cdot r_t(\boldsymbol{x}_i) y_i}$$

$$\alpha_1 = \frac{1}{2} \times \ln \frac{c_+ \cdot \mathcal{W}_+(d_1 \cap d_2) + c_- \cdot \mathcal{W}_-(d_1 \cap d_2) + c_+^2 \cdot \mathcal{W}_+(d_1 \cap d_2) - c_-^2 \cdot \mathcal{W}_-(d_1 \cap d_2)}{c_+ \cdot \mathcal{W}_+(d_1 \cap d_2) + c_- \cdot \mathcal{W}_-(d_1 \cap d_2) - c_+^2 \cdot \mathcal{W}_+(d_1 \cap d_2) + c_-^2 \cdot \mathcal{W}_-(d_1 \cap d_2)}$$

$$\alpha_2 = \frac{1}{2} \times \ln \frac{c_+ \cdot \mathcal{W}_+(d_1 \cap \neg d_2) + c_- \cdot \mathcal{W}_-(d_1 \cap \neg d_2) + c_+^2 \cdot \mathcal{W}_+(d_1 \cap \neg d_2) - c_-^2 \cdot \mathcal{W}_-(d_1 \cap \neg d_2)}{c_+ \cdot \mathcal{W}_+(d_1 \cap \neg d_2) + c_- \cdot \mathcal{W}_-(d_1 \cap \neg d_2) - c_+^2 \cdot \mathcal{W}_+(d_1 \cap \neg d_2) + c_-^2 \cdot \mathcal{W}_-(d_1 \cap \neg d_2)}$$

[0090] Despite the fact that cost sensitive learning can manifest itself in different forms in boosting, we chose to create CS-ADTree in one illustrative non-limiting implementation by integrating AdaC2 into ADTree's training procedure since the weight updating rule of AdaC2 weighs each example by its associated cost item directly, which naturally fits the algorithm into the realm of the translation theorem. In addition, our preliminary empirical studies showed that AdaC2 performed the best across a number of benchmarks.

Example Non-Limiting Evaluation Parameters

[0091] Most of the time the metrics for evaluating a classifier's performance on record linkage problem are *precision*, *recall* and their harmonic mean, *f-measure*— metrics obtained by fixing the classifier's decision threshold to zero. A confusion matrix shown in Fig. 13 is always helpful to fully capture what these metrics stand for. A quick impression is that a confusion matrix is very similar to a cost matrix. The difference is that what resides in the matrix is the number of instances satisfying the scenario specified by the row and column indexes. Therefore (True Positive) and (True Negative) are the number of correctly classified positive and negative instances, respectively. *FP* (False Positive) and *FN* (False Negative) are the number of instances *falsely* classified into positive class and negative class, respectively. Considering a matched pair of records as a positive example, and an unmatched pair of records as a negative example, *precision*, *recall*, and *f-measure* for record linkage are defined as,

$$precision = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

$$recall = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

$$f\text{-}measure = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

[0092] From the definitions, *precision* measures the *accuracy* of a classifier's predictions of positive instances, while *recall* tests the *completeness* of a classifier's coverage of real positive instances. It is clear that there is a trade-off between *precision* and *recall*. This can be seen by the explanation that if all instances are frivolously predicted as positive, *recall* is maximized to be 1, whereas *precision* is equal to the proportion of positive examples in the database, which can be very bad in some illustrative circumstances. To this end, *f-measure* is frequently used to measure the overall performance of a classification algorithm when the relative importance of recall and precision are evenly balanced.

Customization of evaluation metrics

[0093] Good performance at a single threshold such as zero may lead to unwanted subjectivity. Recent studies show that the performance of a classifier should be evaluated upon a range of thresholds. An appealing property of ADTree is that instead of merely outputting a hard class label, it gives a score value, $\sum_{t=0}^{N} r_t(x)$, to estimate the confidence level of its decision. While ADTree generally uses the sign function to classify unlabeled instances, other numerical value can also be used to serve as the *threshold*. In this case, the classifier of ADTree can be represented as

$\mathcal{H}(x)=\mathrm{sgn}(\sum_{t=0}^{N}(r_t(x))-d)$, where $d\neq 0$. Since the business priority for the industrial database on which we were working was to keep *precision* above a threshold which was in excess of 99%, *precision* in a high range was of much more interest to us and we thus focused on metrics other than *f-measure*. In this example non-limiting implementation, we start by setting the threshold, *d*, to be large enough to make *precision*=1, and then tune down the threshold to progressively decrease *precision* down to 0.985 in steps of 0.001 to determine the *recall* that we are able to achieve at varying high *precision* levels.

Choice of cost ratio

[0094] Some business applications do have strategies to decide costs for different examples. For instance, the amount of money involved in a transaction can be used to quantify the cost related to it. Nevertheless for many other applications such as record linkage, the only prior knowledge available is often the biased interest towards one class over the other. In industrial applications, this is usually expressed as a requirement that the *precision* be at least π, where in our applications, we generally had $\pi\geq 0.985$ and a more typical requirement for π was in excess of 0.995. Note that two parameters influence the *precision* of the classifier, the cost factor, *C*, and the required *precision*, π. We have found that a reasonable heuristic is to adjust your cost factor *C* so that the threshold, *d*, that yields your desired *precision*, π is close to 0, this generally yields close to optimal *recall* at π. We give an informal theoretical justification for this heuristic below.

Non-Limiting Example

[0095] *Objectives* The following provides experimental results of ADTree and CS-ADTree for comparing records of person profile databases. The objectives here are to 1. demonstrate the effectiveness of ADTree and CS-ADTree on classifying pairs of record into match/unmatch for record linkage; 2. illustrate how the run-time representation of ADTree can enable humans' interpretation of the classifier derived by ADTree; 3. demonstrate the competitiveness of ADTree/CS-ADTree with alternative ML techniques heavily used by most existing record linkage frameworks; and 4. Show how CS-ADTree demonstrates superior performance to ADTree at both very high precision requirements.

[0096] *Non-Limiting Implementation* ADTree, CS-ADTree, boosted decision tree, and decision tree algorithms are all implemented based on the JBoost platform. The SVM algorithm is implemented based on SVM-Light.

Performance of ADTree/CS-ADTree

[0097] In one non-limiting example, we used a database $\mathcal{S}$ with match/unmatch labels assigned by expert internal annotators which had 20,541 pairs of person profile records characterized by more than 42 features including those described above. Since the cost of hiring experts is relative high, we later switched to Amazon's Mechanical Turk (MT) System which provides a significantly cheaper and faster way to collect label data from a broad base of non-expert contributions over the web. A drawback of this method is that the database is likely to become noisy since it is possible that some workers on MT ("Turkers") are purely spammers and some are lacking in the necessary domain knowledge for the task.

[0098] We report average results for a 10-fold cross validation. For these initial experiments, we held the number of boosting iterative rounds, $\mathcal{T}$, fixed at 200 and we used a cost factor of $\mathcal{C}$=4 based on earlier experiments showing these to be optimal values over the precision range [0.985,1].

[0099] With *T* and *C* determined, the record linkage performance of CS-ADTree, ADTree, and CS-ADTree with cost-sensitive learning frameworks of AdaC1 and AdaC3 on our initial person profile database $\mathcal{S}$ are studied. Their average *recall* are given in Fig. 14, which clearly shows that CS-ADTree performs the best across all methods under comparison. Average *recall* can sometimes be misleading when the physical P-R curves of two methods under comparison cross in P-R space. In this case, it would be nonsensical to conclude one method is better than the other based on the area under P-R curve which is conceptually equal to the averaged recall. To that end, we also plot P-R curves of all methods under comparison in Fig 15. One can clearly see that CS-ADTree can consistently contain other methods in terms of P-R curve, which validates the supremacy of CS-ADTree.

[00100] One perspective on how CS-ADTree is able to achieve this superior performance can be seen in Fig. 16. CS-ADTree achieves a given level of *precision* at a much lower threshold than the other three methods. It is particularly instructive to look at CS-ADTree as compared to ADTree. CS-ADTree achieves 0.995 *precision* at a threshold of roughly 0, while ADTree achieves this at about 4. An intuition as to CS-ADTree's superiority is that the model is seeking to push positive examples to have scores above 0 and negative scores below 0. In the case of CS-ADTree, we are able to put our threshold, *d*, at about the same threshold as the model's threshold, whereas with ADTree, the model is not "aware" that the threshold of interest to us is 4. In summary:

- Setting threshold at c. 4.0 in ADTree gives 99.5% precision
- ADTree treats false positives and false negatives equally
- CS-ADTree achieves 99.5% precision at c. 0.0
- CS-ADTree algorithm is optimizing relative to 0.0

[00101] To demonstrate how run-time representation of CS-ADTree can help a practitioner better understand the record linkage problem of his/her database in a human-interpretable way, we present a snapshot of a CS-ADTree generated by our initial labeled database $\mathcal{S}$ in Fig. 17A-W, and also partial snapshot of that tree in Fig. 17X. In Fig. 17A-W (and see also the specific nodes shown in Figure 17X), one can clearly see that the decision node of *name_suffix* resides right below the root node. When *name_suffix=differ*, the corresponding path ends immediately and assigns a large negative score to the instance under consideration, otherwise CS-ADTree would go on to check many other features. This is exactly as predicted by our discussion for the *name_suffix* feature above. Note also that this tree can be very efficiently computed if the features are lazily evaluated. If the name suffix differs, there is no need to compute the values for features 14 - 18, which all have *name_suffix* ≠ "*differ*" as a precondition (in fact, 87 out of the 100 features in this sample CS-ADTree have this as a precondition). More hierarchical levels are involved in the example in Fig. 18. If the value of feature *birthday_difference* for a pair of records is relatively small, (the birthdays are less than 432.5 days apart), CS-ADTree would terminate the corresponding path by just examining the value of *birthday_difference*. This is intuitive because having nearly matching birthdays is a strong "match" indicator. We don't need to ask further questions to reach a decision. Otherwise it asks if *birthday_difference* is greater than 5×10^{19}, i.e., *infinity*, which is how we indicate a null value (birthday isn't present on one or the other records). In this case, CS-ADTree would continue to check a number of other features to determine the match/unmatch status of the pair of records. So in both cases, the derived model is easy to understand and can be seen to be doing something reasonable.

[00102] *Efficiency* Our non-limiting experiments also show that ADTree and CS-ADTree are efficient at training and run-time. Given 50 candidate features, a training set of 11,309 example pairs, and a pre-computed feature vector, we trained the system on JBoost using CS-ADTree for 200 iterations in 314 seconds. On a major run of the system using a Python implementation and a 100 node model, the model performed 180 record pair comparisons per second per processor, including both the time to compute feature computations and the computation in the CS-ADTree. All statistics were on an Intel Xeon 2.53 GHz processor with 60 GB of RAM.

Performance of ADTree/CS-ADTree with active learning

[00103] The initial database $\mathcal{S}$ can be amplified by picking records from a data pool with size 8×10^6 which are then presented to Turkers for labeling. Denoting the pairwise classifier trained on $\mathcal{S}$ to be $\mathcal{H}$, the principle of choosing records are based on three active learning approaches listed as follows,

[00104] The first example non-limiting approach serves as our baseline active learning model, which is basically to randomly choose $r\in\mathcal{S}$, s.t., $\mathcal{H}(r)>-3$. The reason we choose -3 as the threshold to pick records is based on the empirical observation that examples in $\mathcal{S}$ that are given a prediction score by $\mathcal{H}$ greater than -3 exhibit an appropriate proportion between positive and negative examples, i.e., negative examples shouldn't be excessively outnumbered by positive examples to avoid Turkers from clicking "match" all the way through.

[00105] The second illustrative non-limiting approach is based on the randomized parameter approach. Instead of always splitting with the feature of minimum loss *Z*, we enable CS-ADTree to always *randomly* choose 1 of the 3 features with minimum losses across all features. A committee of 5 CS-ADTree's trained on $\mathcal{S}$ is then created as $\{\mathcal{H}_0,\mathcal{H}_1,\mathcal{H}_2,\mathcal{H}_3,\mathcal{H}_4\}$, records $r \in \mathcal{S}$ with minimal absolute committee score, i.e., $|\sum_{i=0}^{4} (\mathcal{H}_i(r))|$, will be picked for class labeling.

[00106] An additional example non-limiting approach is pretty straightforward. We choose records $r \in \mathcal{S}$ with minimal absolute prediction score by $\mathcal{H}$, i.e., $|\mathcal{H}(r)|$, since it represents a small *margin* given the natural threshold 0 which signifies the uncertainty of $\mathcal{H}$ for classifying the particular instance. This approaches resonates with our argument that a model that yields our desired *precision*, π at a threshold close to 0 will have close to optimal *recall*.

[00107] We pick 1000 pairs of records from the data pool using each of the three active learning approaches, which yields 3000 records in total. We also pick another 5000 pairs of records using active learning approach 1 to serve as the independent data set $\mathcal{E}$ for evaluating the learning models. The P-R curves of CS-ADTree trained on $\mathcal{S}$ plus the records picked by each of three active learning approaches are given in Fig. 19, which clearly shows active learning approaches 2 and 3 perform better than the baseline approach.

[00108] Expanded by the 3000 pairs of records picked by active learning approaches, $\mathcal{S}$ now becomes noisier since it is unavoidable that class labels assigned by Turkers will be partially erroneous. Using ADTree and CS-ADTree to derive classifiers from $\mathcal{S}$, Fig. 20 shows the P-R curves of applying these classifiers on independent data set $\mathcal{E}$, which clearly indicates CS-ADTree exhibits improved performance compared to ADTree for record linkage of a noisy database.

[00109] In order to demonstrate the competitiveness of ADTree/CS-ADTree against other popular ML techniques used by record linkage designers, we also apply decision tree (DT), boosted DT (T=200), and SVMs to classify pairs of records in $\mathcal{E}$. The kernel function selected for SVMs is the *Gaussian kernel* with σ=0.1. Empirical study discovers that SVMs cannot efficiently handle the original feature representation, i.e., a mixture of categorical and numerical features. Thus we apply two strategies to transform the features to cater for SVMs' needs. Categorical features are all transferred into discrete integers for both strategies. For numerical features, strategy 1 is uniform bucketing which evenly divides its data range into 5 buckets, and transforms the feature value into the integral index of the bucket it falls in. Strategy 2 is informative bucketing which efficiently makes use of the information provided by the CS-ADTree structure after training. If numerical feature *f* is split on values of $p_0<p_1<\ldots<p_n$ in the training process of CS-ADTree, any feature value *v* of *f* will be rewritten into integer *i*, s.t., $p_i<v<p_{i+1}$. Using 0 as the threshold for ADTree/CS-ADTree, Fig. 21 shows the *f-measure* of ADTree, CS-ADTree, boosted decision tree, decision tree, and SVM's using two bucketing strategies. It is obvious that ADTree and CS-ADTree both perform better than alternative ML techniques on record linkage of $\mathcal{E}$. They can also be used in combination with such other techniques.

[00110] Applicant incorporates herein by reference as if expressly set forth herein Chen et al, "The Case for Cost-Sensitive and Easy-To-Interpret Models in Industrial Record Linkage," 9th International Workshop on Quality in Databases (Seattle WA August 29, 2011).

[00111] While the technology herein has been described in connection with exemplary illustrative non-limiting embodiments, the invention is not to be limited by the disclosure. The invention is intended to be defined by the claims and to cover all corresponding and equivalent arrangements whether or not specifically disclosed herein.

WE CLAIM

1. A record linkage method comprising:

(a) acquiring data with a computer processing arrangement executing program instructions and connected to an electronic network;

(b) blocking the acquired data;

(c) applying a feature set with a machine learning model based at least in part on alternating decision trees to selectively link the blocked acquired data;

(d) storing the selectively-linked data in a non-transitory storage; and

(e) enabling users to access and search said non-transitory storage via said electronic network.

2. The method of claim 1 wherein the alternating decision trees are cost-sensitive.

3. A record linkage method comprising:

(a) acquiring data with a computer processing arrangement executing program instructions and connected to an electronic network;

(b) blocking the acquired data;

(c) applying a feature set with a machine learning model based at least in part on a cost-sensitive machine learning algorithm to selectively link the blocked acquired data;
(d) storing the selectively-linked data in a non-transitory storage; and
(e) enabling users to access and search said non-transitory storage via said electronic network.

4. A non-transitory storage medium arrangement storing computer program instructions that operate on plural records, said storage medium arrangement storing the following computer program instructions:

(a) blocking instructions that block the plural records and store the blocked plural records in a blocked database; and

(b) training instructions that operate on the contents of the blocked database to provide active machine learning, said active machine learning including generating a machine learned model using a cost-sensitive alternate decision tree.

5. The non-transitory storage medium arrangement of claim 4 wherein the storage medium arrangement further stores execution instructions that use said learned model including the cost-sensitive alternate decision tree to determine whether to link records.

6. The non-transitory storage medium arrangement of claim 5 wherein the execution instructions use the cost-sensitive alternate decision tree to link records by classifying a pair of records in a database into match or unmatch

7. The storage medium arrangement of claim 4 wherein the training instructions provide for recall at precisions in excess of 99%.

8. The non-transitory storage medium arrangement of claim 4 wherein the training instructions generate a learned model that is understandable by humans.

9. The non-transitory storage medium arrangement of claim 4 wherein the training instructions assign a cost factor $c_i \in (0,\infty)$ to each example x_i, y_i in the blocked database to quantify the cost of misclassifying x_i into a class label other than y_i.

10. The non-transitory storage medium arrangement of claim 4 wherein the training instructions update a weight distribution in a biased manner towards examples with higher costs.

11. The non-transitory storage medium arrangement of claim 4 wherein the training instructions provide boosting to derive a classifier from the blocked database.

12. A method of operating on plural records comprising:

(a) blocking the plural records to provide a blocked database; and

(b) operate on the contents of the blocked database to generate a machine learned model using a cost-sensitive alternate decision tree.

13. The method of claim 12 further including using said learned model including the cost-sensitive alternate decision tree to determine whether to link records.

14. The method of claim 13 further including using the cost-sensitive alternate decision tree to link records by classifying a pair of records in a database into match or unmatch.

15. The method of claim 12 further including providing for recall at precisions in excess of 99%.

16. The method of claim 12 further including generating a learned model that is understandable by humans.

17. The method of claim 12 further including updating a weight distribution in a biased manner towards examples with higher costs.

18. The method of claim 12 further including boosting to derive a classifier from the blocked database.

19. A system for operating on plural records comprising:

a blocker that blocks the plural records to provide a blocked database;

a machine learner that operates on the contents of the blocked database to generate a machine learned model using an alternate decision tree; and

an execution component connected to the Internet that provides a background check or an identity check on demand at least in part in response to records linked based on said machine-learned model in response to a user search requested over the Internet.

20. The system of claim 19 wherein the alternate decision tree is cost-sensitive.

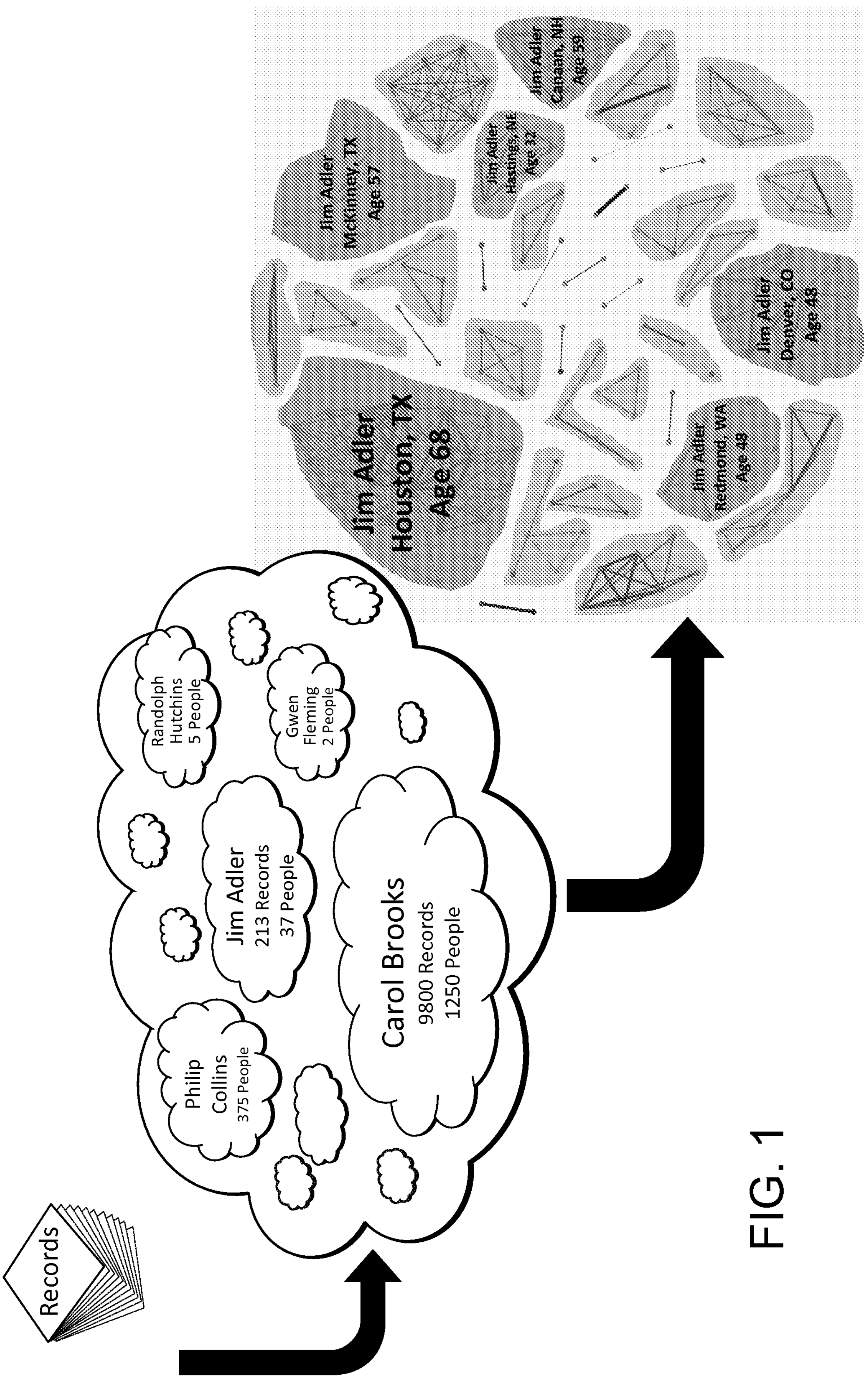

Records
Philip Collins 375 People
Jim Adler 213 Records 37 People
Carol Brooks 9800 Records 1250 People
Randolph Hutchins 5 People
Gwen Fleming 2 People
Jim Adler Houston, TX Age 68
Jim Adler McKinney, TX Age 57
Jim Adler Hastings, NE Age 32
Jim Adler Canaan, NH Age 59
Jim Adler Denver, CO Age 48
Jim Adler Redmond, WA Age 48

FIG. 1

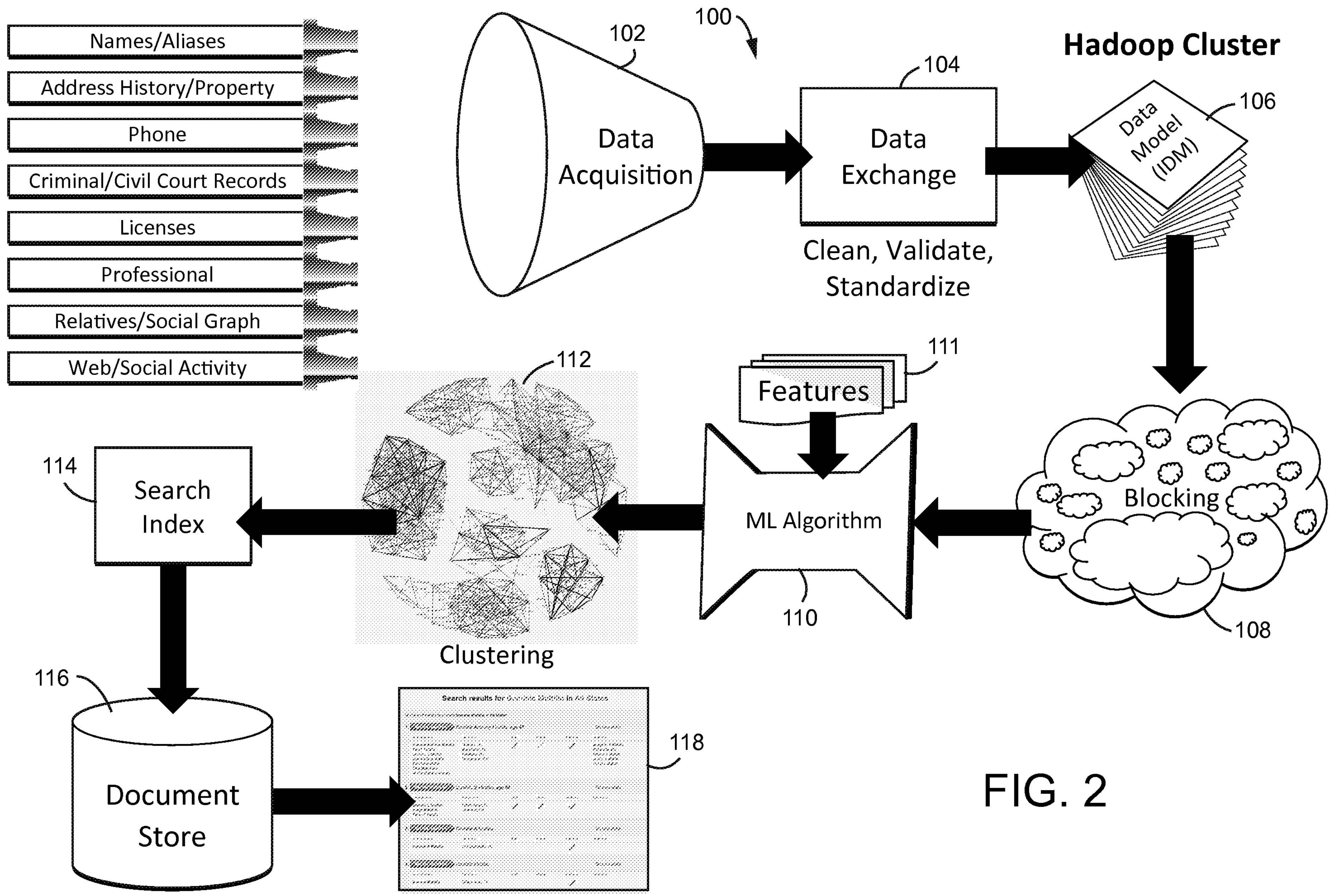
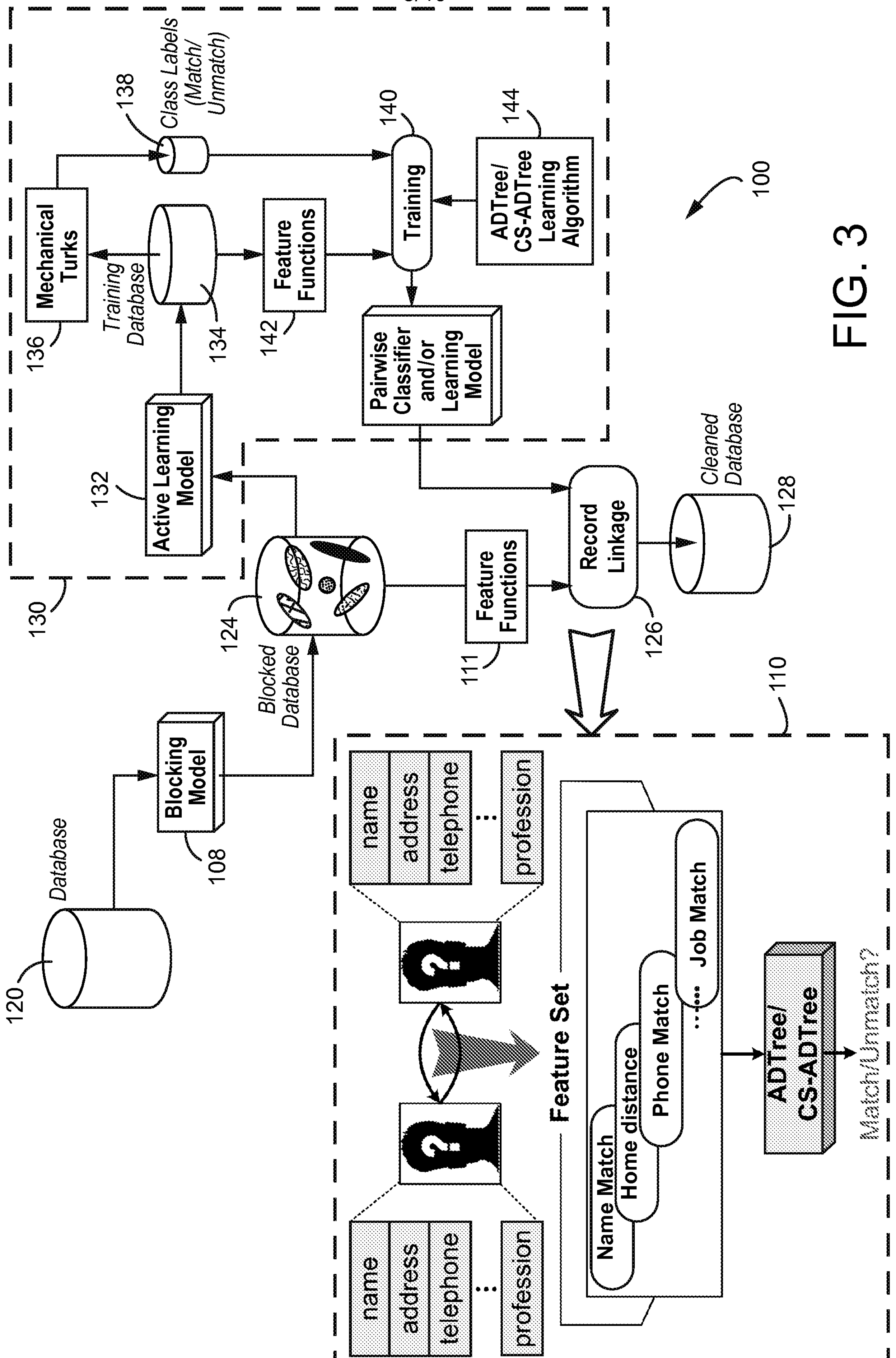

Names/Aliases
Address History/Property
Phone
Criminal/Civil Court Records
Licenses
Professional
Relatives/Social Graph
Web/Social Activity
100
102
Data Acquisition
104
Data Exchange
Clean, Validate, Standardize
Hadoop Cluster
106
Data Model (IDM)
108
Blocking
111
Features
ML Algorithm
110
112
Clustering
114
Search Index
116
Document Store
118
Search results for

FIG. 2



130
Active Learning Model
132
Training Database
134
Mechanical Turks
136
138
Class Labels (Match/ Unmatch)
Feature Functions
142
Training
140
ADTree/ CS-ADTree Learning Algorithm
144
Pairwise Classifier and/or Learning Model
100
120
Database
Blocking Model
108
Blocked Database
124
Feature Functions
111
Record Linkage
126
Cleaned Database
128
110
name
address
telephone
...
profession
Feature Set
Name Match
Home distance
Phone Match
Job Match
ADTree/ CS-ADTree
Match/Unmatch?

FIG. 3

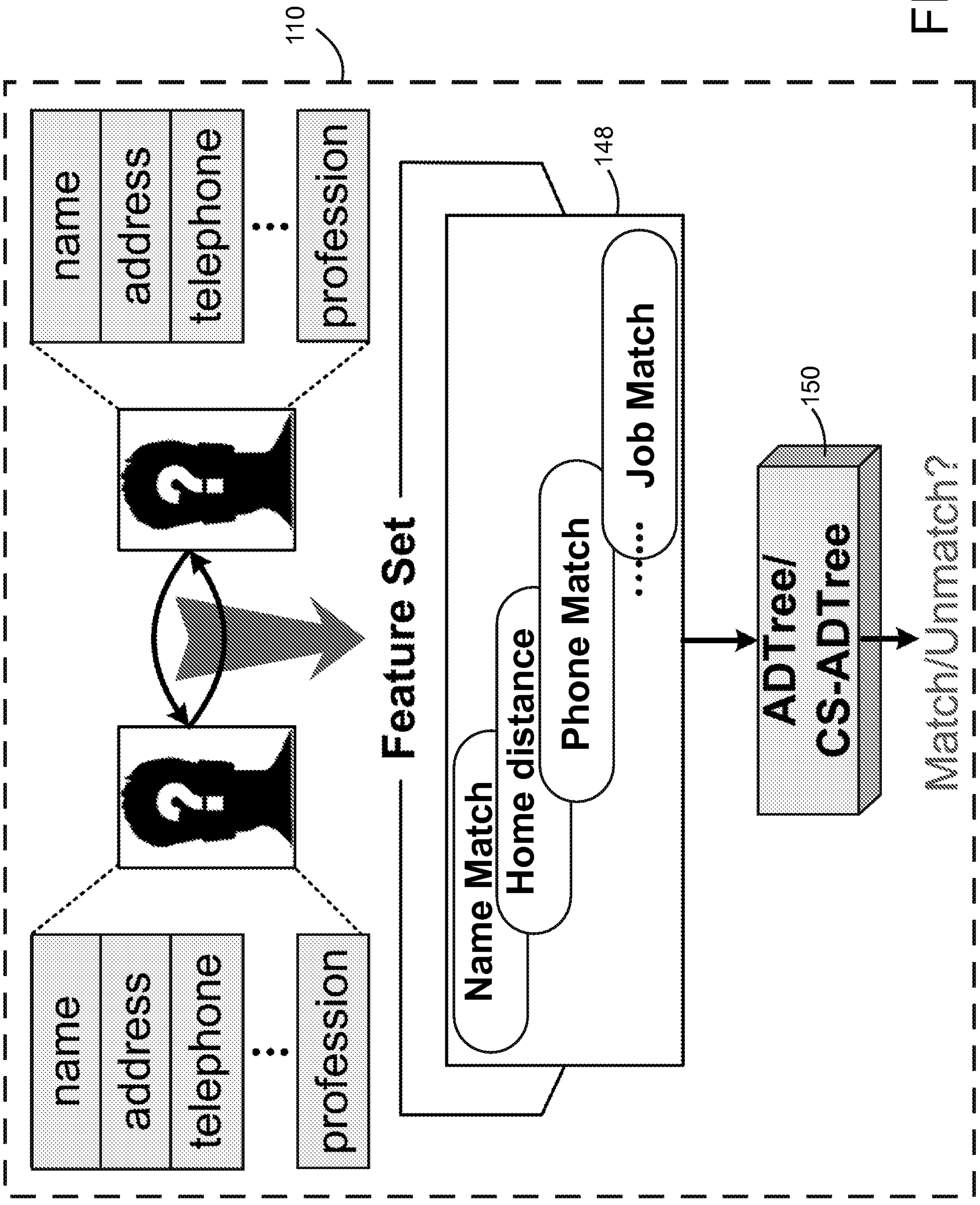
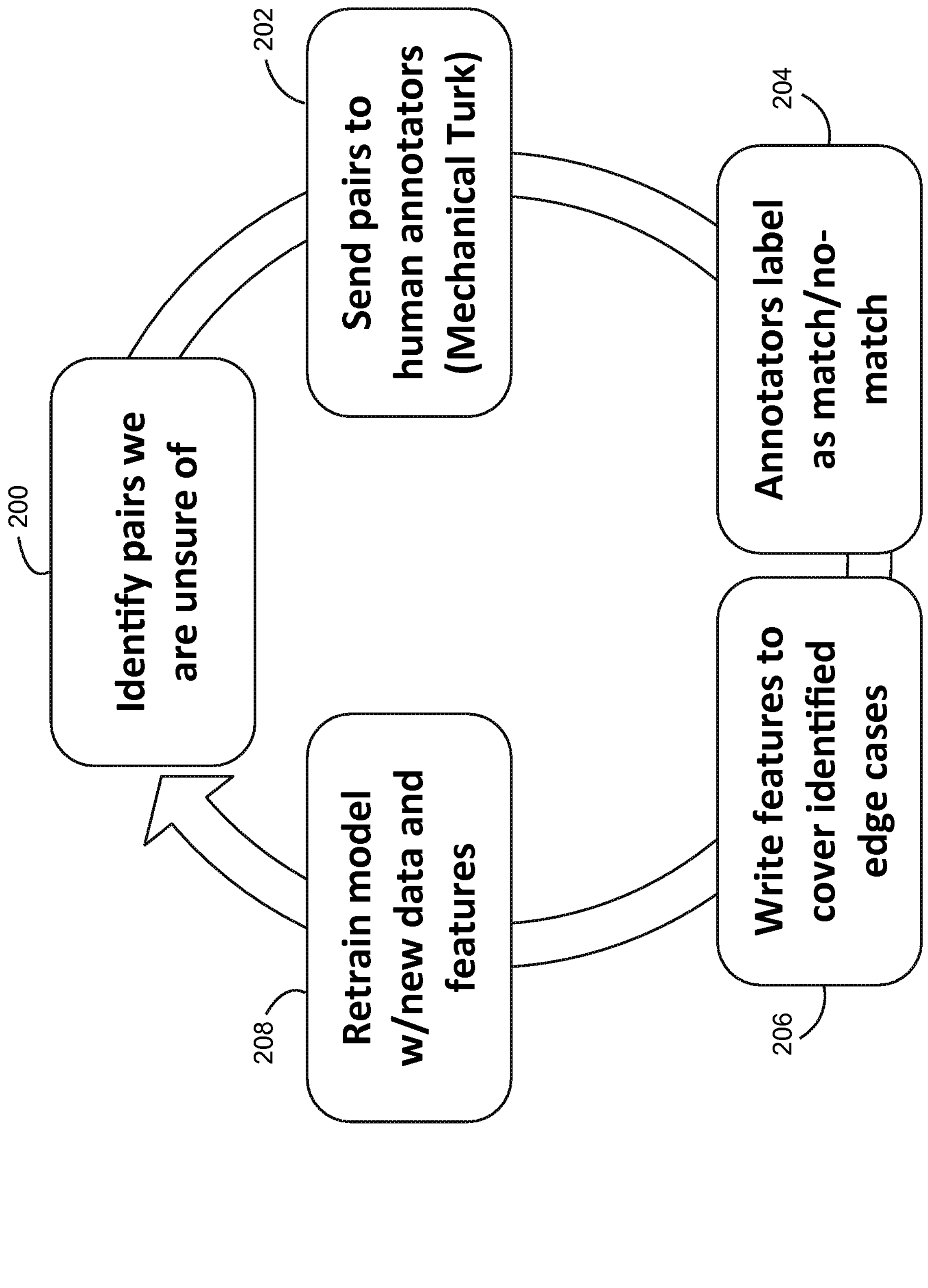

110
name
address
telephone
...
profession
name
address
telephone
...
profession
Feature Set
148
Name Match
Home distance
Phone Match
......
Job Match
150
ADTree/
CS-ADTree
Match/Unmatch?

FIG. 4

Field	Record 1	Record	Feature
Name	Robert Jones	Robert Jones	Name Frequency = c. 882.6
Birthday	3/1/1966	3/15/1966	Difference = 14 days
Street address	234 Chestnut St.	121 Walnut St.	Exact match of street = False
City/State	Seattle, WA	Seattle, WA	Population of Seattle region = c. 3.3 million
Phone number	206-555-1234	206-345-9123	Type of phone match = area code

FIG. 5

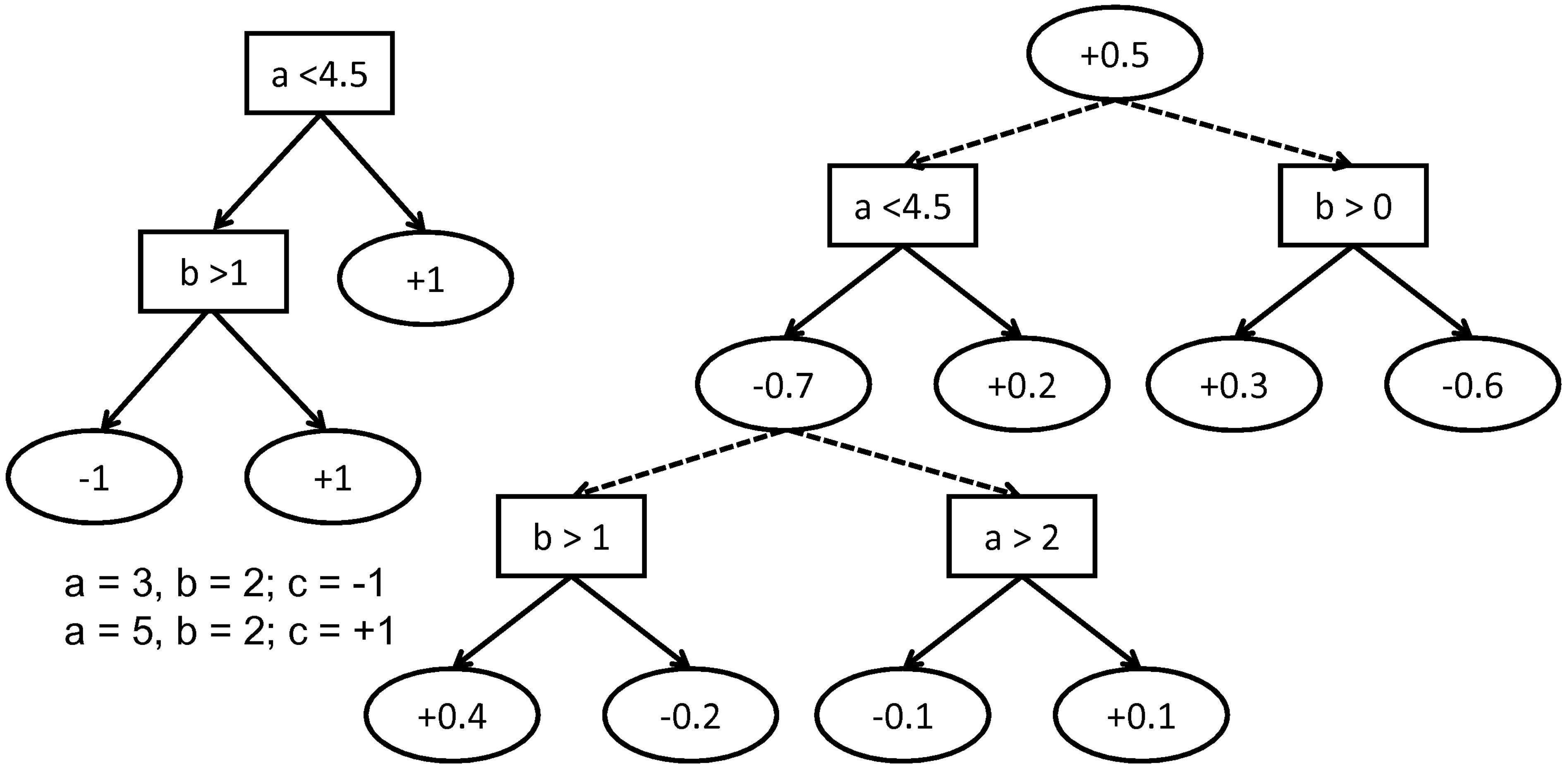


200
Identify pairs we are unsure of
202
Send pairs to human annotators (Mechanical Turk)
204
Annotators label as match/no-match
206
Write features to cover identified edge cases
208
Retrain model w/new data and features

FIG. 6

Algorithm	Human interpretability	Cost Sensitivity	Captures Feature Interactions	Handle Diverse Feature Types
Decision Trees	Yes	No	Yes	Yes
Support Vector Machines	No	With Custom Kernel	With Custom Kernel	Not Easily
Boosted Decision Trees	No	No	Yes	Yes
Alternating Decision Trees	Yes	No	Yes	Yes
CS-ADT's	Yes	Yes	Yes	Yes

FIG. 7



a = 3, b = 2; c = -1
a = 5, b = 2; c = +1

a = 3, b = 2; c = sgn(0.5 + 0.3 – 0.7 + 0.4 – 0.1) = +1
a = 5, b = 2; c = sgn(0.5 + 0.3 + 0.2) = +1

FIG. 8

		True class	
		P	N
Hypothesis output	P	$C(P, P)$	$C(P, N)$
	N	$C(N, P)$	$C(N, N)$

FIG. 9

A cost matrix for cost sensitive learning

ID	*Company*	*Address*		*Name*	*Phone #*
Person 1	Evanston Chiropractic Life Center Inc	5716 S Evanston Ave	IN, 46123	Ronald Eugene Raulston, ***Sr.***	219-821-8900
Person 2	Evanston Chiropractic Life Center Inc	5716 S Evanston Ave	IN, 46123	Ronald Eugene Raulston, ***Jr.***	219-821-8900

FIG. 10

An example of a pair of person profile records

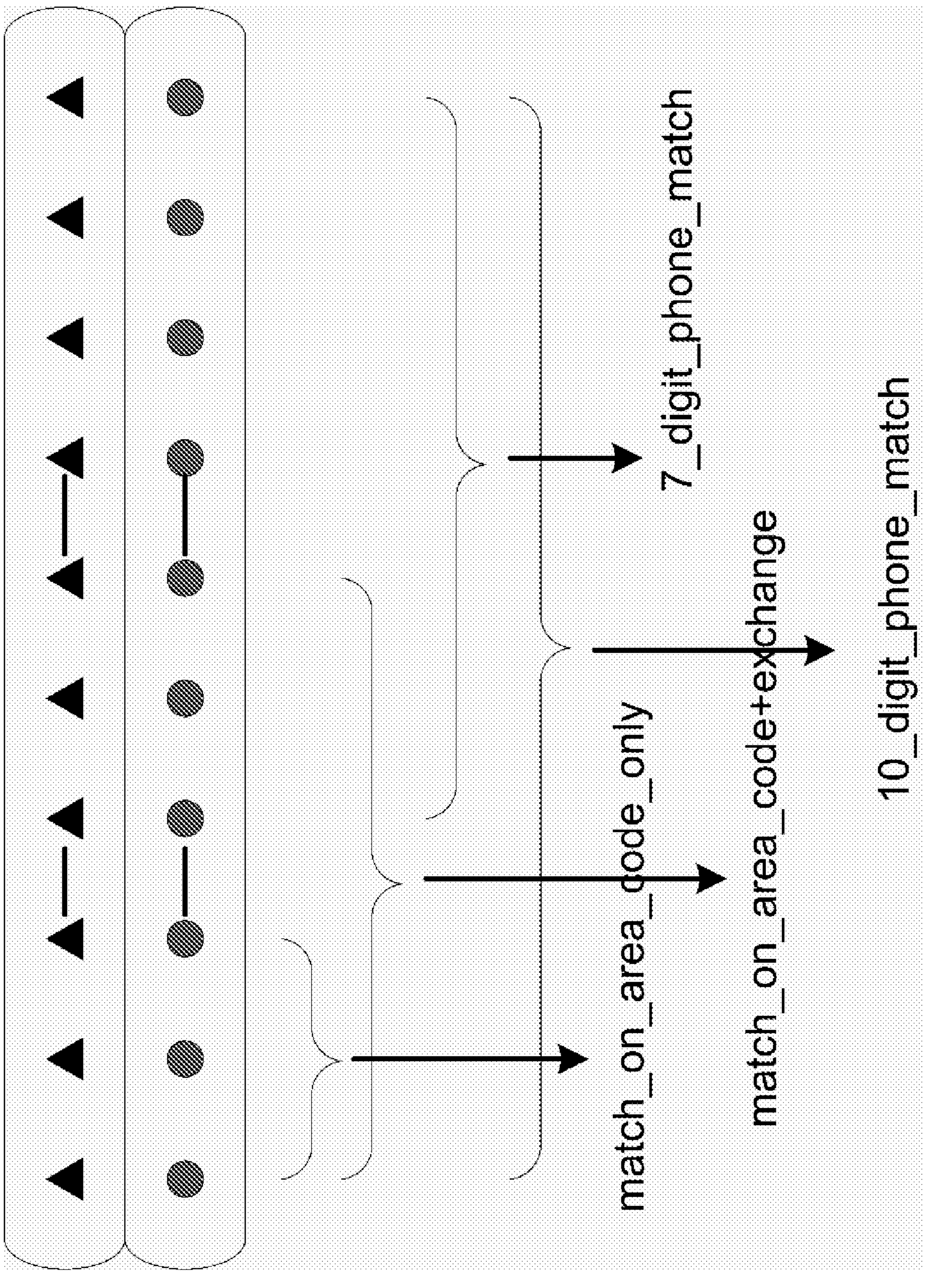


FIG. 11
Feature values of phone_match

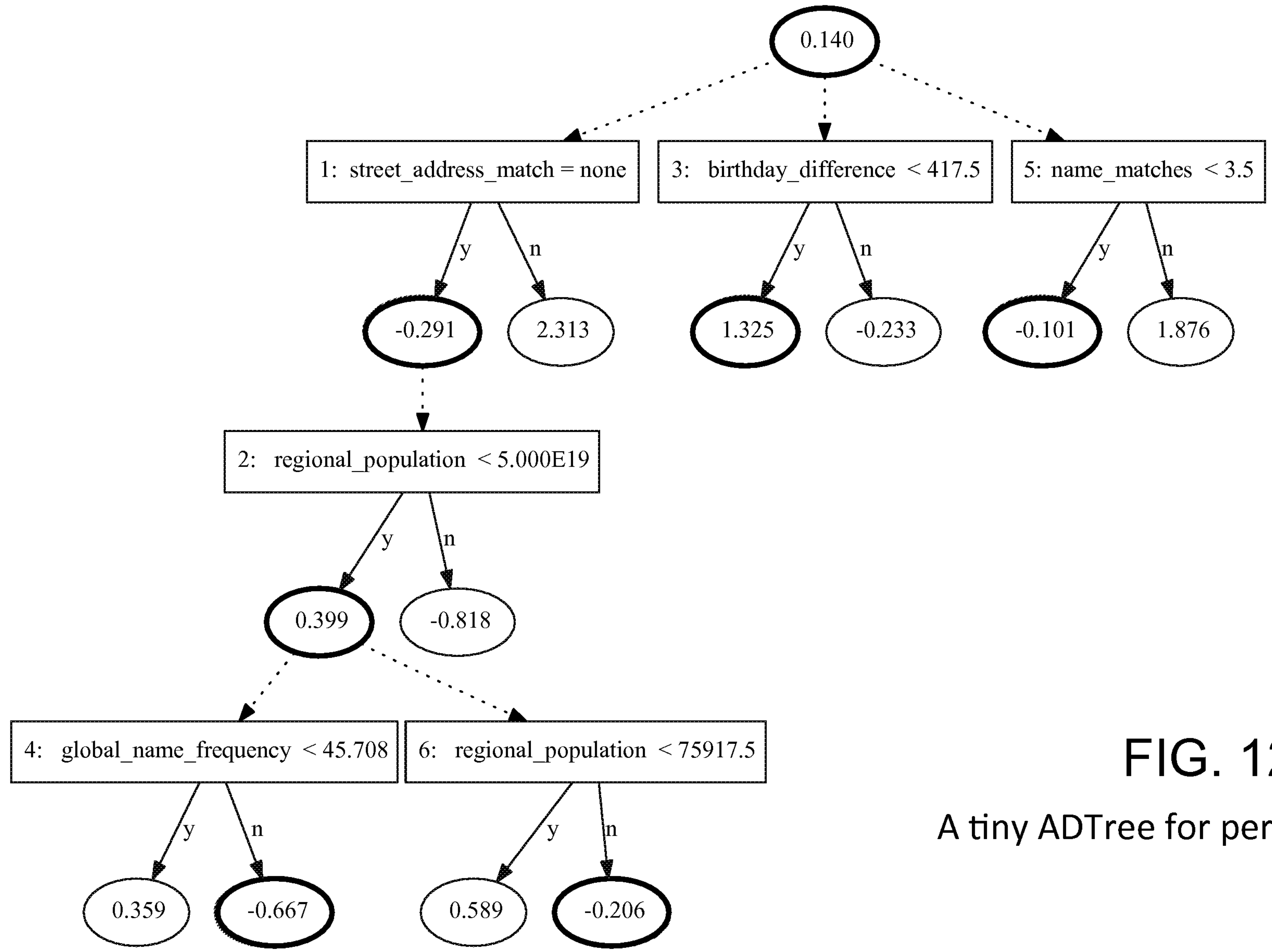

0.140
1: street_address_match = none
3: birthday_difference < 417.5
5: name_matches < 3.5
y
n
-0.291
2.313
1.325
-0.233
-0.101
1.876
2: regional_population < 5.000E19
0.399
-0.818
4: global_name_frequency < 45.708
6: regional_population < 75917.5
0.359
-0.667
0.589
-0.206

FIG. 12

A tiny ADTree for person linkage

		True class	
		P	*N*
Hypothesis output	*P*	TP	FN
	N	FP	TN

FIG. 13

A confusion matrix for binary classification

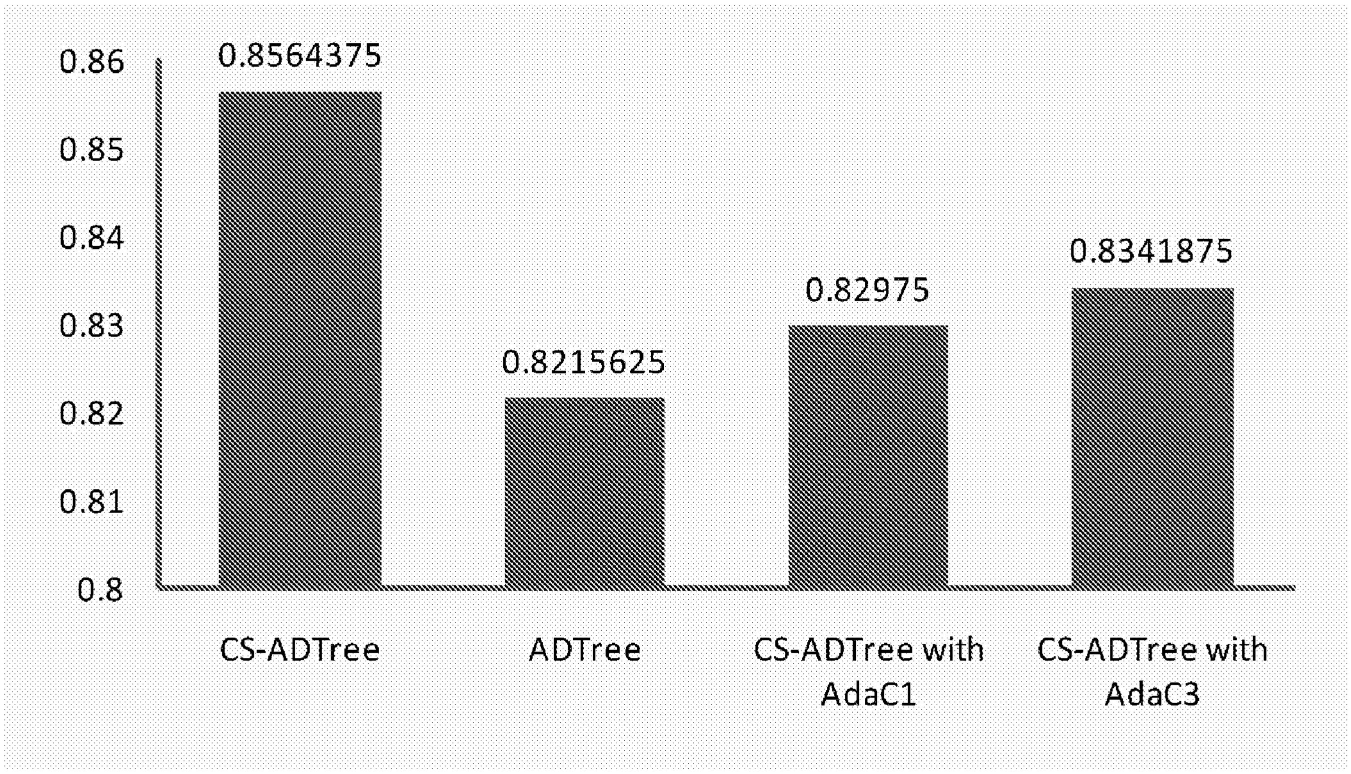


FIG. 14

Comparison of average recall's for ADTree/CS-ADTree

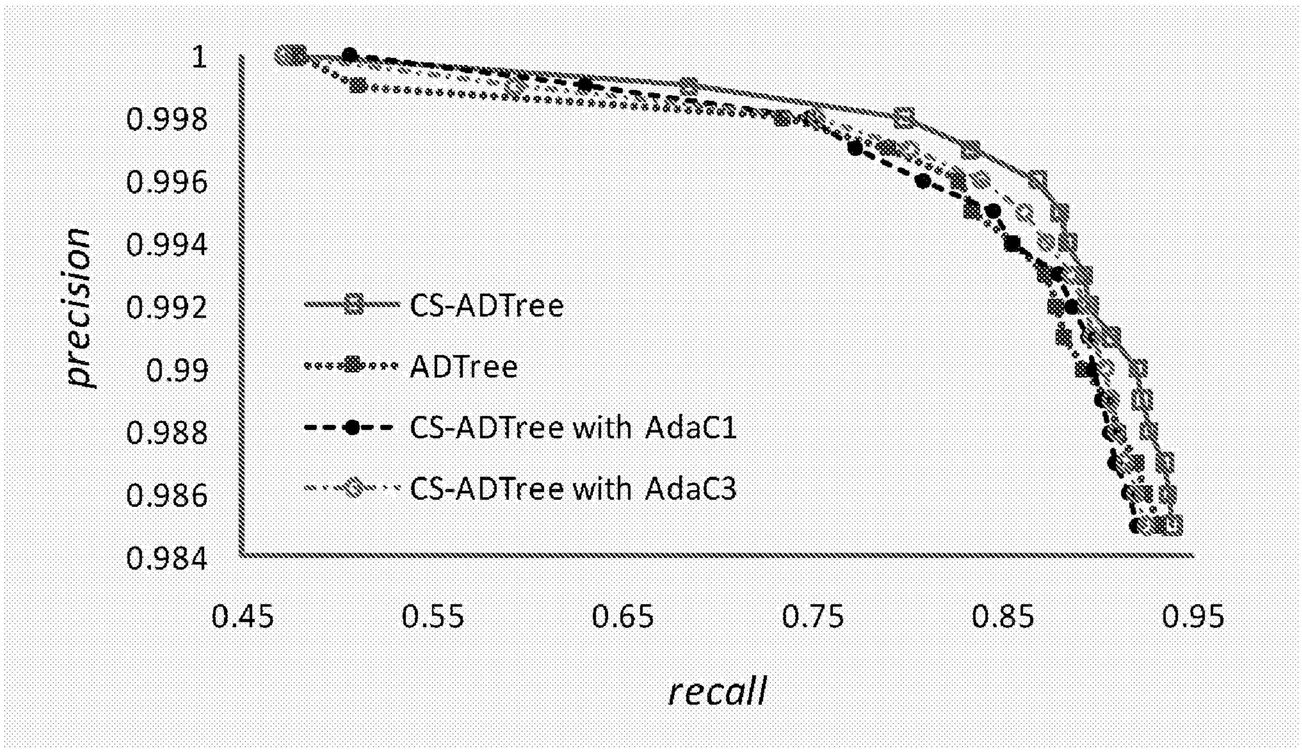


FIG. 15

P-R curves of ADTree/CS-ADTree

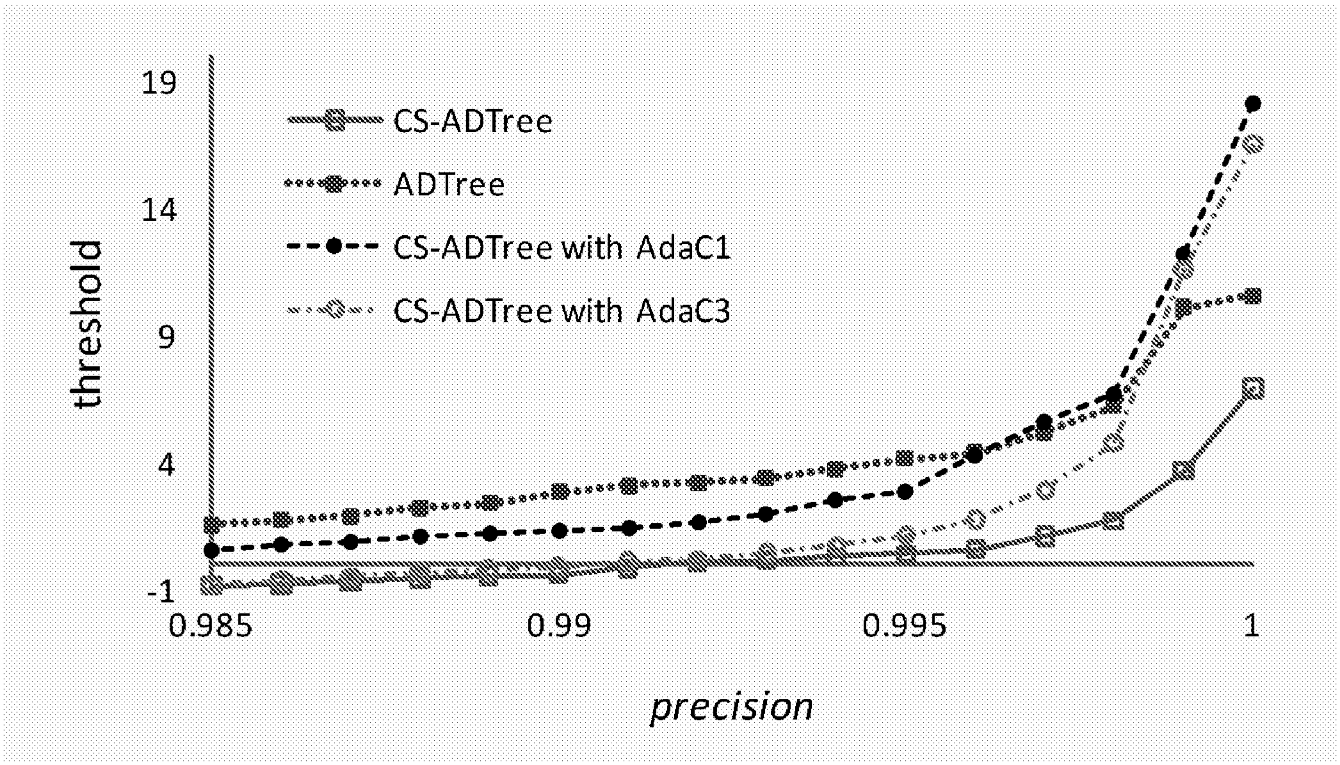
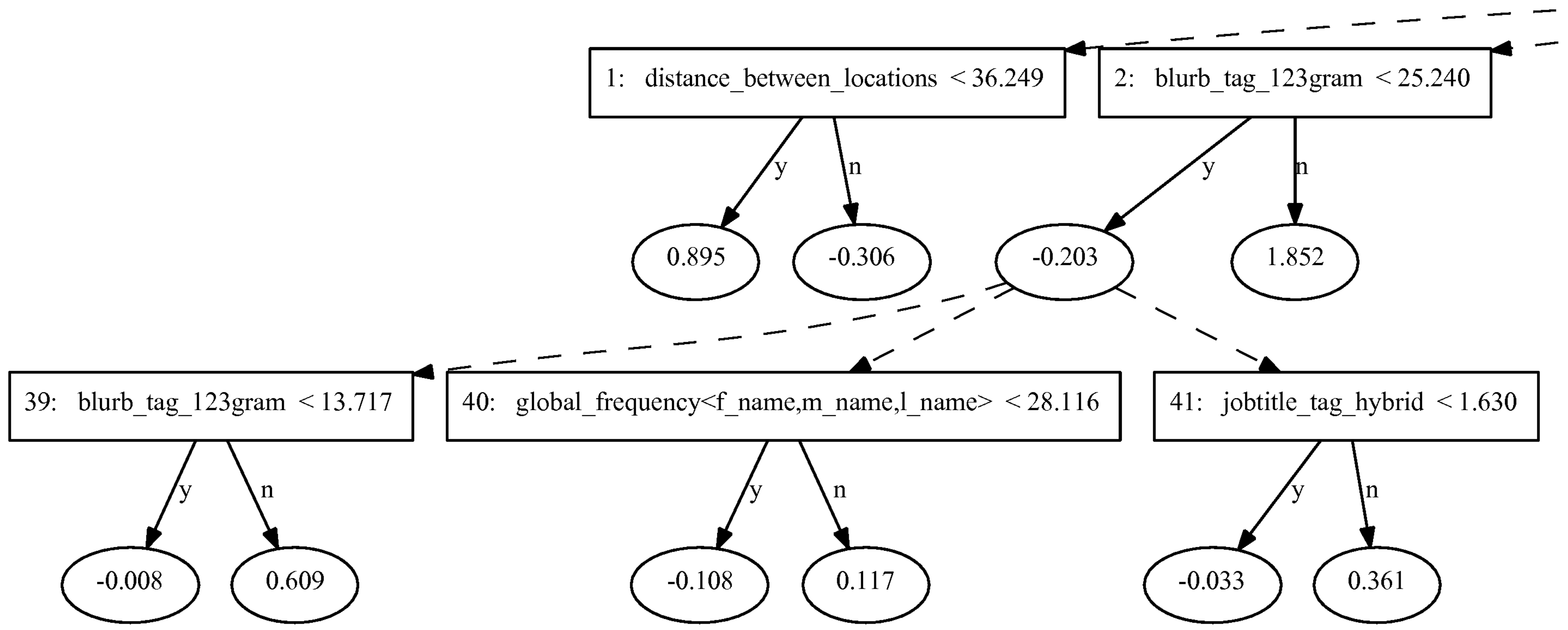


FIG. 16

Threshold range of ADTree/CS-ADTree w.r.t high range



1: distance_between_locations < 36.249
2: blurb_tag_123gram < 25.240
y
n
y
n
0.895
-0.306
-0.203
1.852
39: blurb_tag_123gram < 13.717
40: global_frequency<f_name,m_name,l_name> < 28.116
41: jobtitle_tag_hybrid < 1.630
y
n
y
n
y
n
-0.008
0.609
-0.108
0.117
-0.033
0.361

FIG. 17A

Partial CS-ADTree. If name_suffix=differ, we get a strong negative score and don’t query additional features.

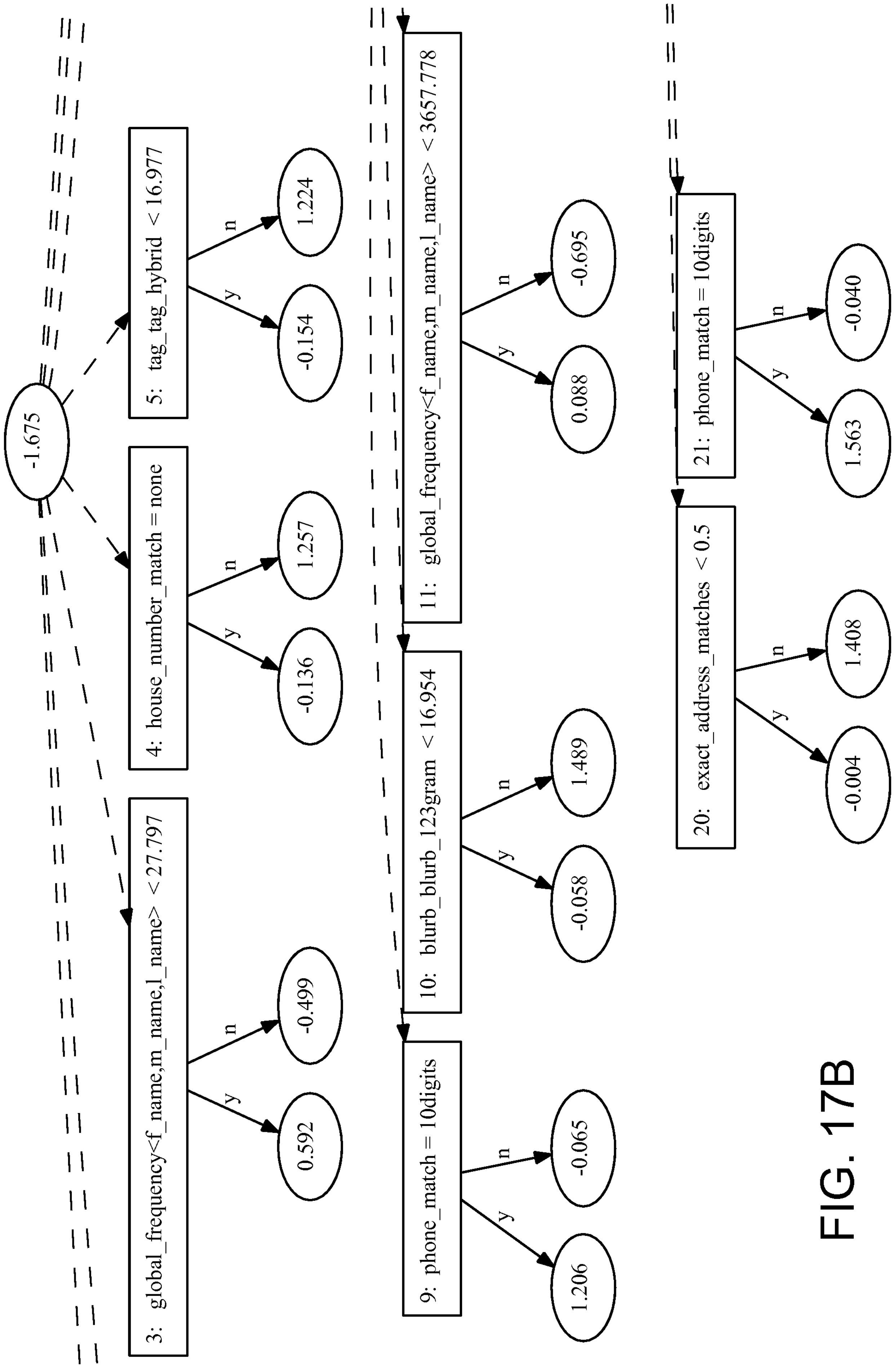

-1.675
5: tag_tag_hybrid < 16.977
n
1.224
y
-0.154
4: house_number_match = none
n
1.257
y
-0.136
3: global_frequency<f_name,m_name,l_name> < 27.797
n
-0.499
y
0.592
11: global_frequency<f_name,m_name,l_name> < 3657.778
n
-0.695
y
0.088
10: blurb_blurb_123gram < 16.954
n
1.489
y
-0.058
9: phone_match = 10digits
n
-0.065
y
1.206
21: phone_match = 10digits
n
-0.040
y
1.563
20: exact_address_matches < 0.5
n
1.408
y
-0.004

FIG. 17B

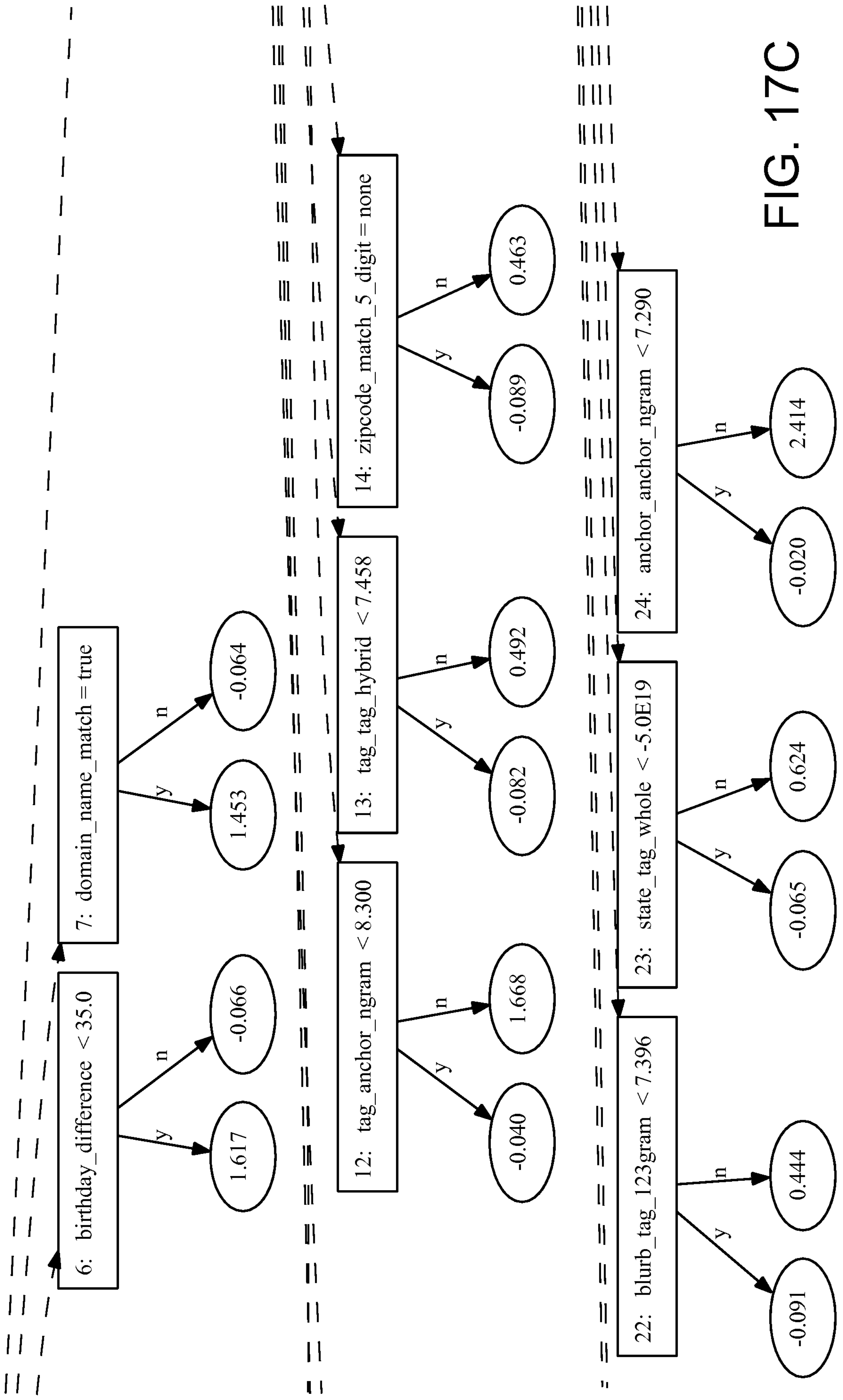

6: birthday_difference < 35.0
y
n
1.617
-0.066
7: domain_name_match = true
y
n
1.453
-0.064
12: tag_anchor_ngram < 8.300
y
n
-0.040
1.668
13: tag_tag_hybrid < 7.458
y
n
-0.082
0.492
14: zipcode_match_5_digit = none
y
n
-0.089
0.463
22: blurb_tag_123gram < 7.396
y
n
-0.091
0.444
23: state_tag_whole < -5.0E19
y
n
-0.065
0.624
24: anchor_anchor_ngram < 7.290
y
n
-0.020
2.414

FIG. 17C

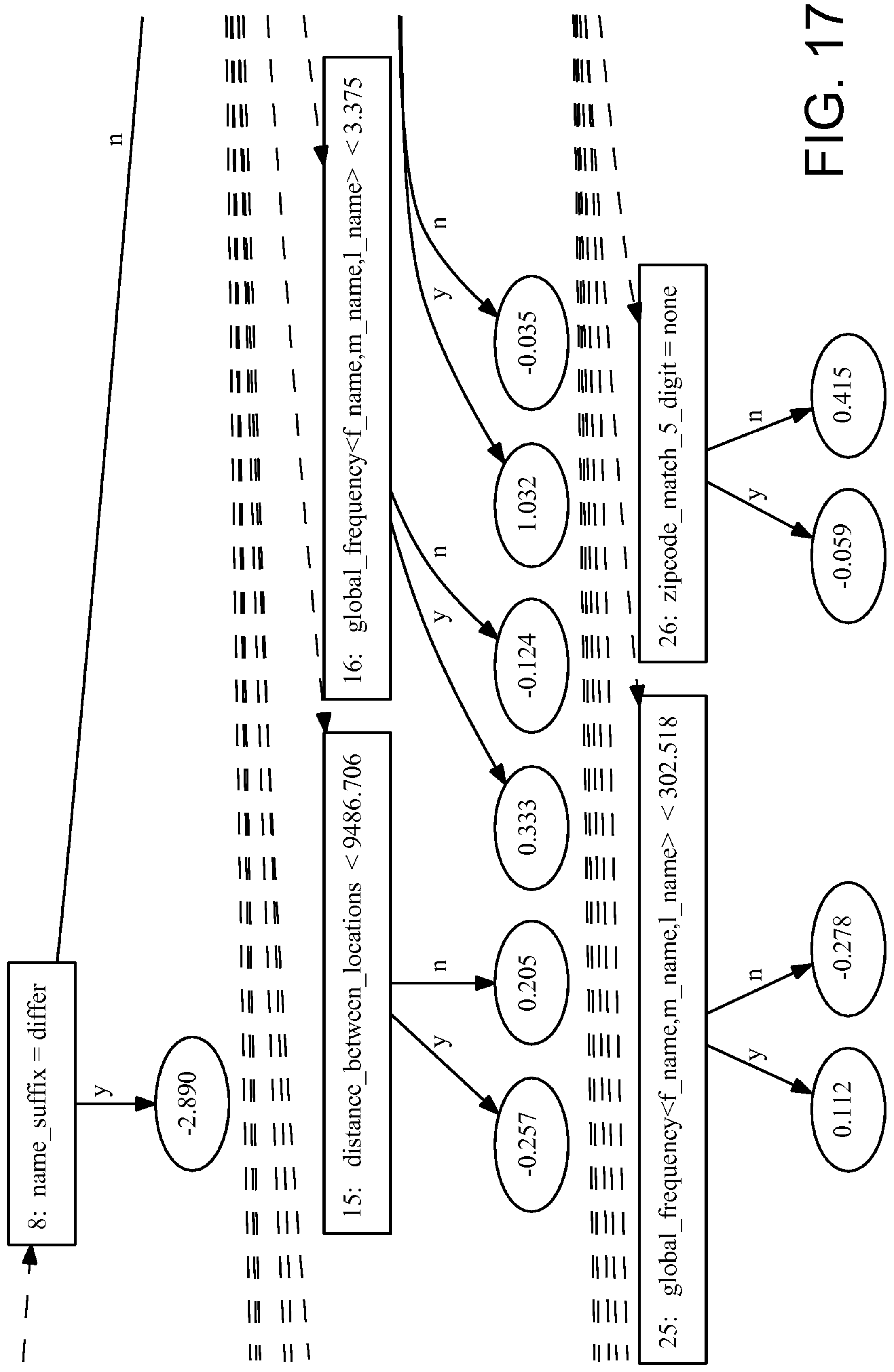
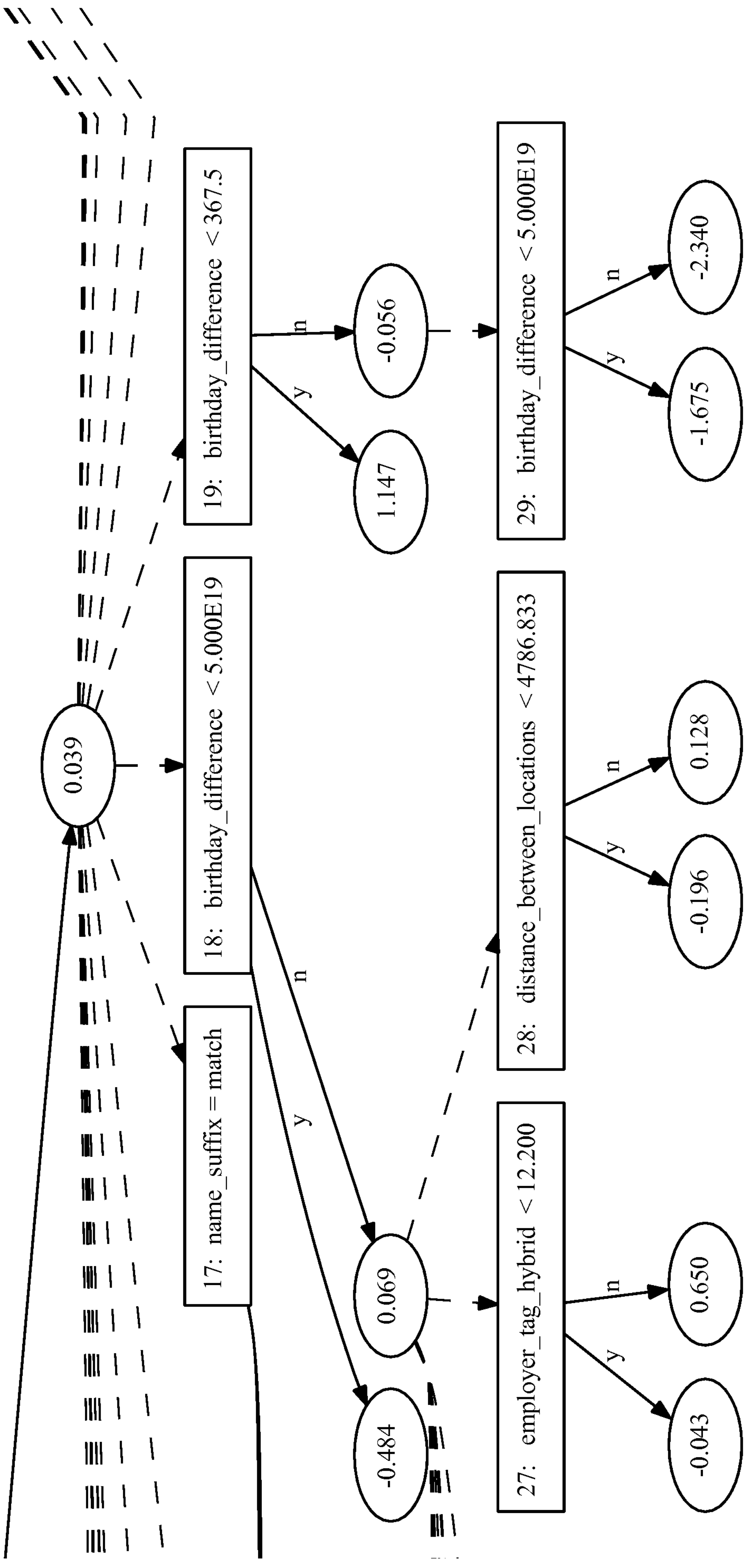

8: name_suffix = differ
y
n
-2.890
15: distance_between_locations < 9486.706
y
n
-0.257
0.205
16: global_frequency<f_name,m_name,l_name> < 3.375
y
n
0.333
-0.124
y
n
1.032
-0.035
25: global_frequency<f_name,m_name,l_name> < 302.518
y
n
0.112
-0.278
26: zipcode_match_5_digit = none
y
n
-0.059
0.415

FIG. 17D

FIG. 17E


0.039
17: name_suffix = match
18: birthday_difference < 5.000E19
19: birthday_difference < 367.5
n
y
-0.056
1.147
0.069
-0.484
29: birthday_difference < 5.000E19
28: distance_between_locations < 4786.833
27: employer_tag_hybrid < 12.200
-2.340
-1.675
0.128
-0.196
0.650
-0.043

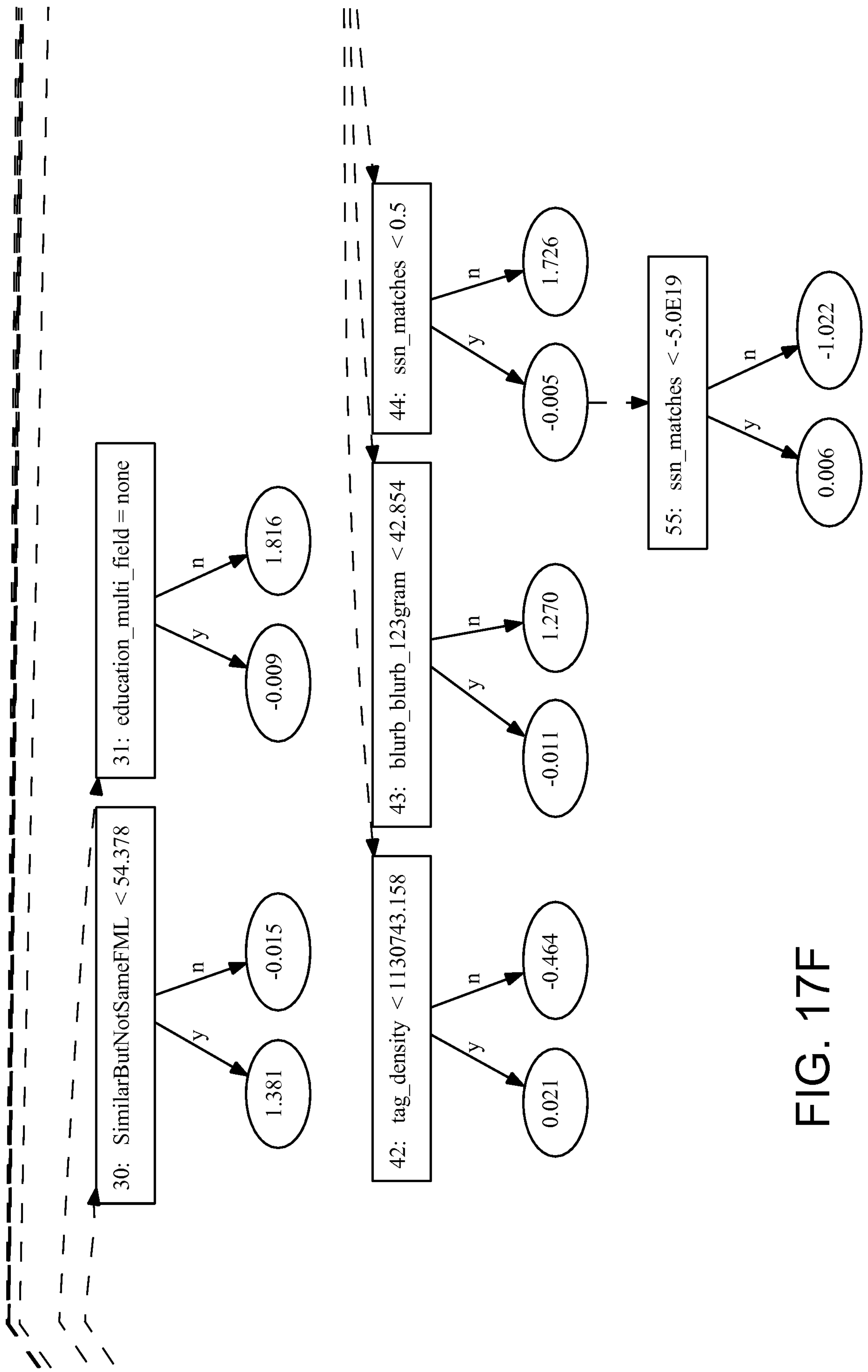

30: SimilarButNotSameFML < 54.378
n
-0.015
y
1.381
31: education_multi_field = none
n
1.816
y
-0.009
42: tag_density < 1130743.158
n
-0.464
y
0.021
43: blurb_blurb_123gram < 42.854
n
1.270
y
-0.011
44: ssn_matches < 0.5
n
1.726
y
-0.005
55: ssn_matches < -5.0E19
n
-1.022
y
0.006

FIG. 17F

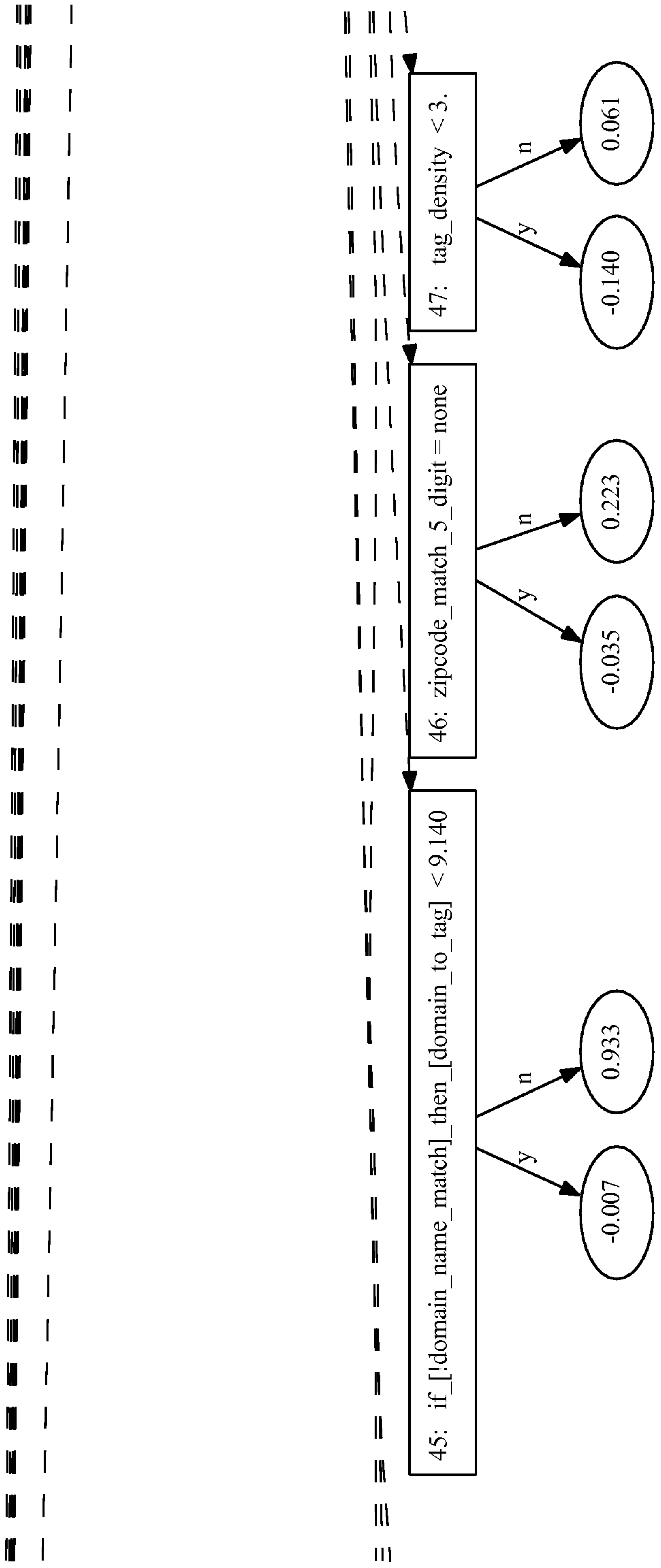

47: tag_density < 3.
n
0.061
y
-0.140
46: zipcode_match_5_digit = none
n
0.223
y
-0.035
45: if_[!domain_name_match]_then_[domain_to_tag] < 9.140
n
0.933
y
-0.007

FIG. 17G

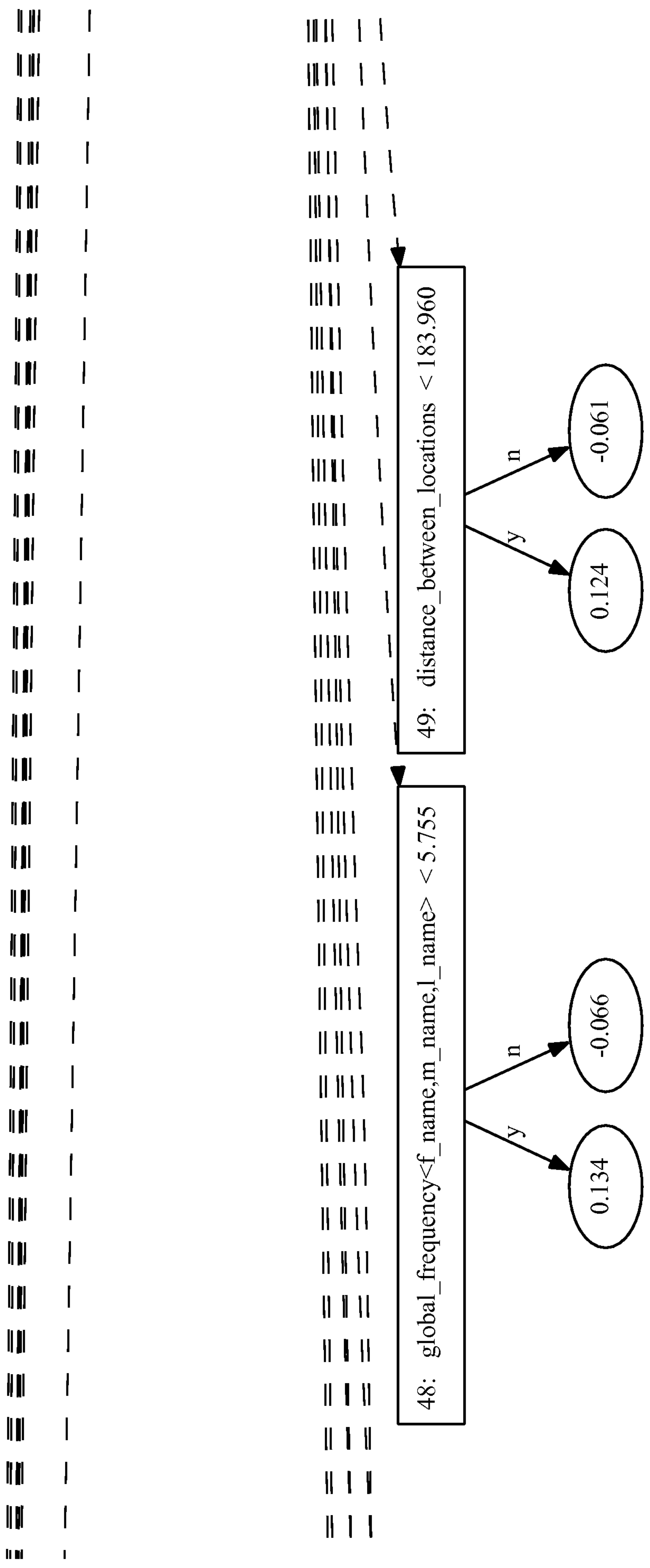

49: distance_between_locations < 183.960
n
-0.061
y
0.124
48: global_frequency<f_name,m_name,l_name> < 5.755
n
-0.066
y
0.134

FIG. 17H

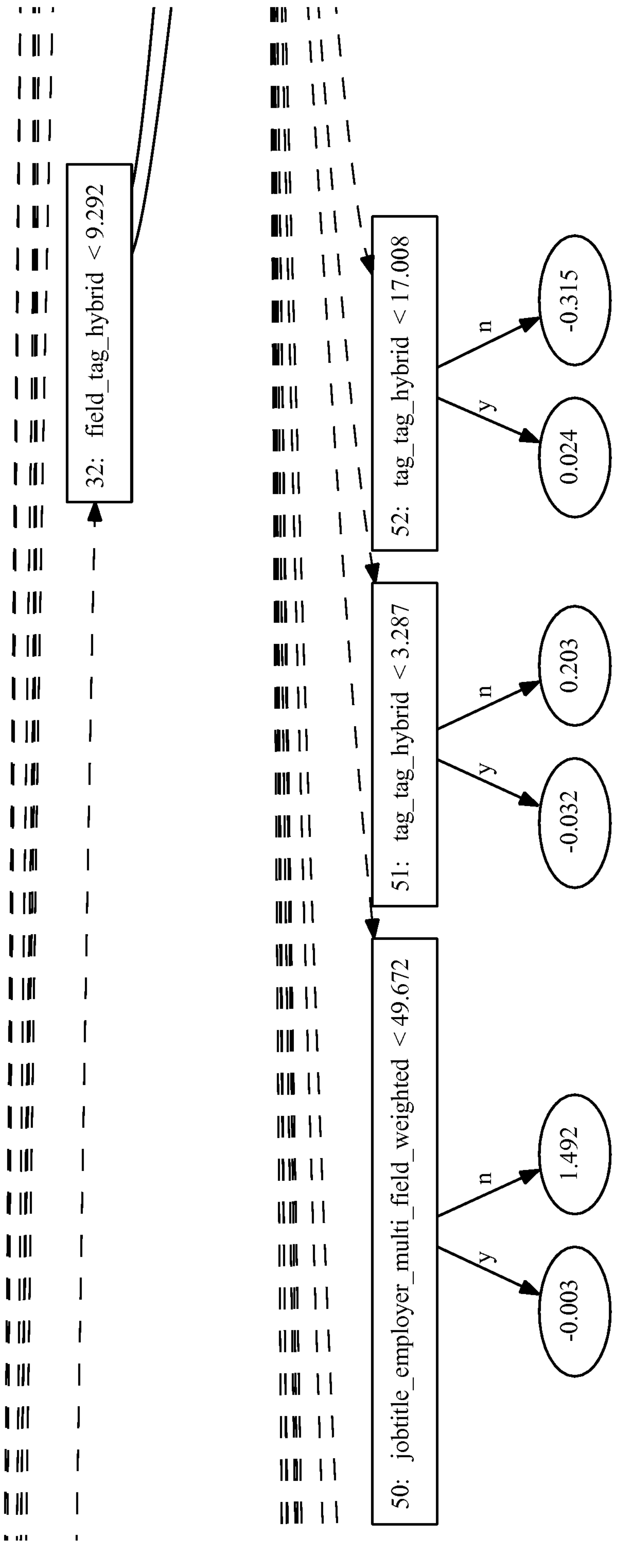
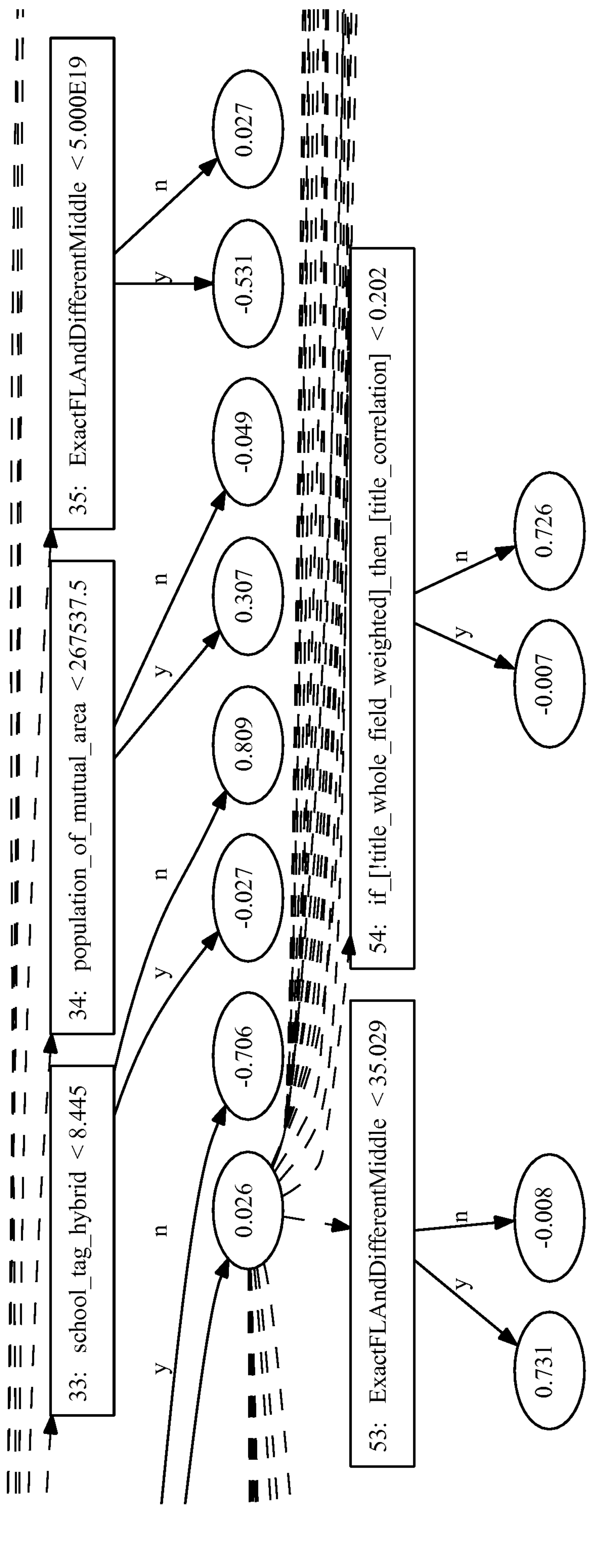

32: field_tag_hybrid < 9.292
52: tag_tag_hybrid < 17.008
n
-0.315
y
0.024
51: tag_tag_hybrid < 3.287
n
0.203
y
-0.032
50: jobtitle_employer_multi_field_weighted < 49.672
n
1.492
y
-0.003

FIG. 17I



33: school_tag_hybrid < 8.445
34: population_of_mutual_area < 267537.5
35: ExactFLAndDifferentMiddle < 5.000E19
53: ExactFLAndDifferentMiddle < 35.029
54: if_[!title_whole_field_weighted]_then_[title_correlation] < 0.202
y
n
0.026
-0.706
-0.027
0.809
0.307
-0.049
-0.531
0.027
0.731
-0.008
-0.007
0.726

FIG. 17J

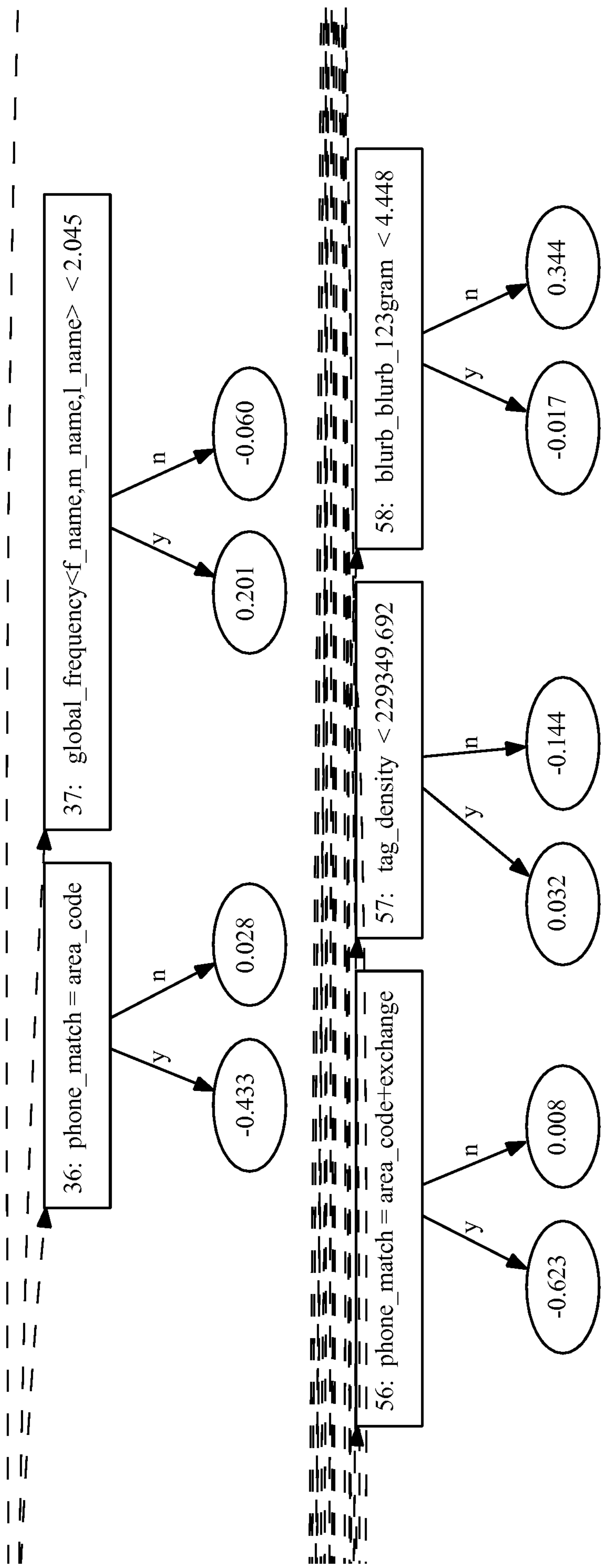

36: phone_match = area_code
y
-0.433
n
0.028
37: global_frequency<f_name,m_name,l_name> < 2.045
y
0.201
n
-0.060
56: phone_match = area_code+exchange
y
-0.623
n
0.008
57: tag_density < 229349.692
y
0.032
n
-0.144
58: blurb_blurb_123gram < 4.448
y
-0.017
n
0.344

FIG. 17K

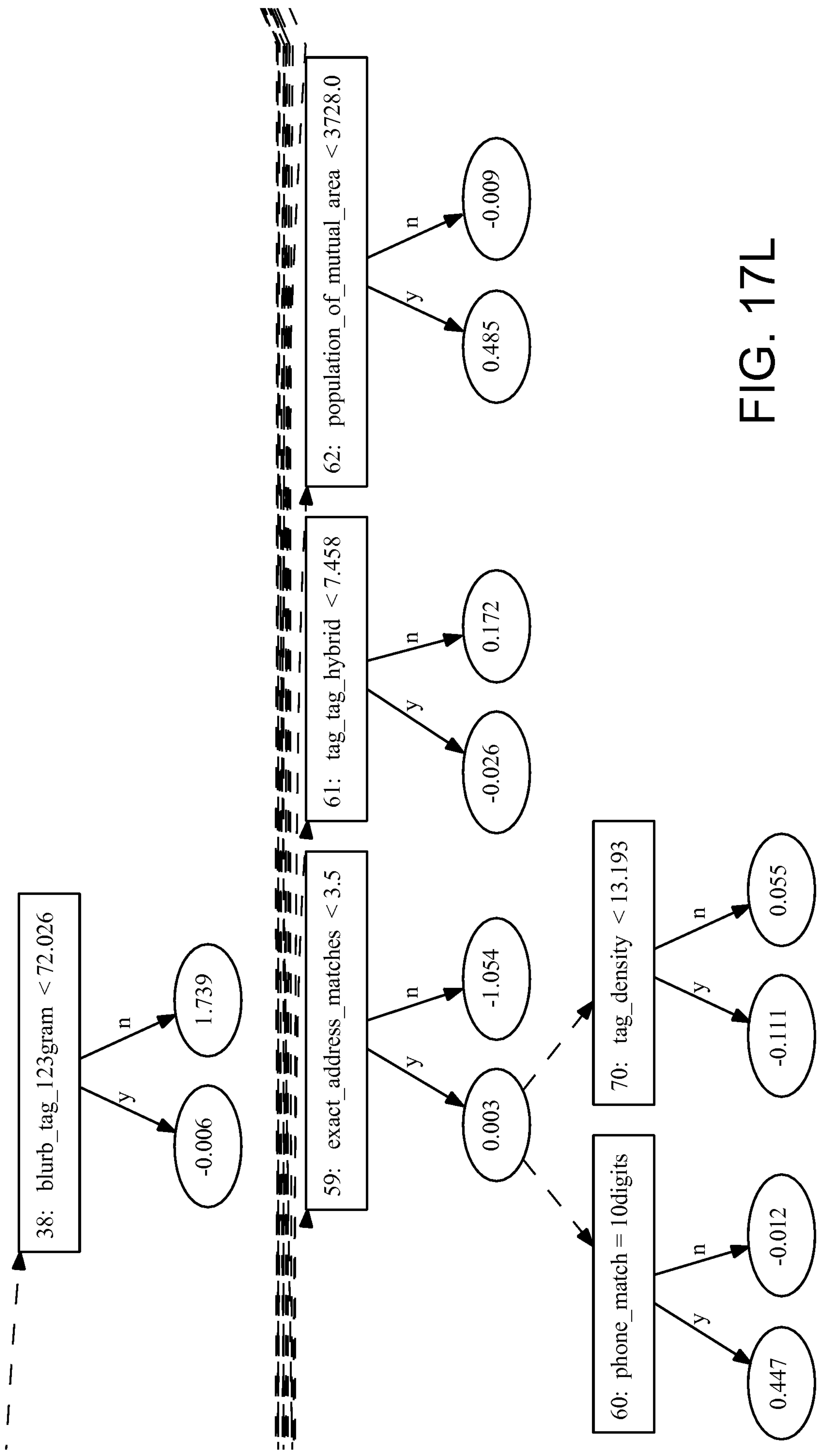

38: blurb_tag_123gram < 72.026
n
y
1.739
-0.006
59: exact_address_matches < 3.5
n
y
-1.054
0.003
61: tag_tag_hybrid < 7.458
n
y
0.172
-0.026
62: population_of_mutual_area < 3728.0
n
y
-0.009
0.485
70: tag_density < 13.193
n
y
0.055
-0.111
60: phone_match = 10digits
n
y
-0.012
0.447

FIG. 17L

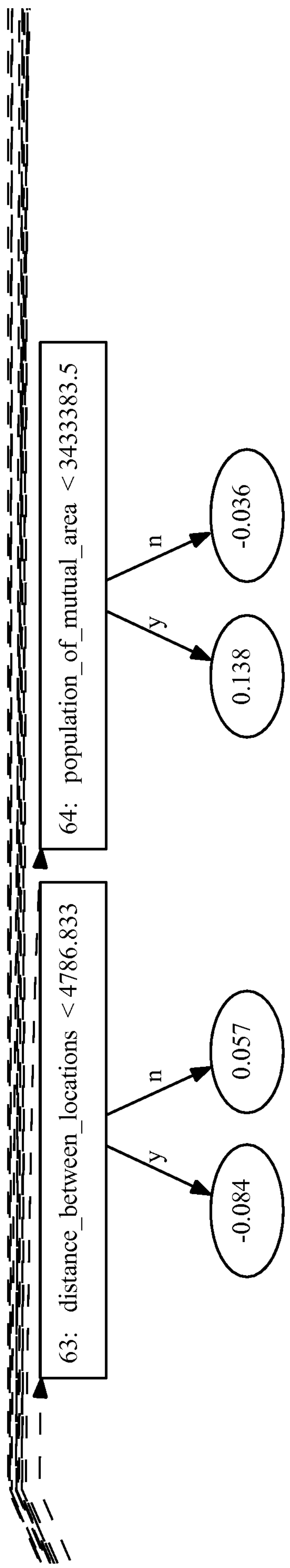

64: population_of_mutual_area < 3433383.5
n
y
-0.036
0.138
63: distance_between_locations < 4786.833
n
y
0.057
-0.084

FIG. 17M

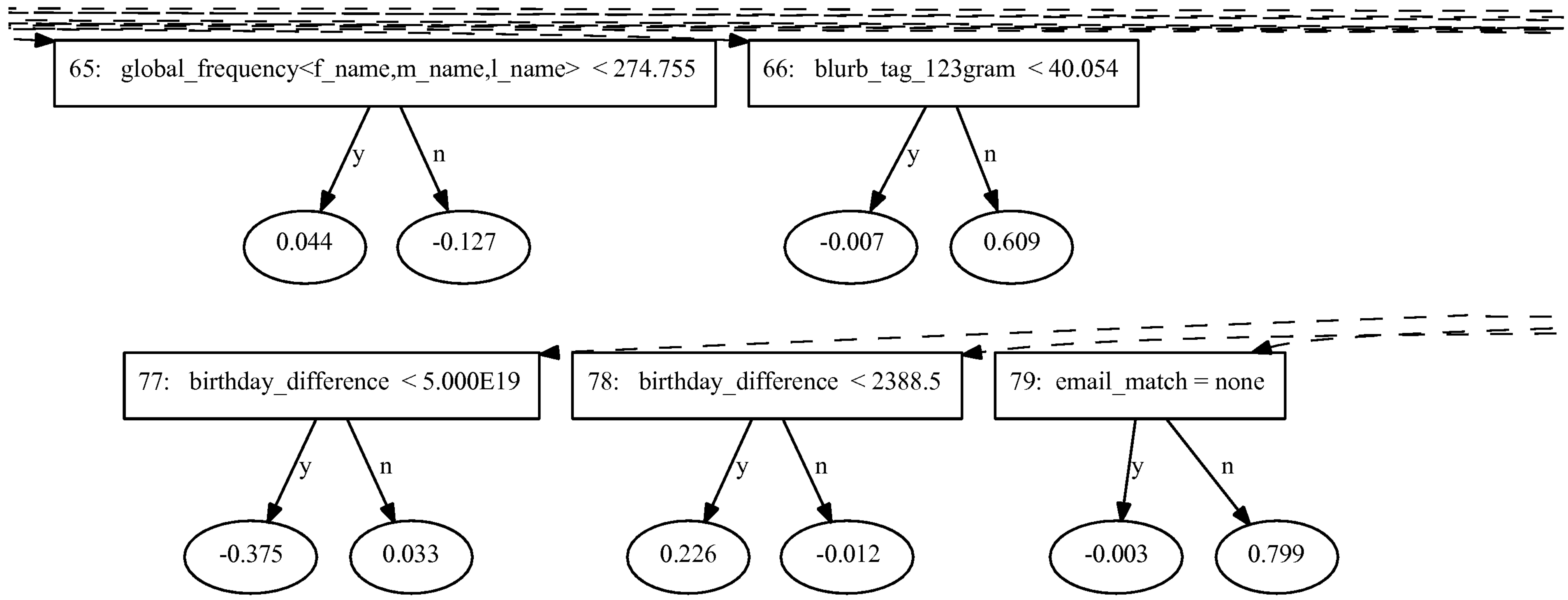

65: global_frequency<f_name,m_name,l_name> < 274.755
y
n
0.044
-0.127
66: blurb_tag_123gram < 40.054
y
n
-0.007
0.609
77: birthday_difference < 5.000E19
y
n
-0.375
0.033
78: birthday_difference < 2388.5
y
n
0.226
-0.012
79: email_match = none
y
n
-0.003
0.799

FIG. 17N

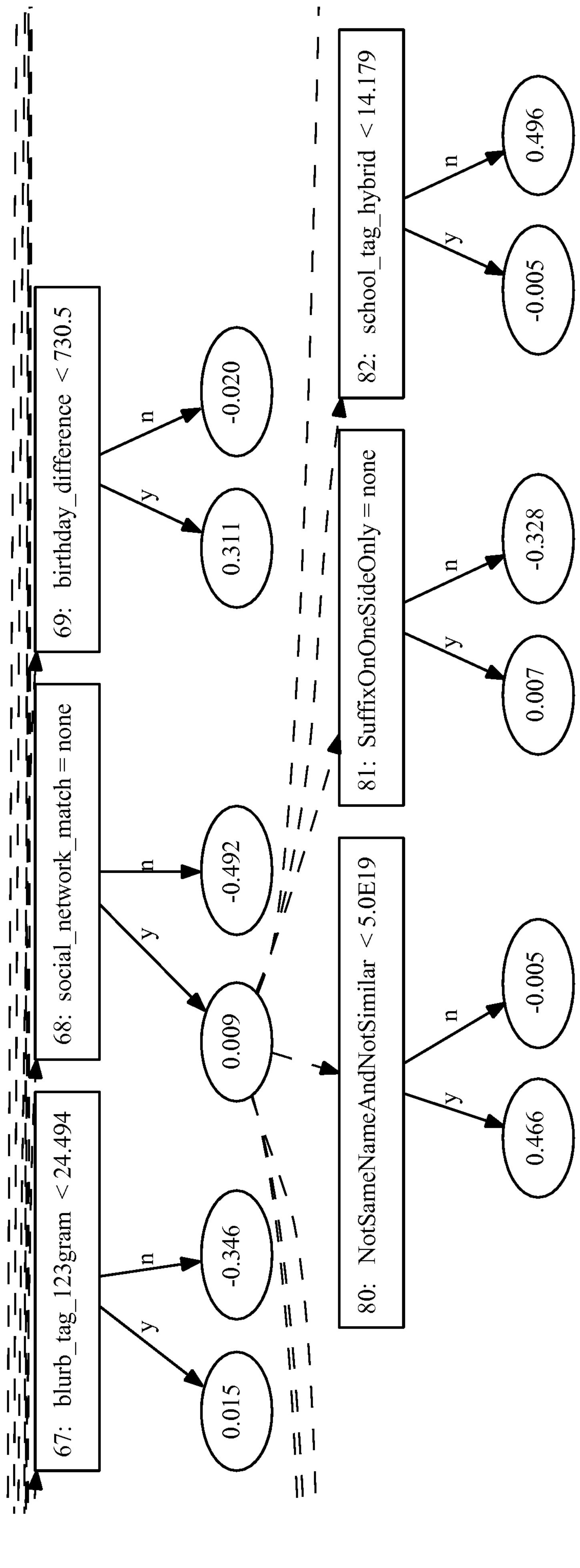

67: blurb_tag_123gram < 24.494
y
n
0.015
-0.346
68: social_network_match = none
y
n
0.009
-0.492
69: birthday_difference < 730.5
y
n
0.311
-0.020
80: NotSameNameAndNotSimilar < 5.0E19
y
n
0.466
-0.005
81: SuffixOnOneSideOnly = none
y
n
0.007
-0.328
82: school_tag_hybrid < 14.179
y
n
-0.005
0.496

FIG. 17O

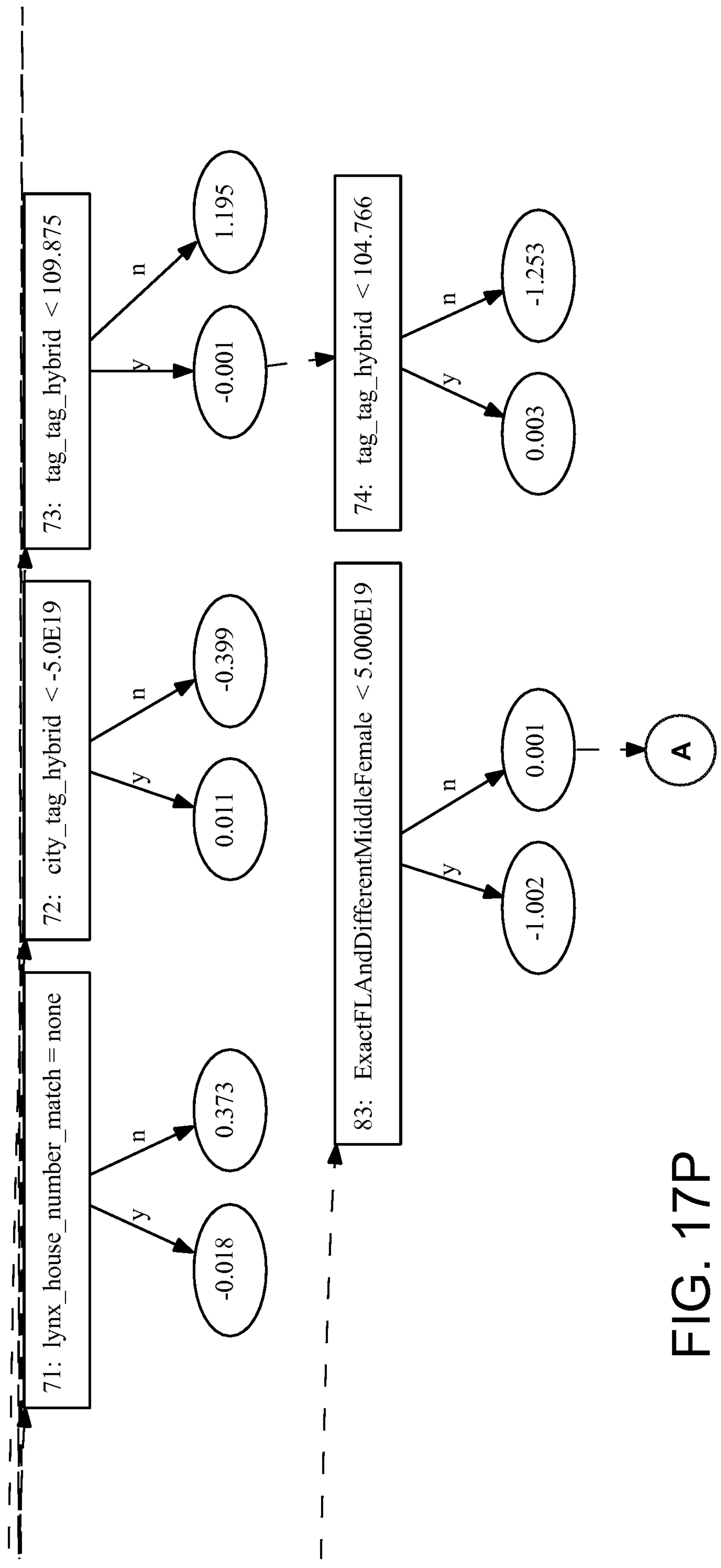

71: lynx_house_number_match = none
y
n
-0.018
0.373
72: city_tag_hybrid < -5.0E19
y
n
0.011
-0.399
73: tag_tag_hybrid < 109.875
y
n
-0.001
1.195
83: ExactFLAndDifferentMiddleFemale < 5.000E19
y
n
-1.002
0.001
A
74: tag_tag_hybrid < 104.766
y
n
0.003
-1.253

FIG. 17P

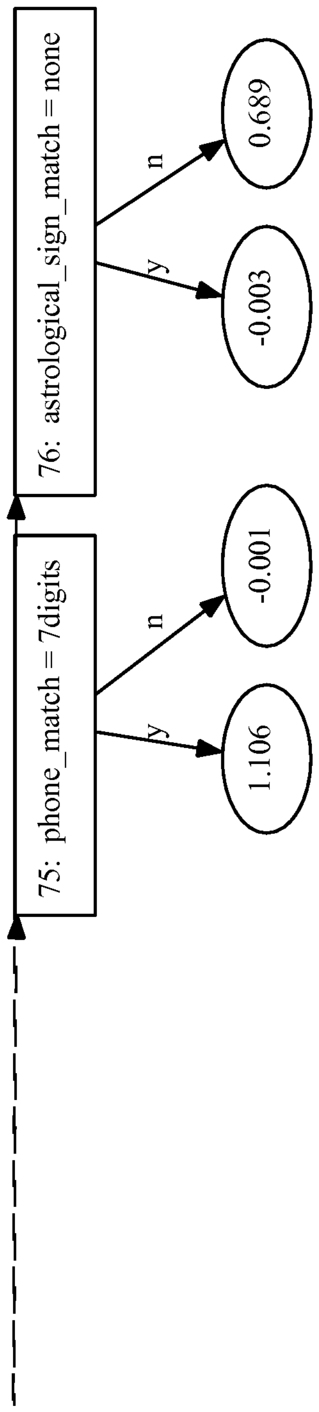

75: phone_match = 7digits
y
n
1.106
-0.001
76: astrological_sign_match = none
y
n
-0.003
0.689

FIG. 17Q

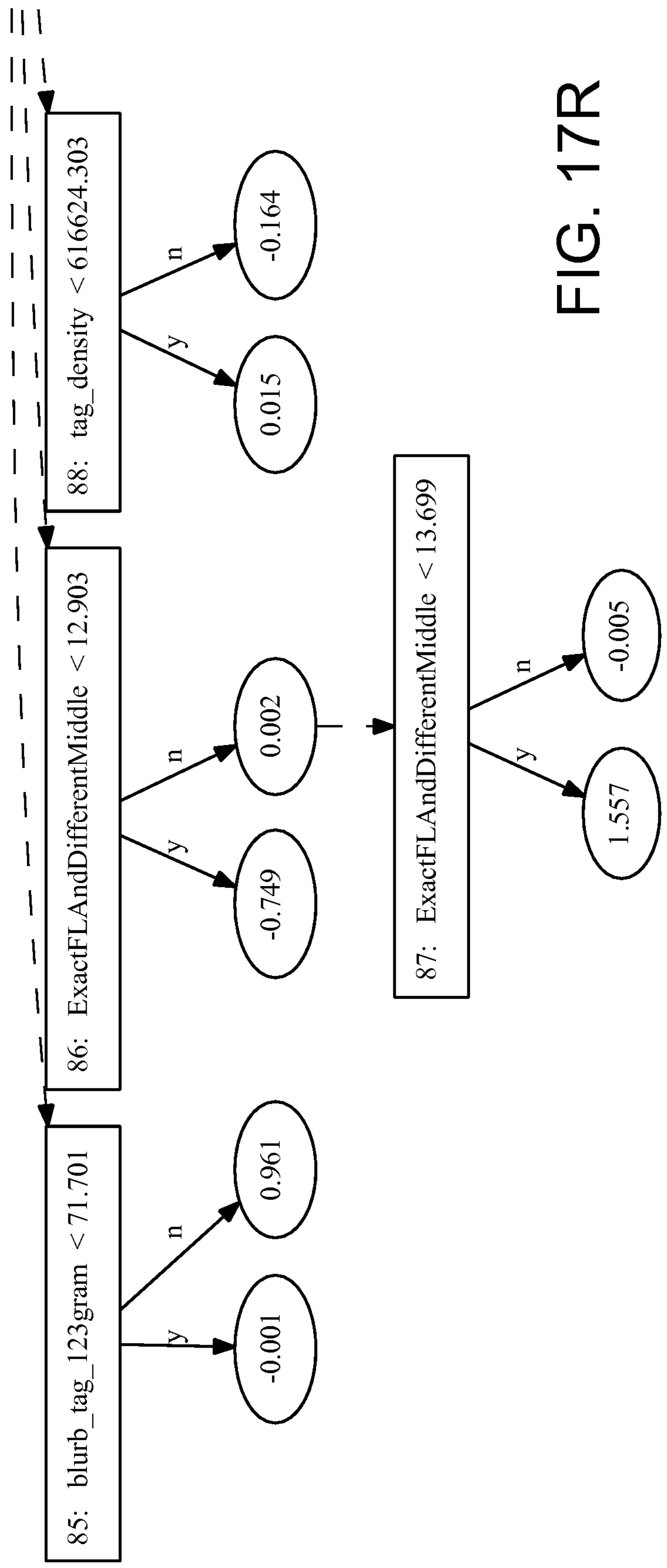

88: tag_density < 616624.303
n
-0.164
y
0.015
86: ExactFLAndDifferentMiddle < 12.903
n
0.002
y
-0.749
87: ExactFLAndDifferentMiddle < 13.699
n
-0.005
y
1.557
85: blurb_tag_123gram < 71.701
n
0.961
y
-0.001

FIG. 17R

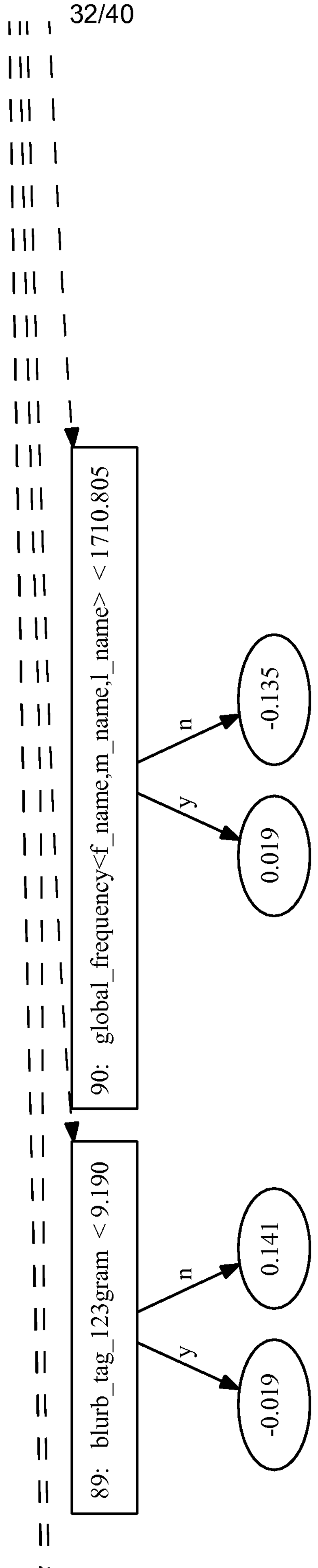

90: global_frequency<f_name,m_name,l_name> < 1710.805
n
-0.135
y
0.019
89: blurb_tag_123gram < 9.190
n
0.141
y
-0.019

FIG. 17S

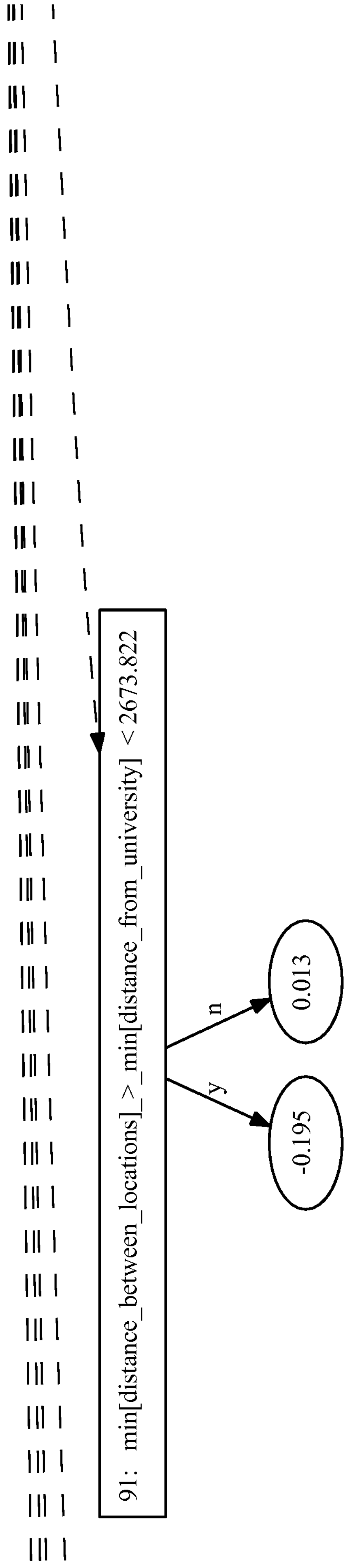

91: min[distance_between_locations] >_min[distance_from_university] <2673.822
n
y
0.013
-0.195

FIG. 17T

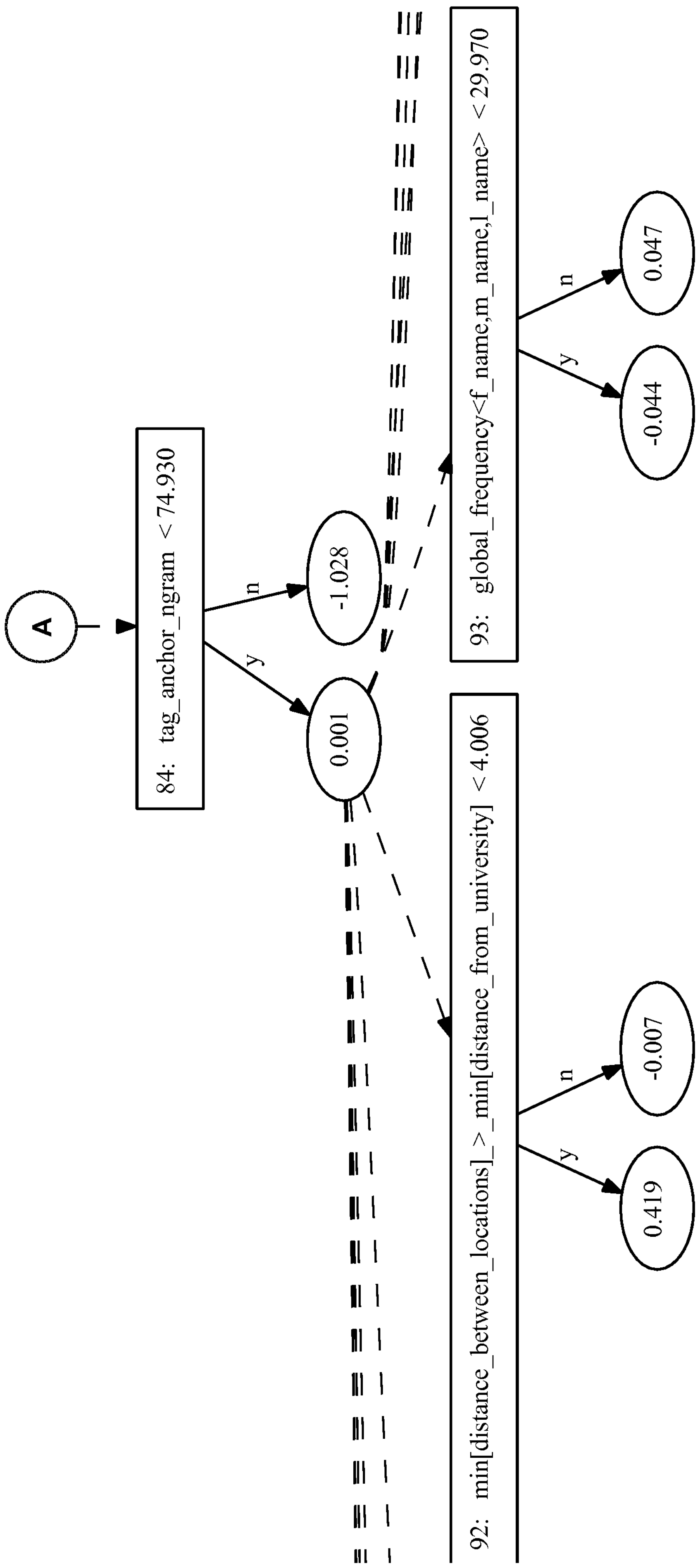

A
84: tag_anchor_ngram < 74.930
n
y
-1.028
0.001
93: global_frequency<f_name,m_name,l_name> < 29.970
n
y
0.047
-0.044
92: min[distance_between_locations]_>_min[distance_from_university] < 4.006
n
y
-0.007
0.419

FIG. 17U

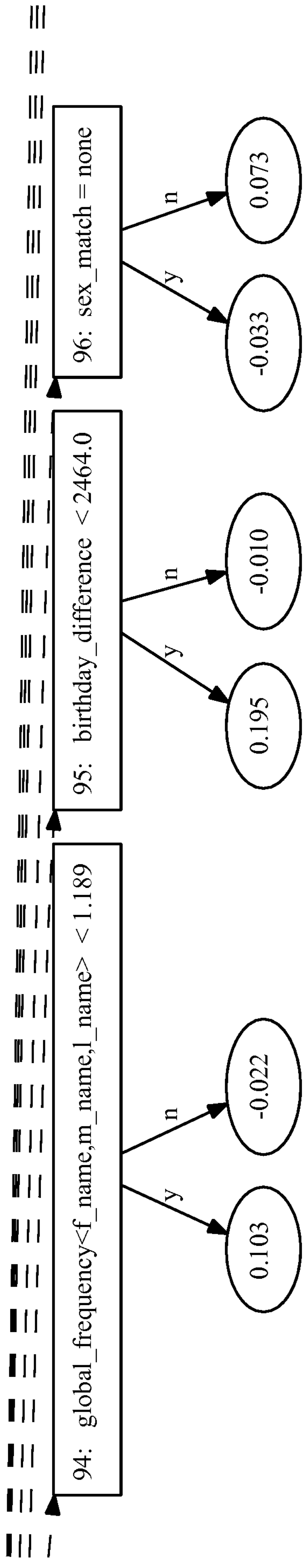

94: global_frequency<f_name,m_name,l_name> < 1.189
y
n
0.103
-0.022
95: birthday_difference < 2464.0
y
n
0.195
-0.010
96: sex_match = none
y
n
-0.033
0.073

FIG. 17V

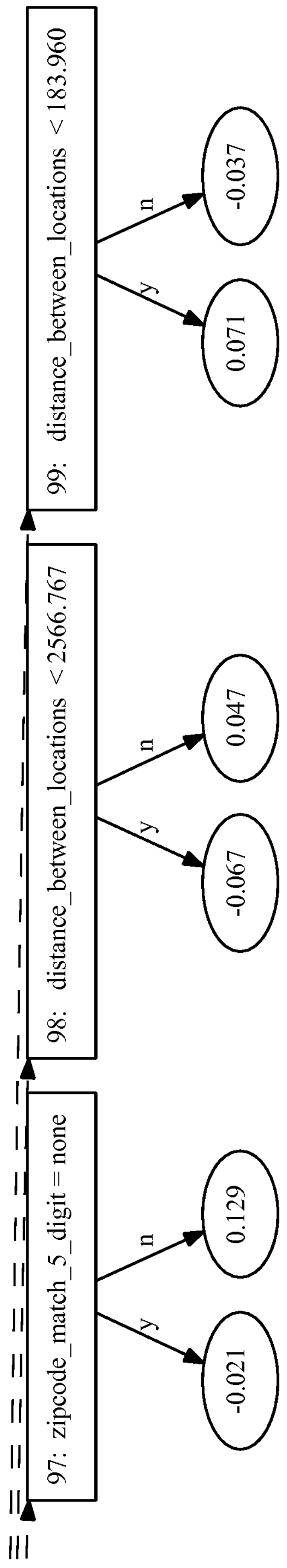

97: zipcode_match_5_digit = none
y
n
-0.021
0.129
98: distance_between_locations < 2566.767
y
n
-0.067
0.047
99: distance_between_locations < 183.960
y
n
0.071
-0.037

FIG. 17W

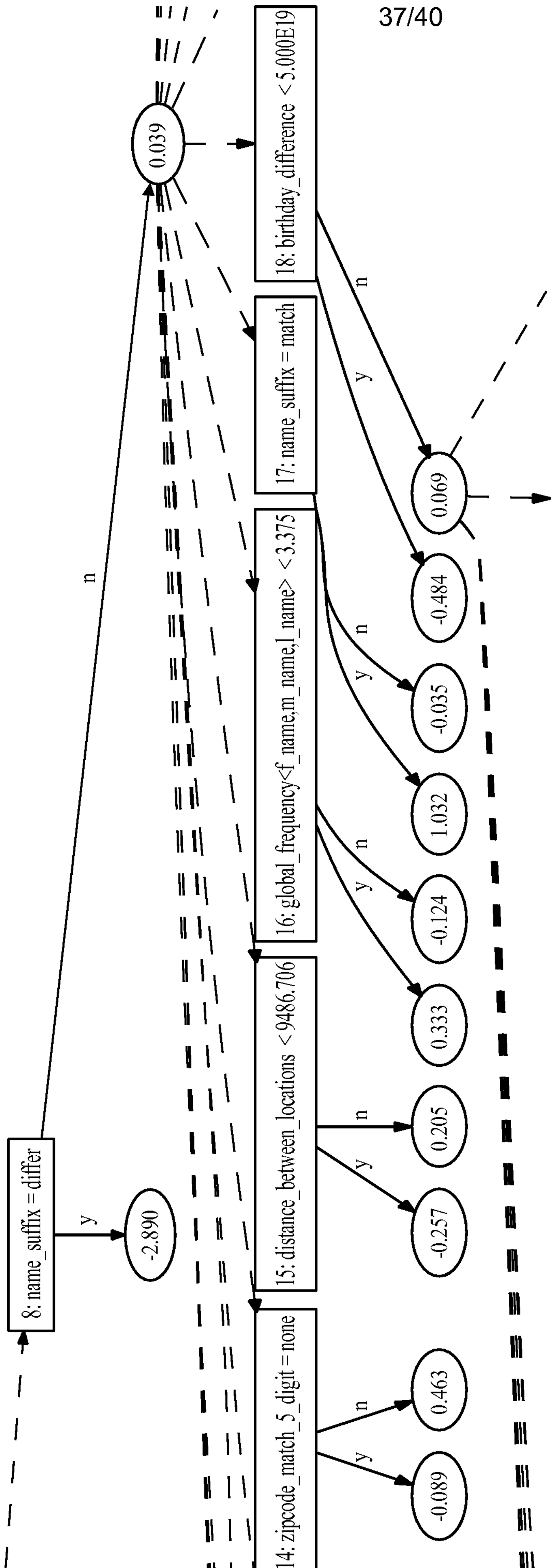

8: name_suffix = differ
y
-2.890
n
0.039
18: birthday_difference < 5.000E19
17: name_suffix = match
16: global_frequency<f_name,m_name,l_name> < 3.375
15: distance_between_locations < 9486.706
14: zipcode_match_5_digit = none
n
y
0.069
-0.484
n
y
-0.035
1.032
n
y
-0.124
0.333
n
y
0.205
-0.257
n
y
0.463
-0.089

FIG. 17X

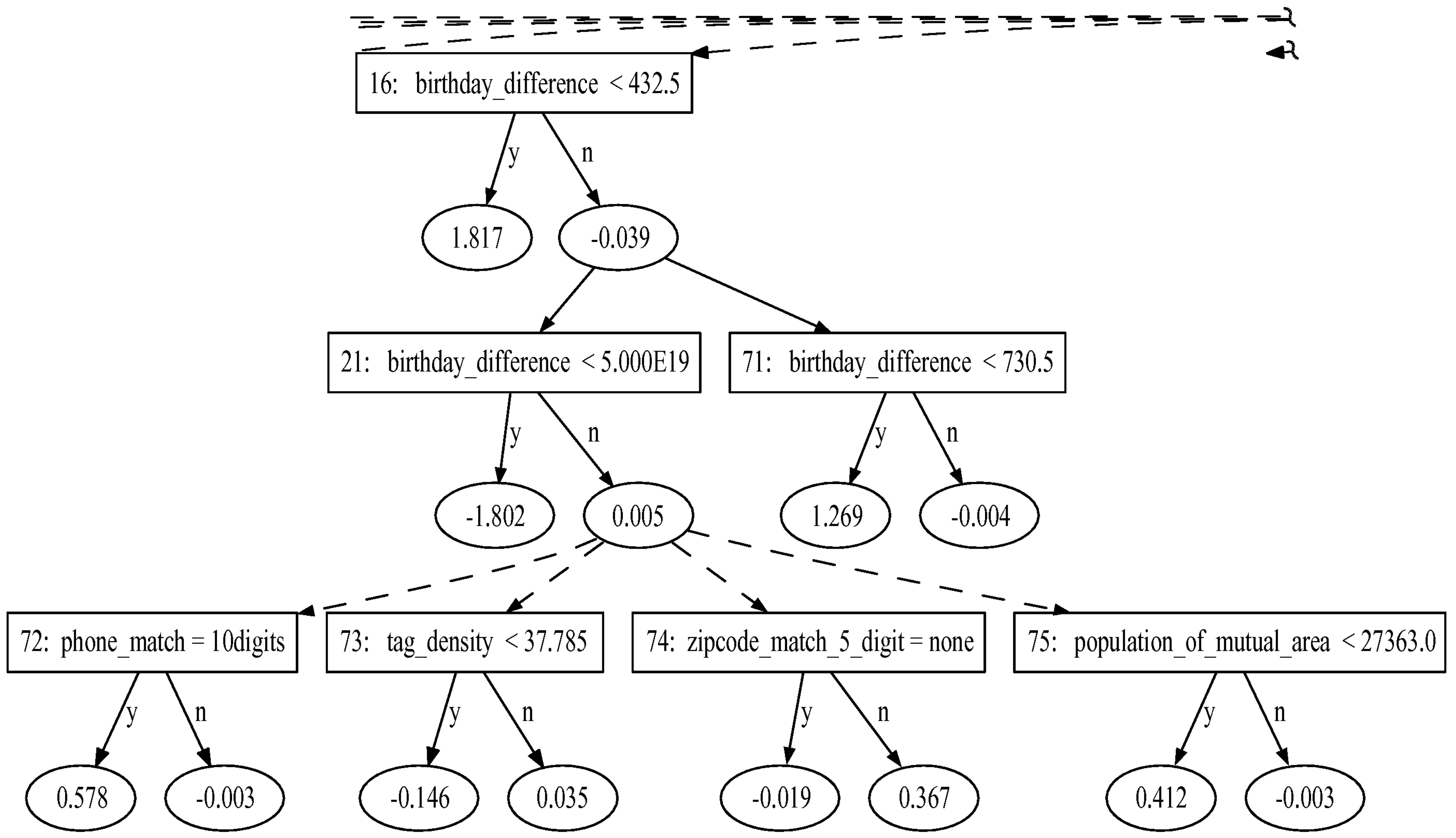

16: birthday_difference < 432.5
y
n
1.817
-0.039
21: birthday_difference < 5.000E19
71: birthday_difference < 730.5
y
n
-1.802
0.005
y
n
1.269
-0.004
72: phone_match = 10digits
73: tag_density < 37.785
74: zipcode_match_5_digit = none
75: population_of_mutual_area < 27363.0
y
n
0.578
-0.003
y
n
-0.146
0.035
y
n
-0.019
0.367
y
n
0.412
-0.003

FIG. 18

Partial CS-ADTree. Note how we assign different scores to different birthday distances and query additional features if birthday difference is unknown (infinite).

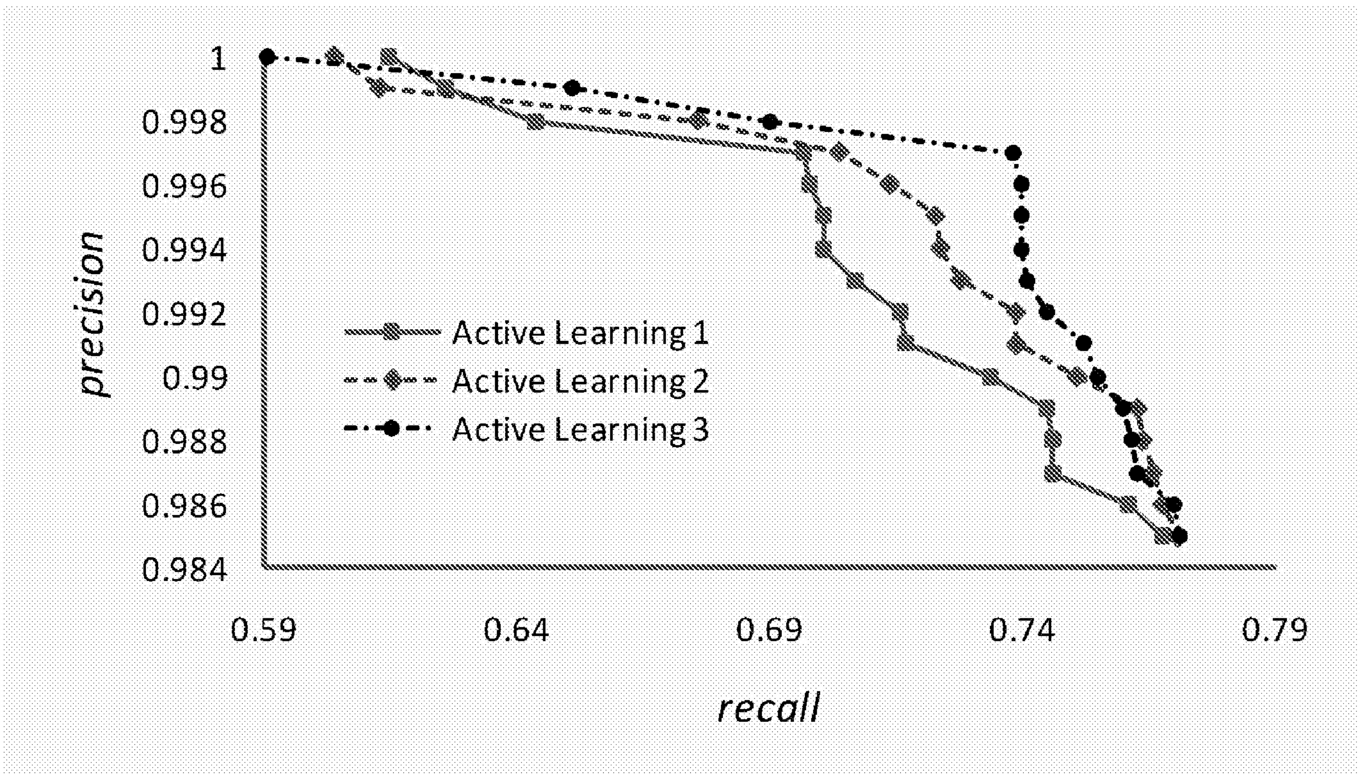


FIG. 19

P-R curves of 3 active learning approaches

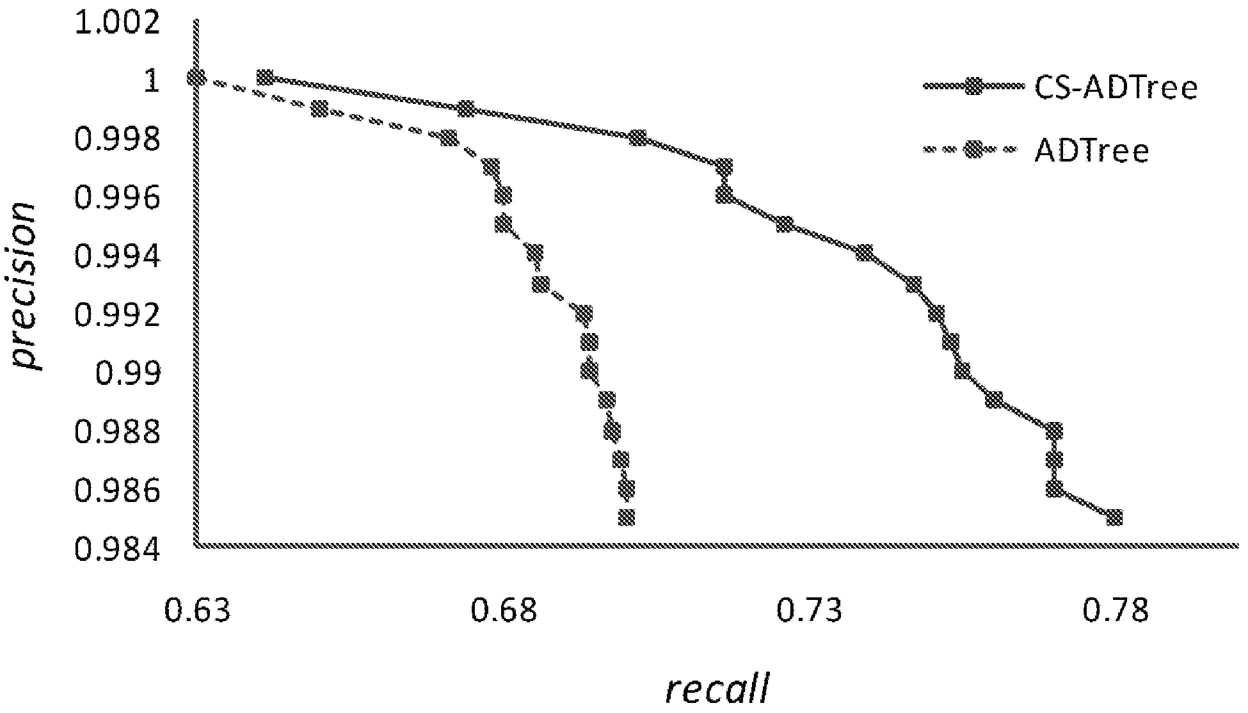


FIG. 20

P-R curves of ADTree and CS-ADTree on *E*

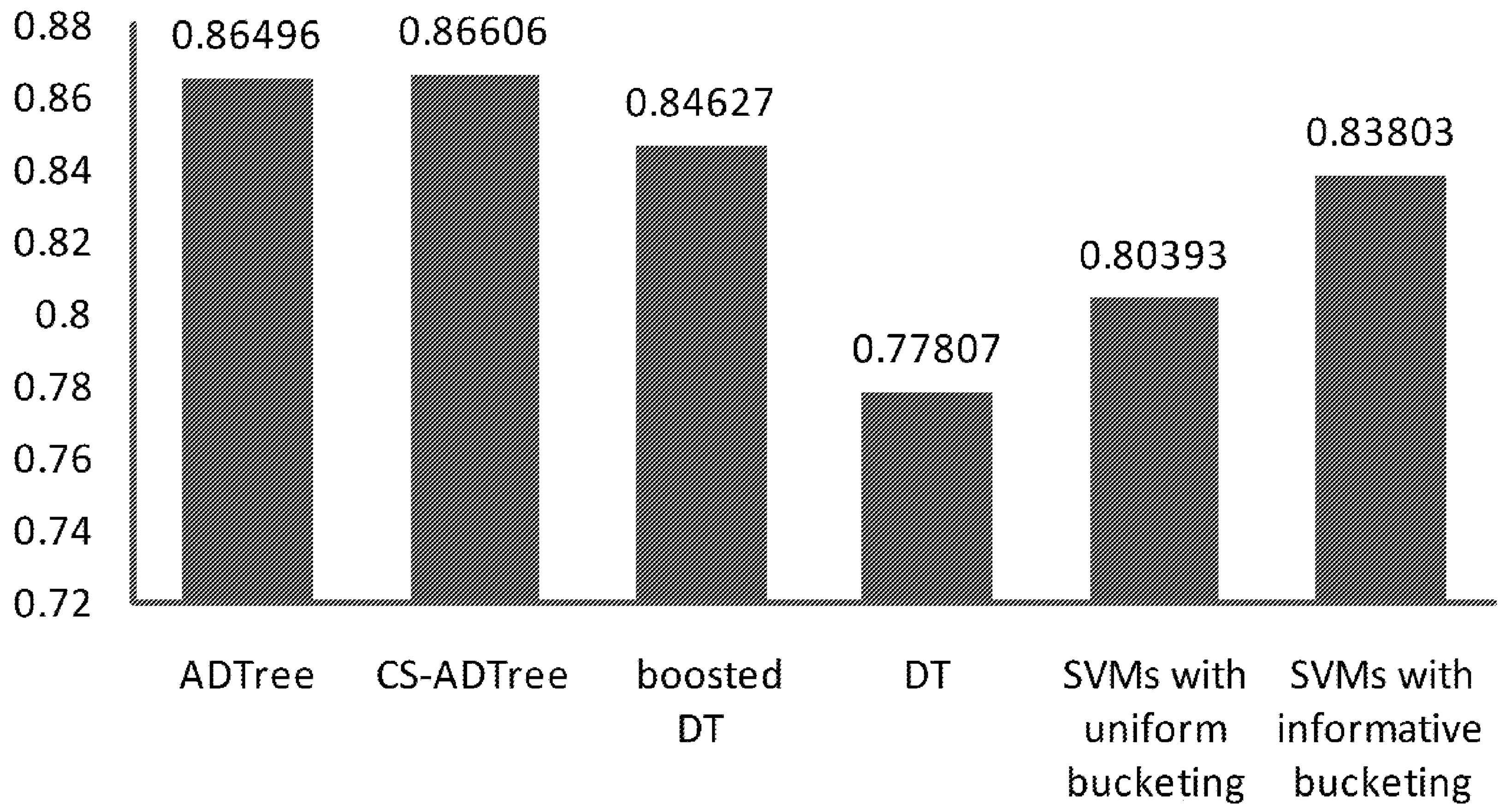


FIG. 21

Histogram of’s for ADTree, CS-ADTree, and alternative ML techniques on *E*

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2011/057757

A. CLASSIFICATION OF SUBJECT MATTER

G06F 17/30(2006.01)i, G06F 17/40(2006.01)i, G06F 17/00(2006.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC : G06F; H04L; G06N

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
Korean utility models and applications for utility models
Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
eKOMPASS(KIPO internal) & Keywords: ADtree*,AD tree*, decision tree*,alternative tree*,block*,segme*,fragme*,classif*,parti*,,learning model*,machine lear*

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2007-0294187 A1 (CHAD SCHERRER) 20 December 2007 See the abstract; paragraphs[0041]-[0047],[0081],[0100]; figures 3,4,9	1-8,11-16,18-20
A	US 7379926 B1 (BELNIAK MACGREGOR et al.) 27 May 2008 See the abstract; column 5, line 48- column 16, line 46; claim 1; figure 3	1-20
A	US 2010-0070457 A1 (KEJARIWAL ARUN et al.) 18 March 2010 See the abstract; paragraphs[0024]-[0026],[0056]; figures 1,2	1-20
A	US 2004-0019593 A1 (ANDREW E. BORTHWICK et al.) 29 January 2004 See the abstract; paragraphs[0019]-[0025]; figures 2,4	1-20

☐ Further documents are listed in the continuation of Box C. ☒ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents,such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search 28 MARCH 2012 (28.03.2012)	Date of mailing of the international search report **10 APRIL 2012 (10.04.2012)**
Name and mailing address of the ISA/KR Korean Intellectual Property Office Government Complex-Daejeon, 189 Cheongsa-ro, Seo-gu, Daejeon 302-701, Republic of Korea Facsimile No. 82-42-472-7140	Authorized officer Ko Jae Yong Telephone No. 82-42-481-8212

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/US2011/057757

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2007-0294187 A1	20.12.2007	US 7739082 B2	15.06.2010
US 7379926 B1	27.05.2008	None	
US 2010-0070457 A1	18.03.2010	None	
US 2004-0019593 A1	29.01.2004	US 7152060 B2	19.12.2006