Adding Local Weather Simulation to the BigPetStore Data Generator for the JVM

Ronald J. Nowling
Red Hat, Inc.
Raleigh, NC 27601
Email: rnowling@redhat.com

Jay Vyas
Red Hat, Inc.
Raleigh, NC 27601
Email: jvyas@redhat.com

***Abstract*—Generating large amounts of semantically-rich data for testing big data workflows is paramount for scalable performance benchmarking and quality assurance in modern machine-learning and analytics workloads. The most obvious use case for such a generative algorithm is in conjunction with a big data application blueprint, which can be used by developers (to test their emerging big data solutions) as well as end users (as a starting point for validating infrastructure installations, building novel applications, and learning analytics methods).**

We present a new domain-driven, generative data model for BigPetStore, a big data application blueprint for the Hadoop ecosystem included in the Apache BigTop distribution. We describe the model and demonstrate its ability to generate semantically-rich data at variable scale ranging from a single machine to a large cluster. We validate the model by using the generated data to answer questions about customer locations and purchasing habits for a fictional targeted advertising campaign, a common business use case.

Keywords*—*big data, synthetic data sets, data generation, benchmarking, testing, probabilistic models

I. Introduction

Big data applications process large, dynamic, multidimensional data sets with the general goal of information and knowledge extraction. With the wide variety of big data tools available and lagging documentation, both developers and users of big data systems benefit from the availability of realistic example applications. For developers, such applications are useful for testing, benchmarking, and evaluating design choices; for users, example applications provide starting points for learning methods, developing their own applications, and implementing new analytics workflows.

BigPetStore is one such big data application blueprint built around processing transaction data for a fictional chain of pet stores. BigPetStore targets the Hadoop [1] ecosystem with examples implemented for loading, cleaning, aggregating, and performing analytics on data using Hive [2], Pig [3], [4], and Mahout [5]. BigPetStore has been incorporated into the open-source Apache BigTop distribution [6], where it is used for testing and as a reference application. One of BigPetStore's key features is the ability to scale from a single machine to a large cluster, making it easy to develop projects on a local machine and transfer the application to a large cluster for production testing.

To achieve the project's goals of providing high-quality, realistic examples, BigPetStore requires semantically-rich, complex data. At its core, BigPetStore relies on a generative data model for producing synthetic transaction data. Despite the growing number of real data sets now publically available, synthetic data generators have a number of advantages for BigPetStore over real data sets. The synthetic data generator can be packaged with BigPetStore, avoiding the cost, time, and infrastructure needed to host, transfer, and store large data sets. Synthetic data generators are scalable, allowing the user to choose how much data to generate – a requirement for supporting BigPetStore's goal of running on both single machines and large clusters. Licensing and privacy issues are avoided with the use of synthetic data sets. And lastly, the generated data can be customized by the user, allowing the user to generate data with specific criteria amenable to testing. For example, a user may generate transactions from a small number of purchasable items so that clustering results can be easily visualized.

A variety of approaches for generating synthetic data sets exist. TeraGen and the Intel Hadoop Benchmark Suite [7] are popular tools that can generate data sets quickly and at any scale, but the resulting data is semantically-void and not useful for much more than simple benchmarks. Multiple frameworks exist for generating synthetic data sets that satisfy relational database schemas [8]–[14], and several approaches even provide domain-specific languages [13]–[15] for specifying additional constraints (such as which distributions to sample from and allow for modeling basic relationships between fields using simple arithmetic equations [16]).

Recent work has addressed some aspects of the need for more dynamic data set generation: Arasu, et al. [17] demonstrated that constraint-solving techniques could be used as an alternative to procedural approaches. Such frameworks allow for reproducing the structural properties of real data, but the frameworks are not expressive enough to describe the dynamic generation processes and latent variables that would be needed to embed the desired informational complexity and rich semantics needed for BigPetStore's analytics examples.

Realizing the difficulty in creating a generic framework capable of modeling the semantics of real data, recent work [18] has focused on generating synthetic data sets that satisfy characteristics learned from real reference data. Such approaches appear promising but will need to overcome the difficulties of accurately training models, especially on raw data instead of features.

Until such purely generic methods reach maturity, BigPetStore's needs are best met by a customized, domain-

driven model. Another example of such a model is The Information Discovery and Analysis System (IDAS) Data Set Generator (IDSG) [19], [20]. IDSG was developed to test the effectiveness of IDASs in identifying potential terrorism threat scenarios using synthetic data. Like BigPetStore, the developers of IDSG needed to avoid licensing restrictions and privacy issues associated with real data.

BigPetStore's current model has been used successfully to generate terabytes of synthetic data and is used regularly for testing in Apache BigTop, showcasing the value of the approach. The current model is limited, however, in its ability to generate data that is semantically rich, limiting progress on BigPetStore's goals of providing realistic analytics examples.

In this work, we present a new domain-driven model and simulation method for BigPetStore. Compared with BigPetStore's previous data generator, our model and implementation can generate data which contains geospatial, temporal, and quantitative features, similar to the type of data which businesses and organizations might typically encounter. Combining the features of the TeraGen (scalability) and MovieLens [21] (semantically rich) input data sets commonly used for big data benchmarking, we present a synthetic data set generator which can be used to benchmark lower level tasks (such as sorting) as well as higher level tasks (such as clustering, recommending, and search indexing) at arbitrary scale. To demonstrate that the model generates data with desirable properties, we performed an example analysis designed to guide a fictional advertising campaign (a typical business use case). The model's implementation was made available as open source software.

II. Overview of Previous Model

Our generative model combines various well-known mathematical modeling techniques to simulate the factors affecting customers' purchasing habits. Each part of the model is derived from *ab initio* assumptions. In several cases, real data is used to parameterize parts of the model. We described the initial version of the model in [22], which we summarize here.

The model generates data for four types of records: stores, customers, transactions, and transaction items (Figure 1). Store records contain both a unique identifier and a location in the form of a 5-digit zip code. Customer records consist of a unique identifier, a name, and a location given as a 5-digit zip code. Transaction records consist of a transaction identifier, a customer identifier, a store identifier, and a time given in days since the beginning of the simulation. Transaction item records contain a transaction identifier and an item description. The item description is a list of key-value pairs stored as a string. Key-value pairs are used since item characteristics differ depending on the type of item. For example, dog food has a brand, a flavor, a size (in lbs), and a per-unit cost, while poop bags have a brand, a color, a count, and a per-unit cost.

A. Stores

Store records consist of unqiue integer identifiers and locations given as zip codes. The stores' zip codes are sampled from a probability mass function (PMF) where probabilities are linearly proportional to the zip code's population and exponentialy proportional to the zip code's median household income. The population and household income data for the zip codes were taken from from the U.S. Census American Community Survey [23].

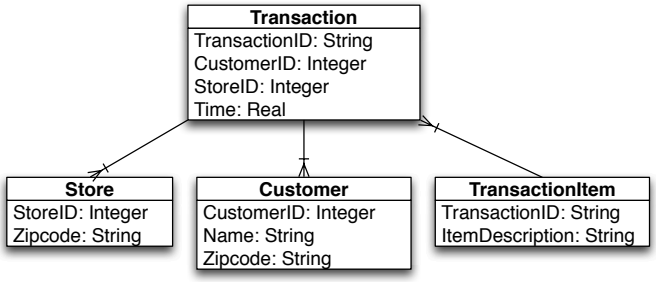


Fig. 1: Relational Data Model for Generated Data

B. Customers

Customer records consist of uniquer integer identifers, first names, last names, and locations given as zipcodes. Names are generated using data from the Name Database [24]. Each record in the database gives a name, a weight, and flag indicatings if the name can be used as a first name, a last name, or both. PMFs generated for the first and last names using the weights. The customer's name is generated by sampling a first name and a last name from each PMF respectively. Zip codes are sampled from a PMF where the probabilities are exponentially distributed according to the distance (in miles) to the nearest store's location. Distances are calculated using latitude and longitude coordinates taken from the Zip Code Database Project [25]).

C. Transactions

Transactions records consist of customer, transaction, and store identifiers, the date and time of the transaction (as a real number indicating the number of days since the start of the simulation), and a list of purchased products. The customer and transaction identifiers form a unique composite key for each transaction record.

Transactions are simulated individually for each customer. Transaction times are sampled using a Monte Carlo process. Transaction times are proposed by sampling offsets before the exhaustion of products from an exponential distribution, with the average time parameterized separately for each customer by sampling from a uniform interval. Acceptance probabilities are given by a separate PDF which can be used to model the effect of with factors such as store hours and customer work schedules. Currently, the acceptance PDF only ensures that a new transaction time occurs after the previous transaction. The proposed transaction time is accepted if a random number sampled uniformally from $[0, 1)$ is less than the acceptance probability. Otherwise, a new transaction time is proposed.

Product purchases are simulated by a separate process. First, a state is chosen from the set of product categories and a stop state which indicates the end of the transaction. Each product category's weight is exponentially proportional to the elapsed time between the transaction and exhaustion of the product category. The weight of the stop state is given by a constant which is zero until the first product is chosen.

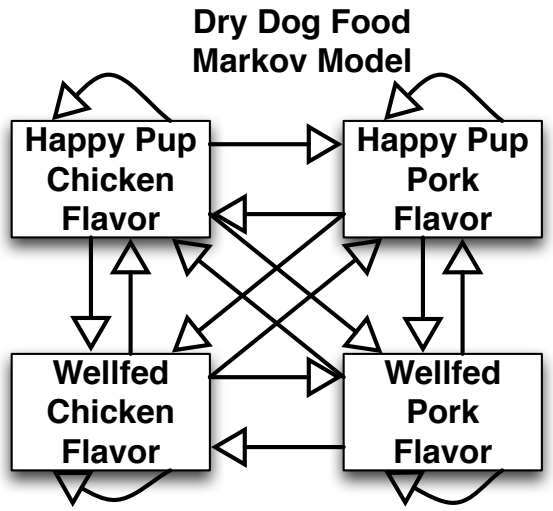


Fig. 2: Example Markov Model for Dog Food

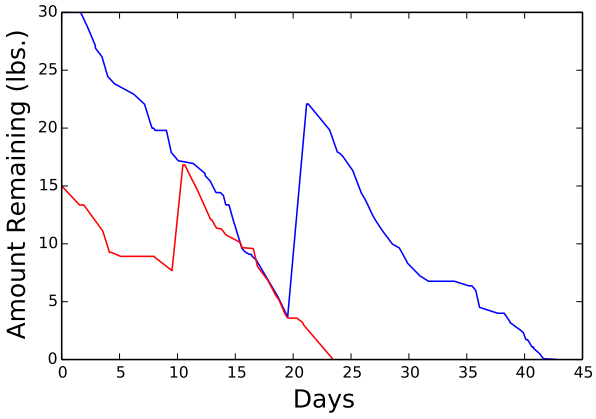


Fig. 3: Example Product Usage Simulation

Once a product category has been chosen, the process of choosing a product within the category is modelled by a Markov Model. The transition probabilites are determined by ..... A product is chosen by advancing the state of the model by simulate one transition. The state of the Markov Model is preserved between transactions.

Once a product has been chosen, the customer's inventory is updated and the exhaustion time is determined by simulating the usage of the products by numerically integrating a stochastic differential equation (SDE). The purchasing process is restarted, using the updated exhaustion time. If the stop state is chosen, the next transaction is generated.

III. Improvements to the Data Generator

A. Products

The original data generator relied on a hand-generated JSON document containing product descriptions. Due to the labor requirements, only a handful of products were available in each category with only 3-4 fields per category. To increase the number and variety of purchasing patterns in the model, a product enumerator was implemented. Given a set of fields and possible values, the enumerator generates all combinations of products.

Rules are used to filter out unwanted combinations of field values. For example, the fictional brand of dog food "Chef Corgi" might be targeted towards budget-oriented consumers. As such, the brands' products might exclude expensive ingredients and marketing (salmon, organic ingredients, etc.). Products that do not satisfy the filters are removed from the list of possible products.

Using the product enumerator, the number of products across four categories has been increased from 40 to over 3,000, and the number of fields has increased from 3-4 to upwards of 10 per category.

Optional products were introduced. Examples of optional products include snacks, collars, sweaters, and toys. Exhaustion of optional products does not trigger transactions. Said another way, the optional products' exhaustion times are not used when proposing transaction times. There is no distinction between optional and mandatory products when simulating purchases within a transacton.

weather-triggered or seasonal products. product PDFs?

B. Customer and Transaction Stores

In the data generator prototype, the store for each each transaction was picked uniformally from the set of all stores. To improve realism and support the incorporation of geospatial patterns, each customer is now associated with a specific store and all of a customer's transactions occur at that store.

Customers' associated stores are sampled from a PMF with weights proportionally to population of each stores' zip code. The model for sampling the customers' locations was updated so that each zip code z is weighted exponentially according to the distance from the zip code s of the customers' associated store:

$$w(s, z) = \lambda \exp(-\lambda d(s, z)) \tag{1}$$

where λ is the inverse desired average distance.

C. Weather Model

TODO why?

Weather data is generated using a local weather model – i.e., weather is not explicitly correlated between regions – so that each region can be simulated independently for computational efficiency. The model was derived and evaluated by comparing to local weather data downloaded from the NOAA QCLCD cite. The goal of the model was not to accurately predict day-to-day weather patterns but to reproduce temporal and static statistical properties. The model is similar other models available in the literature cite.

The models were derived for daily average temperatures ($T(t)$, °F), daily total precipitation ($P(t)$, in), and daily average wind speed ($V(t)$, mph). The temperature for each

location is modeled as a first-order Fourier series plus noise generated by an Ornstein-Uhlenbeck process $Z(t)$ cite:

$$T(t) = a_0 + a_1 \sin\left(\frac{-2\pi t}{365}\right) + a_2 \cos\left(\frac{-2\pi t}{365}\right) + Z(t) \quad (2)$$

$$dZ(t) = -\gamma Z(t)dt + \sigma dW_t \quad (3)$$

The stochastic differential equation (SDE) for the Ornstein-Uhlenbeck process is integrated numerically using the Euler-Maruyama method [26]:

$$Z_{t+1} = -\gamma Z_t \Delta t + \sigma\sqrt{\Delta t}\Delta W_{t+1} \quad (4)$$
$$Z_0 = 0$$

The daily total precipitation for each region is assumed to be independently distributed from day-to-day and sampled from an exponential distribution:

$$P(t) \sim Exp(\lambda) \quad (5)$$

The daily average wind speed for each region is modeled using a first-order Fourier series and stochastic equation:

$$V(t) = b_0 + b_1 \sin\left(\frac{-2\pi t}{365}\right) + b_2 \cos\left(\frac{-2\pi t}{365}\right) + X(t) \quad (6)$$

$$X(t) \sim Erlang(k, \theta) \quad (7)$$

The average temperature a_0, a_1, a_2, inverse daily average precipitation λ, average daily wind speed b_0, b_1, b_2, k, and θ are fit to the observed data (how?) from NOAA QCLCD for each zip code. The damping coefficient γ is determined empirically.

Three years of observed daily average temperature, total precipitation, and average wind speed from the NOAA QCLCD data for South Bend, IN from October 2011 to September 2014 are shown in Figure 4 alongside simulated data from the model. The simulated data is able to produce temporal and stationary patterns similar to those of the real data.

From the temperature, precipitation and wind speed, the snowfall ($S(t)$, in), rainfall ($R(T)$, in), and wind chill ($C(T)$, °F) cite are calculated as follows:

$$r(t) = \frac{1}{1 + \exp(-a(T - b))} \quad (8)$$

$$S(t) = 10(1 - r(t))P(T) \quad (9)$$

$$R(T) = r(t)P(T) \quad (10)$$

$$C(T) = 35.74 + 0.6215T(t) - 35.75V^{0.16}(t) + 0.4275T(t)V^{0.16}(t) \quad (11)$$

The snowfall, rainfall, and wind chill are used as input to a conditional probability density function (PDF) giving the probability of a transaction occuring on the date of time t given the weather conditions:

$$P(\text{trans}|t, S(T), R(T), C(T)) = \ldots \quad (12)$$

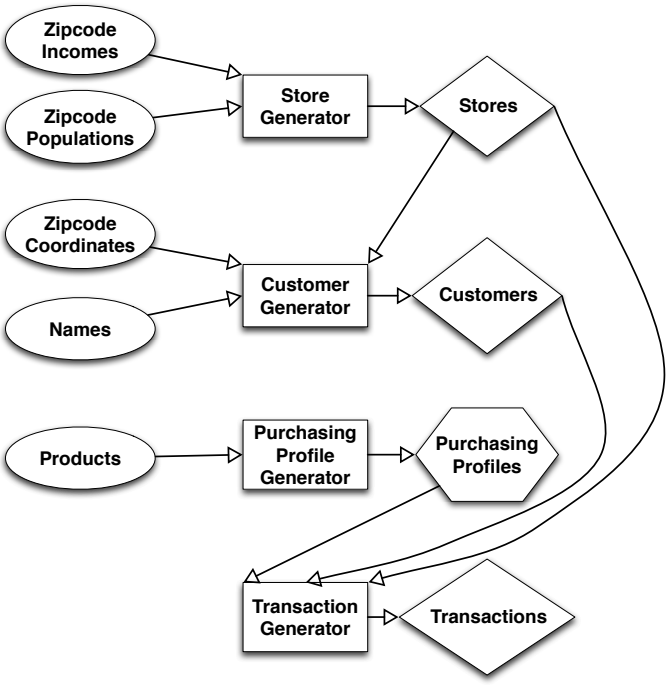
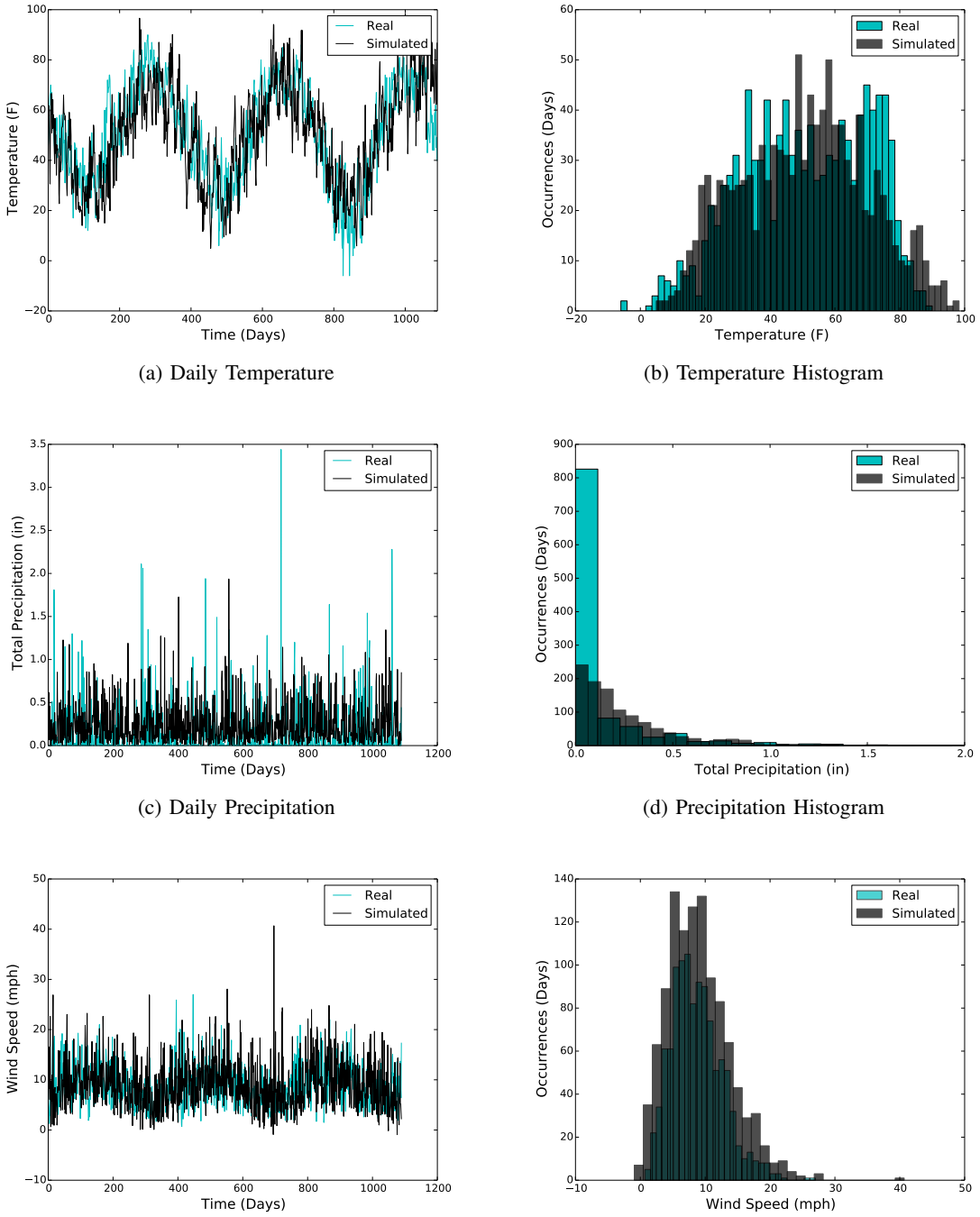


Fig. 5: BigPetStore Flowchart

D. Implementation

The data generator was previously prototyped in Python. Once the utility of the generator was validated, a new production implementation of the data generator was implemented and released as a Java library. Java was chosen for compatibility with popular big data processing systems such as Apache Hadoop cite and Apache Spark cite and other JVM languages such as Scala and Clojure. The Java implementation has 5,500 lines of code, 110 classes, and 90 unit tests.

Users interact with the library through six primary classes (`DataLoader`, `StoreGenerator`, `CustomerGenerator`, `ProductGenerator`, `PurchasingProfileGenerator`, and `TransactionGenerator`) and five data model POJOs (`InputData`, `Store`, `Customer`, `Product`, and `Transaction`). Broad support for multiple parallelization paradigms was a primary design requirement. Generators interact only through standard Java collections containing serializable POJOs and are designed to support multiple instantiated copies to support parallelization. As such, parallelization is as simple as creating as many independent instances as needed and combining the resulting Java collections before each stage. An example driver implemented with Apache Spark with support for parallel and distributed computing is presented in Section IV. For development purposes, the library includes a single-threaded command-line driver.



(a) Daily Temperature

(b) Temperature Histogram

(c) Daily Precipitation

(d) Precipitation Histogram

(e) Daily Wind Speed

(f) Wind Speed Histogram

Fig. 4: NOAA QCLCD weather data alongside simulated data for South Bend, IN for October 2011 to September 2014.

The source code is available at https://github.com/rnowling/bigpetstore-data-generator under the Apache Public License v2. Jars are published via bintray for easy integration with build systems that support automatic dependency resolution.

IV. Parallelization with Spark Driver

TODO code listing

V. Evaluation of the Model

To evaluate the semantic content of the generated data, we performed two example analyses in support of a fictional advertising campaign using data for 10 stores, 10,000 customers, and five years of transactions generated from the model. The analyses were designed to be realistic and driven by real-world business concerns. Due to limited advertising

Fig. 6: Distributions of Distances of Customers to Closest Stores

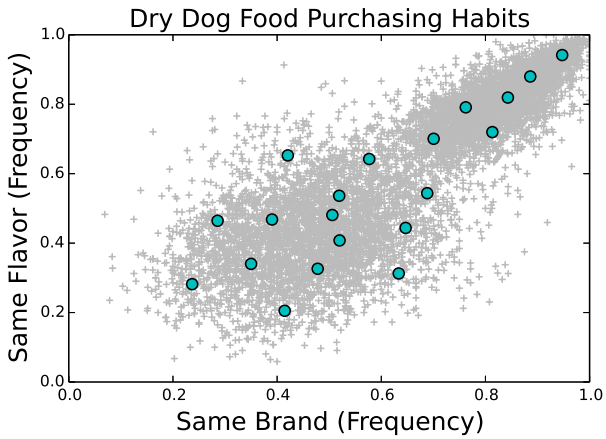


Fig. 7: Clustering of Brand and Flavor Purchasing Preferences. Grey plus signs (+) represent customer data points, and cyan circles represent cluster centers.

budgets, businesses need to decide who to target and what sort of advertisements are likely to be effective for each type of customer. We analyzed data generated from the model to determine where the advertising campaigns should be targeted geographically and identify customer purchasing profiles.

A. *Weekly Transaction Counts by Store*

First, we wanted to identify how close most customers live to a store since advertising to people who live or work too far away from a store is unlikely to be effective and will increase our costs. We analyzed the distribution of distances between customers' homes and their nearest stores. We found that most customers tend to live within 15 miles of their closest store, suggesting that we should limit our advertising to a 15-mile radius around each our store.

B. *Customer Segmentation*

Secondly, we profiled our customers' purchasing habits to optimize the advertising campaign's effectiveness by customizing the advertised products for each customer. For each customer, we generated a feature vector by computing the frequency with which they would purchase the same brand or flavor from one transaction to the next. We clustered the customers' feature vectors using the KMeans algorithm with a range of cluster counts. Twenty clusters converged the error.

Analysis of the feature vectors and clusters revealed four predominant purchasing profiles (Figure 7). High frequencies of purchasing the same flavors repeatedly were tightly-correlated with high frequencies of purchasing the same brands – these customers were likely to be happy with a particular item and kept purchasing the same item repeatedly. For our advertising campaign, it would be unlikely to get these customers to purchase different items, so we should create incentives to purchase larger quantities, especially when inventory levels are high and needed to be depleted. Other customers had a tendency to purchase either the same flavor or brand repeatedly, but varied in their choice of the other. For customers who prefer a particular brand, we should target our advertising campaign to suggest other flavors sold by that brand. Likewise, for customers who prefer a particular flavor, we should target our advertising campaign to suggest other brands with that flavor. Lastly, some customers had very low frequencies of purchasing neither the same brand nor flavor. Other factors, such as cost, not represented in this analysis may be driving these customers' purchasing habits and will need further study.

compare new and old

C. *Product Recommendations*

Need to see if we can evaluate by making vectors sparser

D. *Impact of Weather on Shopping Patterns by Region*

TODO: effect of weather, variations by region

VI. Discussion and Conclusion

The BigPetStore data generator is a domain-driven mathematical model and simulation for generating data with rich patterns. We described several updates to the model designed to increase the number and variety of correlations. By implementing a product enumerator, we increased the number of products and product fields, resulting in a significant increase in the number of possible purchasing profiles. The addition of a model for local weather simulation enabled the incorporation of geospatial and additional temporal correlations. A new JVM library implementation was described. Support for parallelization and interopability with tools in the big data ecosystem was demonstrated using an Apache Spark driver.

The math model was validated using ...

We see multiple opportunities for building on the work presented. ...

We intend to expand the model to incorporate additional factors. Deeper integration of existing fields such as location and purchasing profiles (e.g., to model regional purchasing preferences) will increase the variety of the semantic information encoded in the generated data. The incorporation of weather and climate data can be used to influence when

customers shop, the types of products they buy, and the amount of products purchased. For example, customers would be less likely to shop during snow storms, more likely to buy items such as winter apparel for their pets, and more likely to purchase bulk quantities to reduce the number of transactions. Modeling of time-dependent events such as sales, evolution of customer purchasing profiles over time, and "life events," such as the birth or passing of pets, will enable more interesting time-series analysis of the data. We would also like to expand the scope of the model to incorporate business processes (such as inventory management, customer complaints, employees, etc.), thus enabling queries about the relationship between internal business process and customer behavior.

ACKNOWLEDGMENT

The authors would like to thank Brian P. Clare and Casey Robinson for useful and interesting discussion. Will Benton, Trevor M. Cickovski, Erik Erlandson, Scott A. Hale, Hank Jakiela, Casey Robinson, and Douglas Thain provided invaluable feedback on the manuscript. JV would like to thank the Apache BigTop community for their interest in and support of BigPetStore. The authors would also like to thank Matt Fenwick, Nigel Savage, and Bhashit Parikh for their contributions to BigPetStore. The authors would like to thank Red Hat, Inc. and the University of Notre Dame for their support of this work.

REFERENCES

[1] "Apache Hadoop," http://hadoop.apache.org/.

[2] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy, "Hive - a petabyte scale data warehouse using Hadoop," in *Proceedings of the 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*. Ieee, 2010, pp. 996–1005.

[3] C. Olston, B. Reed, and U. Srivastava, "Pig latin: a not-so-foreign language for data processing," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of Data*, 2008, pp. 1099–1110.

[4] A. F. Gates, O. Natkovich, S. Chopra, P. Kamath, S. M. Narayanamurthy, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava, "Building a high-level dataflow system on top of Map-Reduce: the Pig experience," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1414–1425, 2009.

[5] "Apache Mahout," https://mahout.apache.org/.

[6] "Apache BigTop," http://bigtop.apache.org/.

[7] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The HiBench benchmark suite: Characterization of the MapReduce-based data analysis," in *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)*. IEEE, 2010, pp. 41–51.

[8] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolette, and H.-A. Jacobsen, "Bigbench: Towards an industry standard benchmark for big data analytics," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013, pp. 1197–1208.

[9] T. Rabl and M. Poess, "Parallel Data Generation for Performance Analysis of Large, Complex RDBMS," in *Proceedings of the Fourth International Workshop on Testing Database Systems*, 2011, pp. 1–6.

[10] M. Frank, M. Poess, and T. Rabl, "Efficient update data generation for DBMS benchmarks," in *Proceedings of the Third Joint WOSP/SIPEW International Conference on Performance Engineering - ICPE '12*. Boston, Massachusetts, USA: ACM Press, 2012.

[11] T. Rabl, M. Frank, H. M. Sergieh, and H. Kosch, "A data generator for cloud-scale benchmarking," in *Performance Evaluation, Measurement and Characterization of Complex Systems*, R. Nambiar and M. Poess, Eds. Springer, 2011, pp. 41–56.

[12] J. Gray, P. Sundaresan, S. Englert, K. Baclawski, and P. J. Weinberger, "Quickly generating billion-record synthetic databases," in *Proceedings of the 1994 ACM SIGMOD International Conference on Management of data*, 1994, pp. 243–252.

[13] N. Bruno and S. Chaudhuri, "Flexible Database Generators," in *VLDB '05 Proceedings of the 31st international conference on Very large data bases*, 2005, pp. 1097–1107.

[14] J. E. Hoag and C. W. Thompson, "A parallel general-purpose synthetic data generator," *ACM SIGMOD Record*, vol. 36, no. 1, pp. 19–24, Mar. 2007.

[15] "log-synth," https://github.com/tdunning/log-synth.

[16] A. Alexandrov, K. Tzoumas, and V. Markl, "Myriad: scalable and expressive data generation," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 1890–1893, 2012.

[17] A. Arasu, R. Kaushik, and J. Li, "Data generation using declarative constraints," in *Proceedings of the 2011 International Conference on Management of Data - SIGMOD '11*. New York, New York, USA: ACM Press, 2011, p. 685.

[18] A. Alexandrov, C. Brücke, and V. Markl, "Issues in big data testing and benchmarking," in *Proceedings of the Sixth International Workshop on Testing Database Systems - DBTest '13*. New York, New York, USA: ACM Press, 2013, p. 1.

[19] D. R. Jeske, B. Samadi, P. J. Lin, L. Ye, S. Cox, R. Xiao, T. Younglove, M. Ly, D. Holt, and R. Rich, "Generation of synthetic data sets for evaluating the accuracy of knowledge discovery systems," in *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining - KDD '05*. New York, New York, USA: ACM Press, 2005, p. 756.

[20] P. Lin, B. Samadi, and A. Cipolone, "Development of a synthetic data set generator for building and testing information discovery systems," in *Information Technology – New Generations, 2006*. Las Vegas, NV: IEEE, 2006, pp. 707–712.

[21] "Movielens," http://www.grouplens.org/datasets/movielens/.

[22] R. J. Nowling and J. Vyas, "A Domain-Driven, Generative Data Model for BigPetStore," in *Proceedings of the Fourth IEEE International Conference in Big Data and Cloud Computing*, 2014.

[23] "American Community Survey," http://www.census.gov/acs/www/.

[24] "https://www.drupal.org/project/namedb," http://incompetech.com/named/.

[25] "Zip Code Database Project," http://zips.sourceforge.net/.

[26] P. E. Klouden and E. Platen, *Numerical Solution of Stochastic Differential Equations*. Springer, 2013.