A PROJECT REPORT

On

A PROBABILISTIC MODEL FOR PREDICTING PURCHASE ORDER BY ONLINE CUTOMER

Submitted in partial fulfillment for the award of the degree
IN
Bachelor of Technology

By:

Hemalata (523/15)

Priyanka Mathur (526/15)

Raju (527/15)

Salman (524/15)

Under the supervision of

DR.PRABHAT VERMA

To

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Harcourt Butler Technological University, Kanpur-208002 (India)

(2017-18)

ABSTRACT

The mentioned system creating a Bayesian Belief network to analyze the behavior of the customers,and predict their next purchase, along with the order in which they are bought in that transaction.The model is designed to find the most frequent combinations of items. It is based on developing an efficient rule that outperforms the best available frequent pattern algorithms on a number of typical data sets. This will help in marketing and sales. The technique can be used to uncover interesting cross-sells and related products. Three different algorithms from association mining have been implanted and then best combination method is utilized to find more interesting results.The analyst then can perform the data mining and extraction and finally conclude the result and make appropriate decision.

We are given a large database of customer transactions. Each transaction consists of items purchased by a customer in a visit. We present an efficient algorithm model that generates all significant association rules between items in the database.

CERTIFICATE

This is to certify that this report entitled develop "**PROBABLISTIC MODEL FOR PREDICTING PURCHASE ORDER BY ONLINE CUSTOMER.**" which is submitted by Hemlata, Priyanka Mathur, Raju and Salman in partial fulfillment of the requirement for the award of the degree Bachelor of Technology in Computer Science and Engineering to the Department of Computer Science in Harcourt Butler Technical University, Kanpur is a record of candidate own work carried out by them under my supervision. The matter embodied in this thesis is original and has not been submitted for the award of any other degree.

SIGNATURE

DR. PRABHAT VERMA
(PROFESSOR)
Dept: Computer Science and Engg.
H.B.T.U Kanpur(U.P)

SIGNATURE

Prof. Raghuraj Singh
(H.O.D .CSE)
Dept: Computer Science and Engg.
H.B.T.U, Kanpur(U.P)

ACKNOWLEDGEMENT

Development of this project was a meticulous job and requires lot of commitment. It is a pleasure for us to express our thanks and heartiest gratitude to all those who helps us directly or indirectly during the development of this challenging project. We would like to take this opportunity to thank them all.

While we cheerfully share the credit for the accurate aspect of this project report, the mistakes and omissions we have to claim as our alone. Please bring them to our attention.

We would like to express our heartiest gratitude to **Prof. Dr.Prabhat Verma** for their highly valuable support require for development of this project. We would like to thank **Mrs. Monika Verma** for providing us the required useful information and guiding us through this project. They were very encouraging and helpful throughout this project.

Our sincere gratitude to H.B.T.U., in general, for providing excellent material and labs with all facilities for project development.

INDEX

LIST OF FIGURES

1. INTRODUCTION

1.1 Background

This project is a based on Machine Learning. The project objective is to develop PROBABLISTIC MODEL FOR PREDICTING PURCHASE ORDER BY ONLINE CUSTOMER.
Online shopping is the process whereby customer directly buy goods or services from a seller in real – time , without an intermediary service , over the Internet . It uses a dataset of electronic commerce customer's

1.2 Introduction to JUPYTER NOTEBOOK

What Is A Jupyter Notebook?

Jupyter Notebook App (formerly **IPython Notebook**) is an application running inside the browser. This guide describes how to install and use *Jupyter Notebook App* as normal desktop application, without using any remote. In this case, "notebook" or "notebook documents" denote documents that contain both code and rich text elements, such as figures, links, equations, ... Because of the mix of code and text elements, these documents are the ideal place to bring together an analysis description and its results as well as they can be executed perform the data analysis in real time. These documents are produced by the Jupyter Notebook App. "Jupyter" is a loose acronym meaning Julia, Python, and R. These programming languages were the first target languages of the Jupyter application, but nowadays, the notebook technology also supports many other languages.
The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.
As a server-client application, the Jupyter Notebook App allows you to edit and run your notebooks via a web browser. The application can be executed on a PC without Internet access or it can be installed on a remote server, where you can access it through the Internet. Its two main components are the kernels and a dashboard. A kernel is a program that runs and introspects the user's code. The Jupyter Notebook App has a kernel for Python code, but there are also kernels available for other programming languages. The dashboard of the application not only shows you the notebook documents that you have made and can reopen but can also be used to manage the kernels. you can which ones are running and shut them down if necessary. IPython is now the name of the Python backend, which is also known as the kernel.

you can which ones are running and shut them down if necessary. IPython is now the name of the Python backend, which is also known as the kernel.

Jupyter has a beautiful notebook that lets you write and execute code, analyze data, embed content, and share reproducible work. Jupyter Notebook (previously referred to as IPython Notebook) allows you to easily share your code, data, plots, and explanation in a sinle notebook. Publishing is flexible: PDF, HTML, ipynb, dashboards, slides, and more. Code cells are based on an input and output format.

Some useful packages that we'll use in this project.

- Pandas: import data via a url and create a dataframe to easily handle data for analysis and graphing. See examples of using Pandas here: https://plot.ly/pandas/.
- NumPy: a package for scientific computing with tools for algebra, random number generation, integrating with databases, and managing data. See examples of using NumPy here: https://plot.ly/numpy/.
- SciPy: a Python-based ecosystem of packages for math, science, and engineering.
- Plotly: a graphing library for making interactive, publication-quality graphs. See examples of statistic, scientific, 3D charts, and more here: https://plot.ly/python.

  A Jupyter notebook lets you write and execute Python code in your web browser. Jupyter notebooks make it very easy to tinker with code and execute it in bits and pieces; for this reason Jupyter notebooks are widely used in scientific computing.

1.3 Introduction to MACHINE LEARNING

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. **Machine learning focuses on the development of computer programs** that can access data and use it learn for themselves.

The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. **The primary aim is to allow the computers learn automatically** without human intervention or assistance and adjust actions accordingly.

Machine learning is a field of computer science that uses statistical techniques to give computer systems the ability to "learn" (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed.

The name *machine learning* was coined in 1959 by Arthur Samuel. Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data – such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions, through building a model from sample inputs. Machine learning is employed in a range of computing tasks where designing and programming explicit algorithms with good performance is difficult or infeasible; example applications include email filtering, detection of network intruders or malicious insiders working towards a data breach, optical character recognition (OCR), learning to rank, and computer vision.

Machine learning is closely related to (and often overlaps with) computational statistics, which also focuses on prediction-making through the use of computers. It has strong ties to mathematical optimization, which delivers methods, theory and application domains to the field. Machine learning is sometimes conflated with data mining[where the latter subfield focuses more on exploratory data analysis and is known as unsupervised learning Machine learning can also be unsupervised and be used to learn and establish baseline behavioral profiles for various entities and then used to find meaningful anomalies.Within the field of data analytics, machine learning is a method used to devise complex models and algorithms that lend themselves to prediction; in commercial use, this is known as predictive analytics. These analytical models allow researchers, data scientists, engineers, and analysts to "produce reliable, repeatable decisions and results" and uncover "hidden insights" through learning from historical relationships and trends in the data.

History and relationship to other fields:

Arthur Samuel, an American pioneer in the field of computer gaming and artificial intelligence, coined the term "Machine Learning" in 1959 while at IBM. As a scientific endeavour, machine learning grew out of the quest for artificial intelligence. Already in the early days of AI as an academic discipline, some researchers were interested in having machines learn from data. They attempted to approach the problem with various symbolic methods, as well as what were then termed "neural networks"; these were mostly perceptrons and other models that were later found to be reinventions of the generalized linear models of statistics. Probabilistic reasoning was also employed, especially in automated medical diagnosis. However, an increasing emphasis on the logical, knowledge-based approach caused a rift between AI and machine learning. Probabilistic systems were plagued by theoretical and practical problems of data acquisition and representation. By 1980, expert systems had come to dominate AI, and statistics was out of favor. Work on symbolic/knowledge-based learning did continue within AI, leading to inductive logic programming, but the more statistical line of research was now outside the field of AI proper, in pattern recognition and information retrieval.

Neural networks research had been abandoned by AI and computer science around the same time. This line, too, was continued outside the AI/CS field, as "connectionism", by researchers from other disciplines including Hopfield , Rumelhart and Hinton. Their main success came in the mid-1980s with the reinvention of backpropagation.

Machine learning and data mining often employ the same methods and overlap significantly, but while machine learning focuses on prediction, based on *known* properties learned from the training data, data mining focuses on the discovery of (previously) *unknown* properties in the data (this is the analysis step of knowledge discovery in databases). Data mining uses many machine learning methods, but with different goals; on the other hand, machine learning also employs data mining methods as "unsupervised learning" or as a preprocessing step to improve learner accuracy. Much of the confusion between these two research communities (which do often have separate conferences and separate journals, ECML PKDD being a major exception) comes from the basic assumptions they work with: in machine learning, performance is usually evaluated with respect to the ability to *reproduce known* knowledge, while in knowledge discovery and data mining (KDD) the key task is the discovery of previously *unknown* knowledge. Evaluated with respect to known knowledge, an uninformed (unsupervised) method will easily be outperformed by other supervised methods, while in a typical KDD task, supervised methods cannot be used due to the unavailability of training data.

Relation to statistics:

Machine learning also has intimate ties to optimization: many learning problems are formulated as minimization of some loss function on a training set of examples. Loss functions express the discrepancy between the predictions of the model being trained and the actual problem instances (for example, in classification, one wants to assign a label to instances, and models are trained to correctly predict the pre-assigned labels of a set of examples). The difference between the two fields arises from the goal of generalization: while optimization algorithms can minimize the loss on a training set, machine learning is concerned with minimizing the loss on unseen samples

Overview

Tom M. Mitchell provided a widely quoted, more formal definition of the algorithms studied in the machine learning field: "A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P* if its performance at tasks in *T*, as measured by *P*, improves with experience *E*. This definition of the tasks in which machine learning is concerned offers a fundamentally operational definition rather than defining the field in cognitive terms. This follows Alan Turing's proposal in his paper "Computing Machinery and Intelligence", in which the question "Can machines think?" is replaced with the question "Can machines do what we (as thinking entities) can do?".In Turing's proposal the various characteristics that could be possessed by a *thinking machine* and the various implications in constructing one are exposed.

Some machine learning methods

Machine learning algorithms are often categorized as supervised or unsupervised.

- **Supervised machine learning algorithms** can apply what has been learned in the past to new data using labeled examples to predict future events. Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly.
- In contrast, **unsupervised machine learning algorithms** are used when the information used to train is neither classified nor labeled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system doesn't figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.
- **Semi-supervised machine learning algorithms** fall somewhere in between supervised and unsupervised learning, since they use both labeled and unlabeled data for training – typically a small amount of labeled data and a large amount of unlabeled data. The systems that use this method are able to considerably improve learning accuracy. Usually, semi-supervised learning is chosen when the acquired labeled data requires skilled and relevant resources in order to train it / learn from it. Otherwise, acquiring unlabeled data generally doesn't require additional resources.
- **Reinforcement machine learning algorithms** is a learning method that interacts with its environment by producing actions and discovers errors or rewards. Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning. This method allows machines and software agents to automatically determine the ideal behavior within a specific context in order to maximize its performance. Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal.

  Machine learning enables analysis of massive quantities of data. While it generally delivers faster, more accurate results in order to identify profitable opportunities or dangerous risks, it may also require additional time and resources to train it properly. Combining machine learning with AI and cognitive technologies can make it even more effective in processing large volumes of information.

Machine learning applications

Another categorization of machine learning tasks arises when one considers the desired *output* of a machine-learned system:

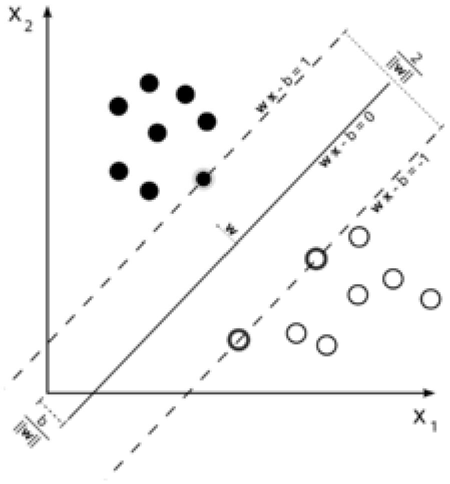


Fig-1

A support vector machine is a classifier that divides its input space into two regions, separated by a linear boundary. Here, it has learned to distinguish black and white circles

- In classification, inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one or more (multi-label classification) of these classes. This is typically tackled in a supervised way. Spam filtering is an example of classification, where the inputs are email (or other) messages and the classes are "spam" and "not spam".
- In regression, also a supervised problem, the outputs are continuous rather than discrete.
- In clustering, a set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task.
- Density estimation finds the distribution of inputs in some space.
- Dimensionality reduction simplifies inputs by mapping them into a lower-dimensional space. Topic modeling is a related problem, where a program is given a list of human language documents and is tasked to find out which documents cover similar topics.

Among other categories of machine learning problems, learning to learn learns its own inductive bias based on previous experience. Developmental learning, elaborated for robot learning, generates its own sequences (also called curriculum) of learning situations to cumulatively acquire repertoires of novel skills through autonomous self-exploration and social interaction with human teachers and using guidance mechanisms such as active learning, maturation, motor synergies, and imitation.

Theory:

A core objective of a learner is to generalize from its experience. Generalization in this context is the ability of a learning machine to perform accurately on new, unseen examples/tasks after having experienced a learning data set. The training examples come from some generally unknown probability distribution (considered representative of the space of occurrences) and the learner has to build a general model about this space that enables it to produce sufficiently accurate predictions in new cases.

The computational analysis of machine learning algorithms and their performance is a branch of theoretical computer science known as computational learning theory. Because training sets are finite and the future is uncertain, learning theory usually does not yield guarantees of the performance of algorithms. Instead, probabilistic bounds on the performance are quite common. The bias–variance decomposition is one way to quantify generalization error.

For the best performance in the context of generalization, the complexity of the hypothesis should match the complexity of the function underlying the data. If the hypothesis is less complex than the function, then the model has underfit the data. If the complexity of the model is increased in response, then the training error decreases. But if the hypothesis is too complex, then the model is subject to overfitting and generalization will be poorer.

In addition to performance bounds, computational learning theorists study the time complexity and feasibility of learning. In computational learning theory, a computation is considered feasible if it can be done in polynomial time. There are two kinds of time complexity results. Positive results show that a certain class of functions can be learned in polynomial time. Negative results show that certain classes cannot be learned in polynomial time.

Approaches

List of machine learning algorithms

Decision tree learning:

Decision tree learning uses a decision tree as a predictive model, which maps observations about an item to conclusions about the item's target value.

Association rule learning:

Association rule learning is a method for discovering interesting relations between variables in large databases.

Artificial neural networks:

An artificial neural network (ANN) learning algorithm, usually called "neural network" (NN), is a learning algorithm that is vaguely inspired by biological neural networks.

Computations are structured in terms of an interconnected group of artificial neurons, processing information using a connectionist approach to computation. Modern neural networks are non-linearstatistical data modeling tools. They are usually used to model complex relationships between inputs and outputs, to find patterns in data, or to capture the statistical structure in an unknown joint probability distribution between observed variables.

Deep learning:

Falling hardware prices and the development of GPUs for personal use in the last few years have contributed to the development of the concept of deep learning which consists of multiple hidden layers in an artificial neural network. This approach tries to model the way the human brain processes light and sound into vision and hearing. Some successful applications of deep learning are computer vision and speech recognition.

Inductive logic programming:

Inductive logic programming (ILP) is an approach to rule learning using logic programming as a uniform representation for input examples, background knowledge, and hypotheses. Given an encoding of the known background knowledge and a set of examples represented as a logical database of facts, an ILP system will derive a hypothesized logic program that entails all positive and no negative examples. Inductive programming is a related field that considers any kind of programming languages for representing hypotheses (and not only logic programming), such as functional programs.

Support vector machines:

Support vector machines (SVMs) are a set of related supervised learning methods used for classification and regression. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other.

Clustering:

Cluster analysis is the assignment of a set of observations into subsets (called *clusters*) so that observations within the same cluster are similar according to some predesignated criterion or criteria, while observations drawn from different clusters are dissimilar. Different clustering techniques make different assumptions on the structure of the data, often defined by some *similarity metric* and evaluated for example by *internal compactness* (similarity between members of the same cluster) and *separation* between different clusters. Other methods are based on *estimated density* and *graph connectivity*. Clustering is a method of unsupervised learning, and a common technique for statistical data analysis.

Bayesian networks:

A Bayesian network, belief network or directed acyclic graphical model is a probabilistic graphical model that represents a set of random variables and their conditional independencies via a directed acyclic graph (DAG). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases. Efficient algorithms exist that perform inference and learning.

Reinforcement learning:

Reinforcement learning is concerned with how an *agent* ought to take *actions* in an *environment* so as to maximize some notion of long-term *reward*. Reinforcement learning algorithms attempt to find a *policy* that maps *states* of the world to the actions the agent ought to take in those states. Reinforcement learning differs from the supervised learning problem in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected.

Representation learning:

Several learning algorithms, mostly unsupervised learning algorithms, aim at discovering better representations of the inputs provided during training. Classical examples include principal components analysis and cluster analysis. Representation learning algorithms often attempt to preserve the information in their input but transform it in a way that makes it useful, often as a pre-processing step before performing classification or predictions, allowing reconstruction of the inputs coming from the unknown data generating distribution, while not being necessarily faithful for configurations that are implausible under that distribution.

Manifold learning algorithms attempt to do so under the constraint that the learned representation is low-dimensional. Sparse coding algorithms attempt to do so under the constraint that the learned representation is sparse (has many zeros). Multilinear subspace learning algorithms aim to learn low-dimensional representations directly from representations for multidimensional data, without reshaping them into (high-dimensional) vectors. Deep learning algorithms discover multiple levels of representation, or a hierarchy of features, with higher-level, more abstract features defined in terms of (or generating) lower-level features. It has been argued that an intelligent machine is one that learns a representation that disentangles the underlying factors of variation that explain the observed data.

Similarity and metric learning:

In this problem, the learning machine is given pairs of examples that are considered similar and pairs of less similar objects. It then needs to learn a similarity function (or a distance metric function) that can predict if new objects are similar. It is sometimes used in Recommendation systems.

Sparse dictionary learning:

In this method, a datum is represented as a linear combination of basis functions, and the coefficients are assumed to be sparse. Let *x* be a *d*-dimensional datum, *D* be a *d* by *n*matrix, where each column of *D* represents a basis function. *r* is the coefficient to represent *x* using *D*.
Learning a dictionary along with sparse representations is strongly NP-hard and also difficult to solve approximately.A popular heuristic method for sparse dictionary learning is K-SVD.
Sparse dictionary learning has been applied in several contexts. In classification, the problem is to determine which classes a previously unseen datum belongs to. Suppose a dictionary for each class has already been built. Then a new datum is associated with the class such that it's best sparsely represented by the corresponding dictionary. Sparse dictionary learning has also been applied in image de-noising. The key idea is that a clean image patch can be sparsely represented by an image dictionary, but the noise cannot.

Genetic algorithms:

A genetic algorithm (GA) is a search heuristic that mimics the process of natural selection, and uses methods such as mutation and crossover to generate new genotype in the hope of finding good solutions to a given problem. In machine learning, genetic algorithms found some uses in the 1980s and 1990s. Conversely, machine learning techniques have been used to improve the performance of genetic and evolutionary algorithms.

Rule-based machine learning:

Rule-based machine learning is a general term for any machine learning method that identifies, learns, or evolves "rules" to store, manipulate or apply, knowledge. The defining characteristic of a rule-based machine learner is the identification and utilization of a set of relational rules that collectively represent the knowledge captured by the system. This is in contrast to other machine learners that commonly identify a singular model that can be universally applied to any instance in order to make a prediction.Rule-based machine learning approaches include learning classifier systems, association rule learning, and artificial immune systems.

Learning classifier systems:

Learning classifier systems (LCS) are a family of rule-based machine learning algorithms that combine a discovery component (e.g. typically a genetic algorithm) with a learning component (performing either supervised learning, reinforcement learning, or unsupervised learning). They seek to identify a set of context-dependent rules that collectively store and apply knowledge in a piecewise manner in order to make predictions.

Applications:

Applications for machine learning include:

- Agriculture
- Automated theorem proving
- Adaptive websites
- Affective computing
- Bioinformatics
- Brain–machine interfaces
- Cheminformatics
- Classifying DNA sequences
- Computational anatomy
- Computer Networks
- Telecommunication
- Computer vision, including object recognition
- Detecting credit-card fraud
- General game playing
- Information retrieval
- Internet fraud detection
- Linguistics
- Marketing
- Machine learning control
- Machine perception
- Medical diagnosis
- Economics
- Insurance
- Natural language processing
- Natural language understanding
- Optimization and metaheuristic
- Online advertising
- Recommender systems
- Robot locomotion
- Search engines
- Sentiment analysis (or opinion mining)
- Sequence mining
- Speech and handwriting recognition
- Financial market analysis
- Structural health monitoring
- Syntactic pattern recognition
- Time series forecasting
- User behavior analytics
- Translation

Ethics:

Machine learning poses a host of ethical questions. Systems which are trained on datasets collected with biases may exhibit these biases upon use (algorithmic bias), thus digitizing cultural prejudices.[50] For example, using job hiring data from a firm with racist hiring policies may lead to a machine learning system duplicating the bias by scoring job applicants against similarity to previous successful applicants.[51][52] Responsible collection of data and documentation of algorithmic rules used by a system thus is a critical part of machine learning.

Because language contains biases, machines trained on language *corpora* will necessarily also learn bias.

Other forms of ethical challenges, not related to personal biases, are more seen in health care. There are concerns among health care professionals that these systems might not be designed in the public's interest, but as income generating machines. This is especially true in the United States where there is a perpetual ethical dilemma of improving health care, but also increasing profits. For example, the algorithms could be designed to provide patients with unnecessary tests or medication in which the algorithm's proprietary owners hold stakes in. There is huge potential for machine learning in health care to provide professionals are great tool to diagnose, medicate, and even plan recovery paths for patients, but this will not happen until the personal biases mentioned previously, and these "greed" biases are addressed.

1.4 Introduction to Python (Programming Language) :

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python interpreters are available for many operating systems. C Python, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations. C Python is managed by the non-profit Python Software Foundation.

Features and Philosophy:

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution(late binding), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the Lisp tradition. It has filter(), map(), and reduce() functions; list comprehensions, dictionaries, and sets; and generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

The language's core philosophy is summarized in the document *The Zen of Python* (*PEP 20*), which includes aphorisms such as:

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts

Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.

While offering choice in coding methodology, the Python philosophy rejects exuberant syntax (such as that of Perl) in favor of a simpler, less-cluttered grammar. As Alex Martelli put it: "To describe something as 'clever' is *not* considered a compliment in the Python culture."[48] Python's philosophy rejects the Perl "there is more than one way to do it" approach to language design in favor of "there should be one—and preferably only one—obvious way to do it".

Python's developers strive to avoid premature optimization, and reject patches to non-critical parts of CPython that would offer marginal increases in speed at the cost of clarity.]When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

An important goal of Python's developers is keeping it fun to use. This is reflected in the language's name—a tribute to the British comedy group Monty Python—and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (from a famous Monty Python sketch) instead of the standard foo and bar.

A common neologism in the Python community is *pythonic*, which can have a wide range of meanings related to program style.

To say that code is pythonic is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called *unpythonic*

Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as *Pythonists*, *Pythonistas*, and *Pythoneers.*

Syntax And Semantic:

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than C or Pascal.

Indentation:

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents the program's semantic structure. This feature is also sometimes termed the off-side rule.

Libraries:

Python's large standard library, commonly cited as one of its greatest strengths, provides tools suited to many tasks. For Internet-facing applications, many standard formats and protocols such as MIME and HTTP are supported. It includes modules for creating graphical user interfaces, connecting to relational databases, generating pseudorandom numbers, arithmetic with arbitrary precision decimals, manipulating regular expressions, and unit testing.

Some parts of the standard library are covered by specifications (for example, the Web Server Gateway Interface (WSGI) implementation wsgiref follows PEP 333), but most modules are not. They are specified by their code, internal documentation, and test suites (if supplied). However, because most of the standard library is cross-platform Python code, only a few modules need altering or rewriting for variant implementations.

As of March 2018, the Python Package Index (PyPI), the official repository for third-party Python software, contains over 130,000 packages with a wide range of functionality, including:

- Graphical user interfaces
- Web frameworks

- Multimedia
- Databases
- Networking
- Test frameworks
- Automation
- Web scraping
- Documentation
- System administration
- Scientific computing
- Text processing
- Image processing

Development Environments:

Most Python implementations (including CPython) include a read–eval–print loop (REPL), permitting them to function as a command line interpreter for which the user enters statements sequentially and receives results immediately.

Other shells, including IDLE and IPython, add further abilities such as auto-completion, session state retention and syntax highlighting.

As well as standard desktop integrated development environments (see Wikipedia's "Python IDE" article), there are Web browser-based IDEs; SageMath (intended for developing science and math-related Python programs); PythonAnywhere, a browser-based IDE and hosting environment; and Canopy IDE, a commercial Python IDE emphasizing scientific computin

Implementations:

- Reference implementation:

CPython is the reference implementation of Python. It is written in C, meeting the C89 standard with several select C99 features. It compiles Python programs into an intermediate bytecode which is then executed by its virtual machine .CPython is distributed with a large standard library written in a mixture of C and native Python. It is available for many platforms, including Windows and most modern Unix-like systems. Platform portability was one of its earliest priorities.

- Other implementations:

PyPy is a fast, compliant interpreter of Python 2.7 and 3.5. Its just-in-time compiler brings a significant speed improvement over CPython. Stackless Python is a significant fork of CPython that implements microthreads; it does not use the C memory stack, thus allowing massively concurrent programs. PyPy also has a stackless version.

- Unsupported implementations:

Google began a project named Unladen Swallow in 2009 with the aim of speeding up the Python interpreter fivefold by using the LLVM, and of improving its multithreading ability to scale to thousands of cores. Psyco is a just-in-time specialising compiler that integrates with CPython and transforms bytecode to machine code at runtime. The emitted code is specialised for certain data types and is faster than standard Python code.

In 2005, Nokia released a Python interpreter for the Series 60 mobile phones named PyS60. It includes many of the modules from the CPython implementations and some additional modules to integrate with the Symbian operating system. The project has been kept up-to-date to run on all variants of the S60 platform, and several third-party modules are available. The Nokia N900 also supports Python with GTK widget libraries, enabling programs to be written and run on the target device.

Performance:

A performance comparison of various Python implementations on a non-numerical (combinatorial) workload was presented at EuroSciPy.

1.5 Introduction To A Probabilistic Model For Predicting Purchase Order By Online Customer

Naive Bayes classifiers, a family of classifiers that are based on the popular Bayes' probability theorem, are known for creating simple yet well performing models, especially in the fields of document classification and disease prediction. In this first part of a series, we will take a look at the theory of naive Bayes classifiers and introduce the basic concepts of text classification. We will implement a Bayesian Belief network to analyze the behavior of the customers, and predict their next purchase, along with the order in which they are bought in that transaction.

predictive modeling is *supervised pattern classification*; supervised pattern classification is the task of training a model based on labeled training data which then can be used to assign a pre-defined class label to new objects. One example that we will explore throughout this article is spam filtering via naive Bayes classifiers in order to predict purchase order. Naive Bayes classifiers, a family of classifiers that are based on the popular Bayes' probability theorem, are known for creating simple yet well performing models, especially in the fields of document classification and disease prediction.

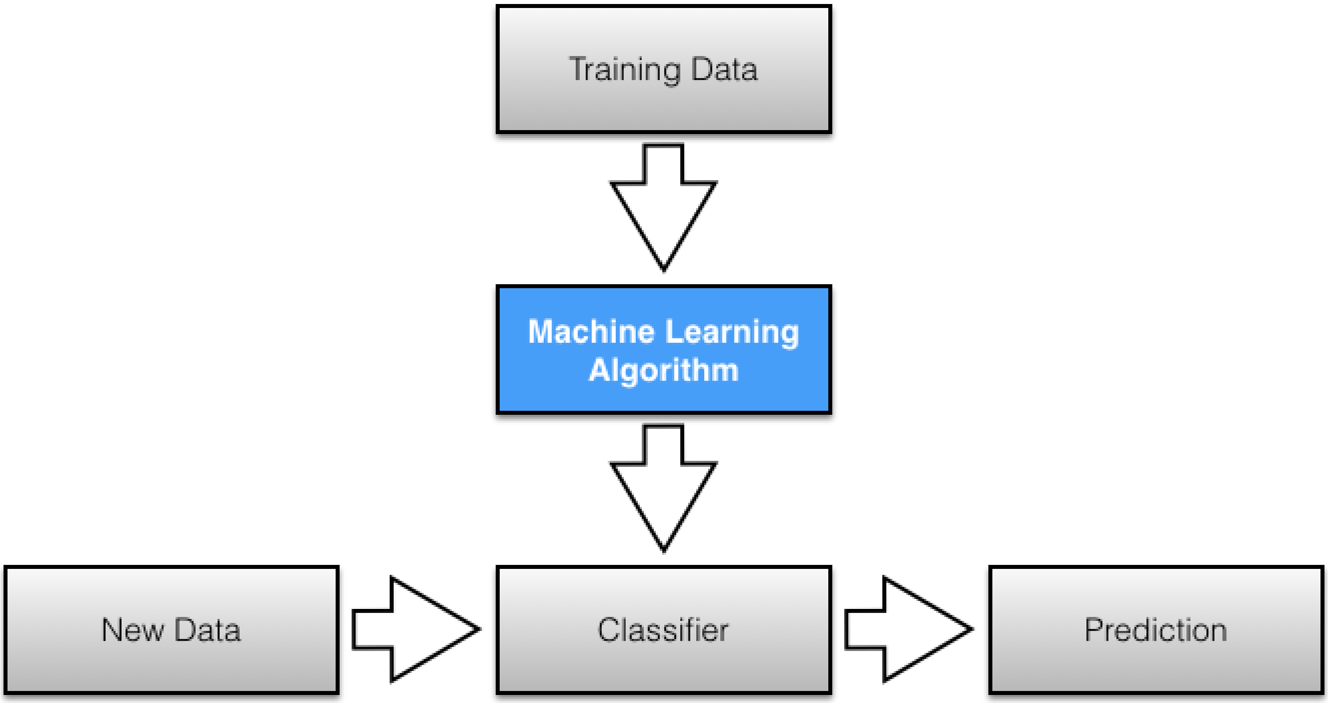


Fig-2 **(predictive modeling *is* supervised pattern classification)**

Modern machine learning solutions using Sir Thomas Bayes' Theorem are in wide use today. Here I will show how a Bayesian Belief Network (BN) can be used to solve this problem. BN models are very suitable models for operations like Instacart because they will run in real time for a good online experience and are trivial to update as new data (new orders) pours in. They must make decisions in an instant, pun intended, and BN models are very fast.

1.6 Bayes Theorem

The Bayes theorem describes the probability of an event based on the prior knowledge of the conditions that might be related to the event. If we know the conditional probability $P\ (\frac{A}{B})$, we can use the bayes rule to find out the reverse probabilities $P\ (\frac{B}{A})$

How can we do can that??

$$P\ (\frac{A}{B}) = \frac{P\ (A \cap B)}{P\ (B)}$$

$$P\ (\frac{B}{A}) = \frac{P\ (A \cap B)}{P\ (A)}$$

$$P\ (A \cap B) = P\ (\frac{A}{B}) * P\ (B) = P\ (\frac{B}{A}) * P\ (A)$$

The above statement is the general representation of the Bayes rule.

For the previous example – if we now wish to calculate the probability of having a pizza for lunch provided you had a bagel for breakfast would be = 0.7 * 0.5/0.6.

We can generalize the formula further.

If multiple events A_i form an exhaustive set with another event B.

We can write the equation as

$$P\ (A_i/B) = \frac{P(B|Ai)*P(Ai)}{\sum(i=1\ to\ n)\ P(B|Ai)*P(Ai)}$$

Bayesian Probability

Bayesian probability represents a level of certainty relating to a potential outcome or idea. This is in contrast to a frequentistprobability that represents the frequency with which a particular outcome will occur over any number of trials.

Problem setup

We are given all the previous orders of 75,000 test users and asked, "what products will the user most likely reorder in this test order? The customer can pick from any of 49,000 products to put into an order. Yet we are only asked about possible reordered products. We know that p(reordered|product) has no value or is = FV (a Filtered Value - one form of missing-ness) if the product has not been in a previous order. This effects how we make some calculations below.

We have evidence about each order placed including number of products, time of day, days since last order. We know how many prior orders the user has placed. We know how many products are reordered. So what evidence will help us infer the products that are most likely to be reordered? I will try a Bayesian Solution.

The Prior

In the context of Bayes's Theorem, **Priors** refer generically to the beliefs an agent holds regarding a fact, hypothesis or consequence, before being presented with evidence. Upon being presented with new evidence, the agent can multiply their prior with a likelihood distribution to calculate a new (posterior) probability for their belief.

When I use the word Prior, please do not confuse this with file names in this competition. In the files, the word prior means past. OK. When I use the word Prior, I mean The Prior is our complete belief in some phenomena. What do we believe a probability is for the occurrence of this phenomena? In our competition The Prior = p(reordered | product_id). So if product = banana, what do we believe it's probability of being reordered? How do we begin believing?

A Flat Prior is often used. If we have 49,000 products, our Prior might be that each product is equally likely to be reordered. Every product has the same probability of occurrence ~= 1/49,000 (when plotted by product it looks 'Flat'). Use this Prior if you know nothing. But we do have lots of evidence to develop a better Prior than this.

An Informed Prior is used when evidence is available that may apply to a given occurrence of an event. We have the previous orders of our test users as a 'whole'. If we use this bulk information we can easily calculate p(reordered | product_id) = (total times reordered +1) / (total times ordered +2). It is this Prior that I use in my Python Notebook called "Calculate a Prior and Bayes Factors for prediction".

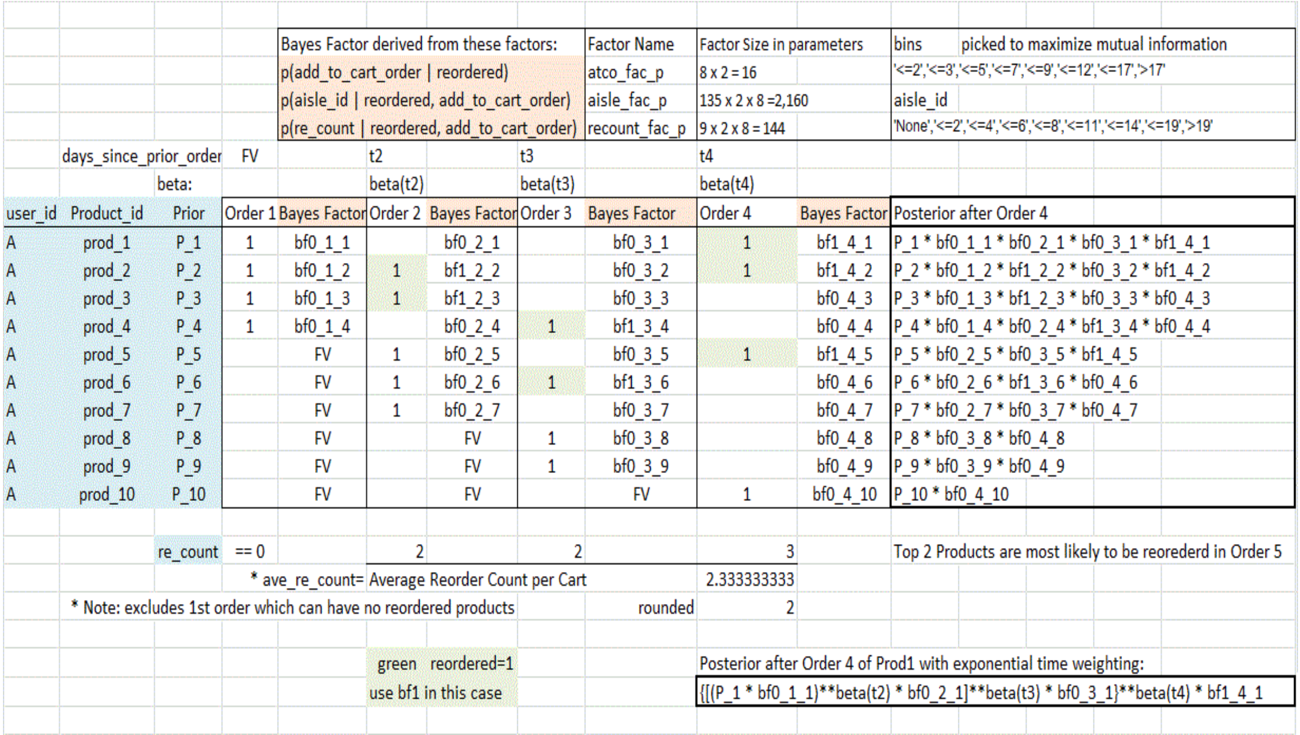
Bayes Factors for updating Prior:

The Bayes Factor is the middle items in Bayes Theorem:
Posterior = Bayes Factor x Prior
or for our competition
p(reordered | e) = [p(e | reordered)/p(e)] x p(reordered)
where e is evidence about the item in the new order. (in my code I label this as bf1 if reordered=1 and bf0 when reordered=0)

Once the Posterior is calculated, it becomes our new Prior (our new belief). An Updated Prior is the new Prior.

So the Bayes Factor = p(e | reordered)/p(e). The probability of evidence p(e) is considered fixed so that a very simple relationship emerges: Posterior ~= p(e | reordered) x Prior!!! where ~= is "proportional to". This is a very simple recursive method to Update a Prior.
Posterior ~= BFn x BFn-1 x ... x BF1 x Prior

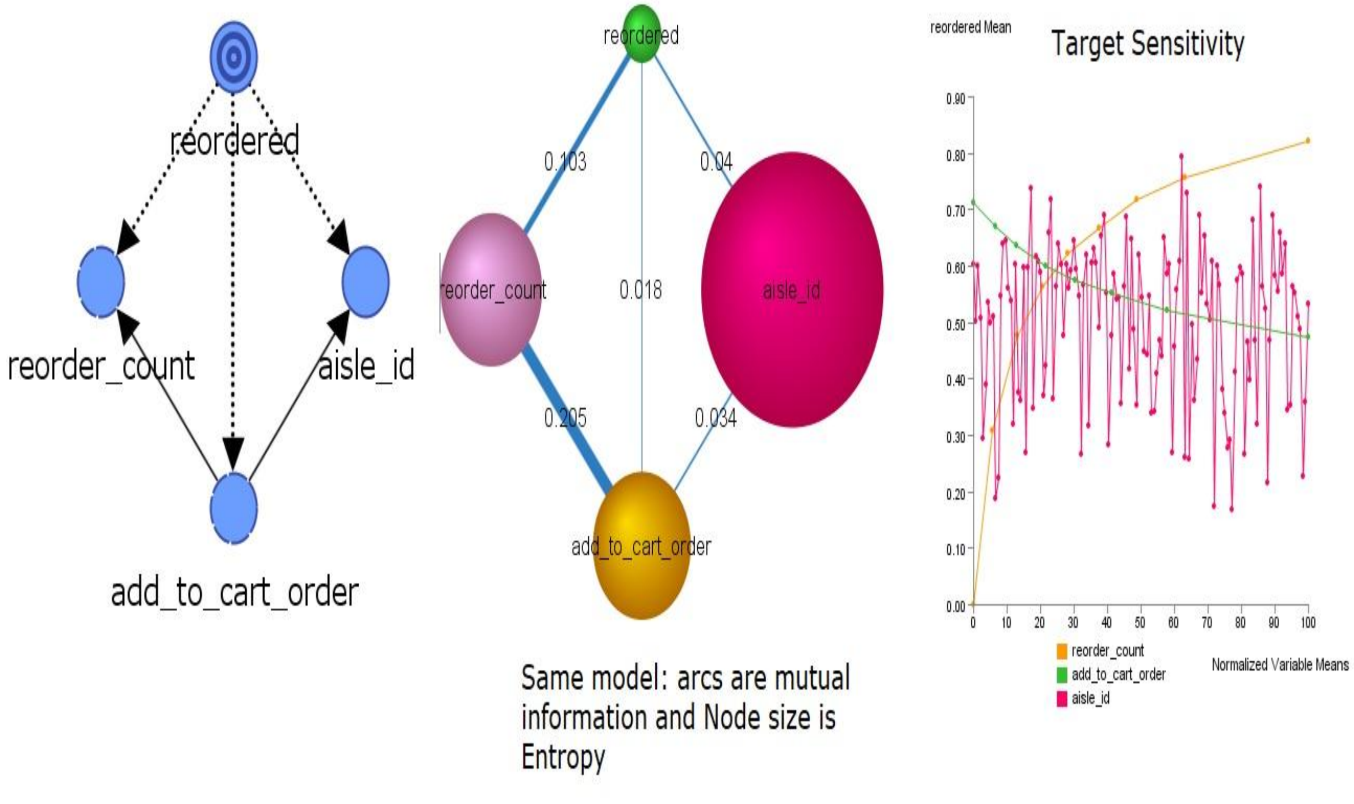
To get the final list of possible products a user may reorder, we get a full list of the orders and items ordered. Each product begins with some Prior and this Prior is Updated with each order. See this, hopefully helpful spreadsheet, that shows four orders and the calculations for p(reordered) (aka posterior) as an example.



				Bayes Factor derived from these factors:				Factor Name	Factor Size in parameters		bins	picked to maximize mutual information
				p(add_to_cart_order | reordered)				atco_fac_p	8 x 2 = 16		'<=2','<=3','<=5','<=7','<=9','<=12','<=17','>17'	
				p(aisle_id | reordered, add_to_cart_order)				aisle_fac_p	135 x 2 x 8 =2,160		aisle_id	
				p(re_count | reordered, add_to_cart_order)				recount_fac_p	9 x 2 x 8 = 144		'None','<=2','<=4','<=6','<=8','<=11','<=14','<=19','>19'	
	days_since_prior_order		FV		t2		t3		t4			
		beta:			beta(t2)		beta(t3)		beta(t4)			
user_id	Product_id	Prior	Order 1	Bayes Factor	Order 2	Bayes Factor	Order 3	Bayes Factor	Order 4	Bayes Factor	Posterior after Order 4	
A	prod_1	P_1	1	bf0_1_1		bf0_2_1		bf0_3_1	1	bf1_4_1	P_1 * bf0_1_1 * bf0_2_1 * bf0_3_1 * bf1_4_1	
A	prod_2	P_2	1	bf0_1_2	1	bf1_2_2		bf0_3_2	1	bf1_4_2	P_2 * bf0_1_2 * bf1_2_2 * bf0_3_2 * bf1_4_2	
A	prod_3	P_3	1	bf0_1_3	1	bf1_2_3		bf0_3_3		bf0_4_3	P_3 * bf0_1_3 * bf1_2_3 * bf0_3_3 * bf0_4_3	
A	prod_4	P_4	1	bf0_1_4		bf0_2_4	1	bf1_3_4		bf0_4_4	P_4 * bf0_1_4 * bf0_2_4 * bf1_3_4 * bf0_4_4	
A	prod_5	P_5		FV	1	bf0_2_5		bf0_3_5	1	bf1_4_5	P_5 * bf0_2_5 * bf0_3_5 * bf1_4_5	
A	prod_6	P_6		FV	1	bf0_2_6	1	bf1_3_6		bf0_4_6	P_6 * bf0_2_6 * bf1_3_6 * bf0_4_6	
A	prod_7	P_7		FV	1	bf0_2_7		bf0_3_7		bf0_4_7	P_7 * bf0_2_7 * bf0_3_7 * bf0_4_7	
A	prod_8	P_8		FV		FV	1	bf0_3_8		bf0_4_8	P_8 * bf0_3_8 * bf0_4_8	
A	prod_9	P_9		FV		FV	1	bf0_3_9		bf0_4_9	P_9 * bf0_3_9 * bf0_4_9	
A	prod_10	P_10		FV		FV		FV	1	bf0_4_10	P_10 * bf0_4_10	
		re_count	== 0		2		2		3		Top 2 Products are most likely to be reorederd in Order 5	
				* ave_re_count=	Average Reorder Count per Cart				2.333333333			
	* Note: excludes 1st order which can have no reordered products							rounded	2			
					green reordered=1				Posterior after Order 4 of Prod1 with exponential time weighting:			
					use bf1 in this case				{[(P_1 * bf0_1_1)**beta(t2) * bf0_2_1]**beta(t3) * bf0_3_1}**beta(t4) * bf1_4_1			

FIG-3(Bayesian Factors)

To calculate p(e|reordered) I have devised a simple augmented Naive Bayesian Network. Many here have noticed profound importance of the add_to_cart_order. It is a primary factor. I engineered a feature called re_count. It is the number of products in an order we reordered. The Aisle where a product is located is important too. These three factors are used to predict the probability of the evidence given in an order. Here is a resulting BN DAG (directed acyclic graph) of the model:



Same model: arcs are mutual information and Node size is Entropy

Fig-4 (Naive Bayesian Network)

Project Methodology for Model:

Our goal is to predict

$$P(reordered = 1|e)$$

Where reordered = 1 indicates that the product was reordered, given e. The random variable ‘e’ is the event where
Add to cart order = ‘atco1’
Aisle = ‘aisle_id’
Number of times reordered = ‘recount_c’

$$p(e \mid reordered) \ =$$

Using Bayes’ Theorem, we calculate the posterior probability using:

We can replace the equality with a proportionality with the following formula:

Once the posterior is calculated, it becomes our new prior.

For our first order, we get the prior calculating an informed prior using the formula:

Giving the model Forgetfulness

The last part, I promise! The technique imparts memory loss on the model and makes more recent events more important in our calculation. It is called exponential time weighting. This process can use exponential function ** when working when updating probabilities. And is very simple to apply in the real world, on occurrence tables, as a linear function (not shown here).

Posterior = BF x Prior ^beta

Imagine your users forget 5% products they have ordered every month. This forgetfulness accumulates over time. People move on to new habits. A Prior can be partially forgotten during the updating process. So if you believe that users remember only 95% of products per month, you update this way. Beta is a testable factor that will have an optimal setting on this data (who knows its value?).

Posterior = (BF2 x ((BF1 x Prior)^beta(t1))^beta(t2)

where beta is less than 1 and if beta = 0, there is no memory.

In my script beta is set to 0.95 per 30 days. New orders have greater weight than old orders.

Calculate a Prior and Bayes Factors for prediction

Work in progress:
-- add bf0 to data for all products NOT reordered to all orders after first ordered
-- add the exponential time weighting - for model memory loss
-- add new factors to model
-- try flat Prior where p(reorder) is same for all products.

This file uses p(reordered|product_id) derived from order_products__prior data as a **Prior**. This is to be used in Bayesian Updating of our Prior: our_products_prior['prob_reordered']. Can also use a flat Prior.

The notion is that after calculating Bayes Factors for each test product purchase the final probability that a product will be reordered is the **Posterior** probability. Beginning when a product is first purchased (say order k of n total orders) then the **Posterior = BFn x BFn-1 x ... x BFk x Prior**.

Many others here have noticed the correlation between reordered and add_to_cart_order and aisle. I have added an engineered factor I call reorder_count (or count of reordered items in a cart). Using these three variables, I have derived a simple Augmented Naive Bayesian Network as a model to calculate the Bayes Factors for updating.

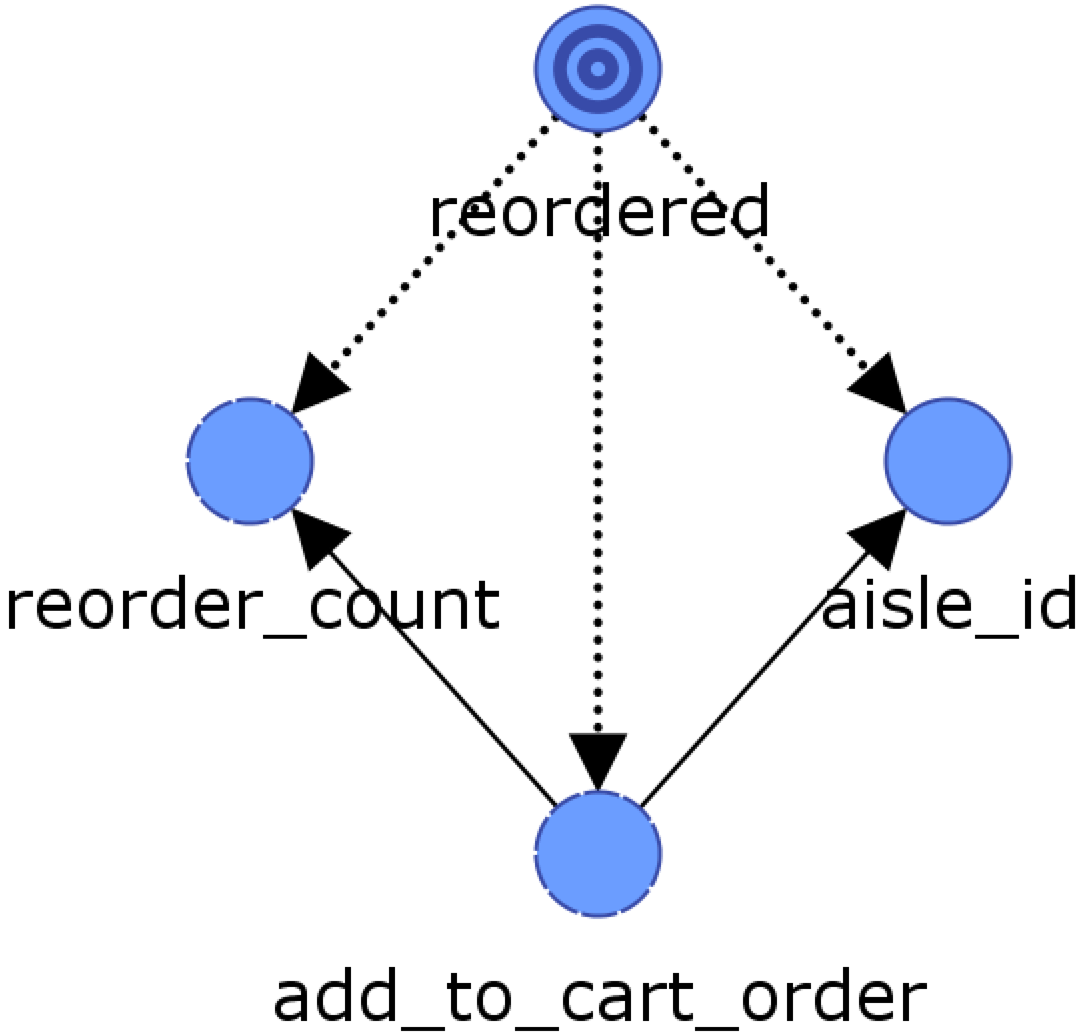


Fig-6(DAG MODEL)

2. LITERATURE REVIEW

A Bayesian Network Model for Interesting Itemsets

Jaroslav Fowkes (✉) and Charles Sutton

School of Informatics, University of Edinburgh, Edinburgh, EH8 9AB, UK
{jfowkes,csutton}@inf.ed.ac.uk

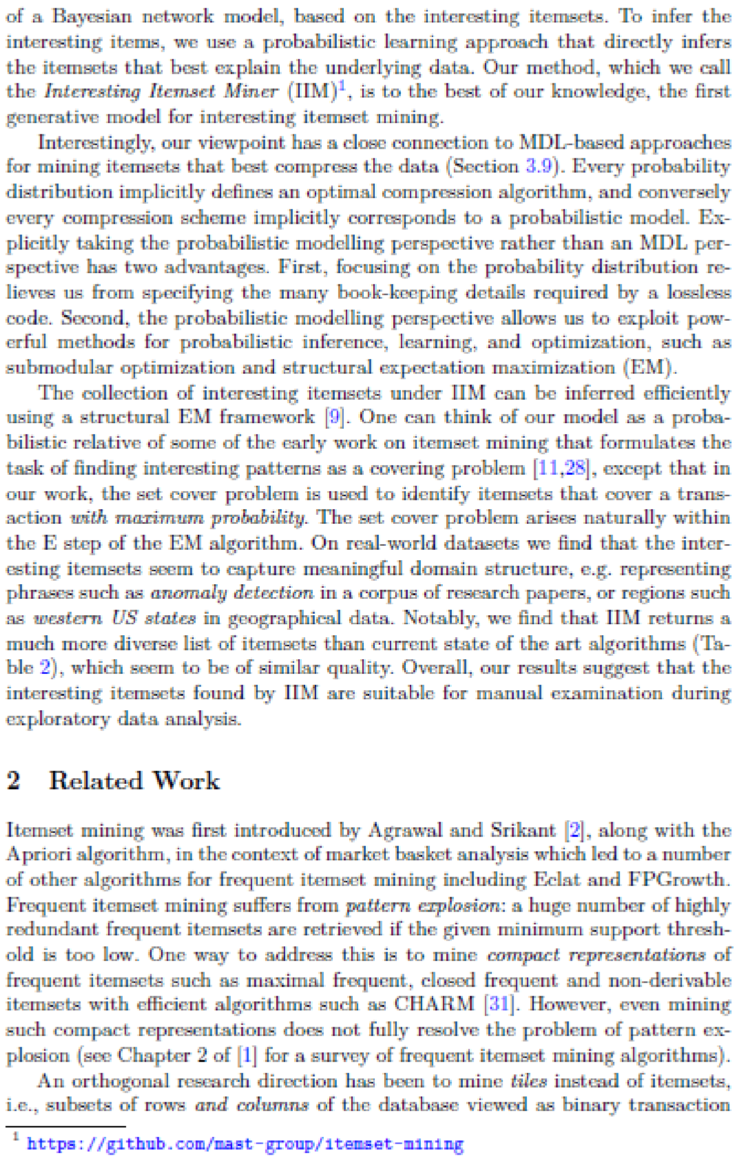
Abstract. Mining itemsets that are the most interesting under a statistical model of the underlying data is a commonly used and well-studied technique for exploratory data analysis, with the most recent interestingness models exhibiting state of the art performance. Continuing this highly promising line of work, we propose the first, to the best of our knowledge, generative model over itemsets, in the form of a Bayesian network, and an associated novel measure of interestingness. Our model is able to efficiently infer interesting itemsets directly from the transaction database using structural EM, in which the E-step employs the greedy approximation to weighted set cover. Our approach is theoretically simple, straightforward to implement, trivially parallelizable and retrieves itemsets whose quality is comparable to, if not better than, existing state of the art algorithms as we demonstrate on several real-world datasets.

1 Introduction

Itemset mining is one of the most important problems in data mining, with applications including market basket analysis, mining data streams and mining bugs in source code [1]. Early work on itemset mining focused on algorithms that identify all itemsets which meet a given criterion for pattern quality, such as all *frequent itemsets* whose support is above a user-specified threshold. Although appealing algorithmically, the list of frequent itemsets suffers from *pattern explosion*, i.e., is typically long, highly redundant and difficult to understand [1]. In an attempt to address this problem, more recent work focuses on mining *interesting itemsets*, smaller sets of high-quality, non-redundant itemsets that can be examined by a data analyst to get an overview of the data. Several different approaches have been proposed for this problem. Some of the most successful recent approaches, such as MTV [19], KRIMP [28] and SLIM [26] are based on the *minimum description length* (MDL) principle, meaning that they define an encoding scheme for compressing the database based on a set of itemsets, and search for the itemsets that best compress the data. These methods have been shown to lead to much less redundant pattern sets than frequent itemset mining.

In this paper, we introduce an alternative, but closely related, viewpoint on interesting itemset mining methods, by starting with a probabilistic model of the data rather than a compression scheme. We define a *generative model* of the data, that is, a probability distribution over the database, in the form

of the data, that is, a probability distribution over the database, in the form



of a Bayesian network model, based on the interesting itemsets. To infer the interesting items, we use a probabilistic learning approach that directly infers the itemsets that best explain the underlying data. Our method, which we call the *Interesting Itemset Miner* (IIM)[1], is to the best of our knowledge, the first generative model for interesting itemset mining.

Interestingly, our viewpoint has a close connection to MDL-based approaches for mining itemsets that best compress the data (Section 3.9). Every probability distribution implicitly defines an optimal compression algorithm, and conversely every compression scheme implicitly corresponds to a probabilistic model. Explicitly taking the probabilistic modelling perspective rather than an MDL perspective has two advantages. First, focusing on the probability distribution relieves us from specifying the many book-keeping details required by a lossless code. Second, the probabilistic modelling perspective allows us to exploit powerful methods for probabilistic inference, learning, and optimization, such as submodular optimization and structural expectation maximization (EM).

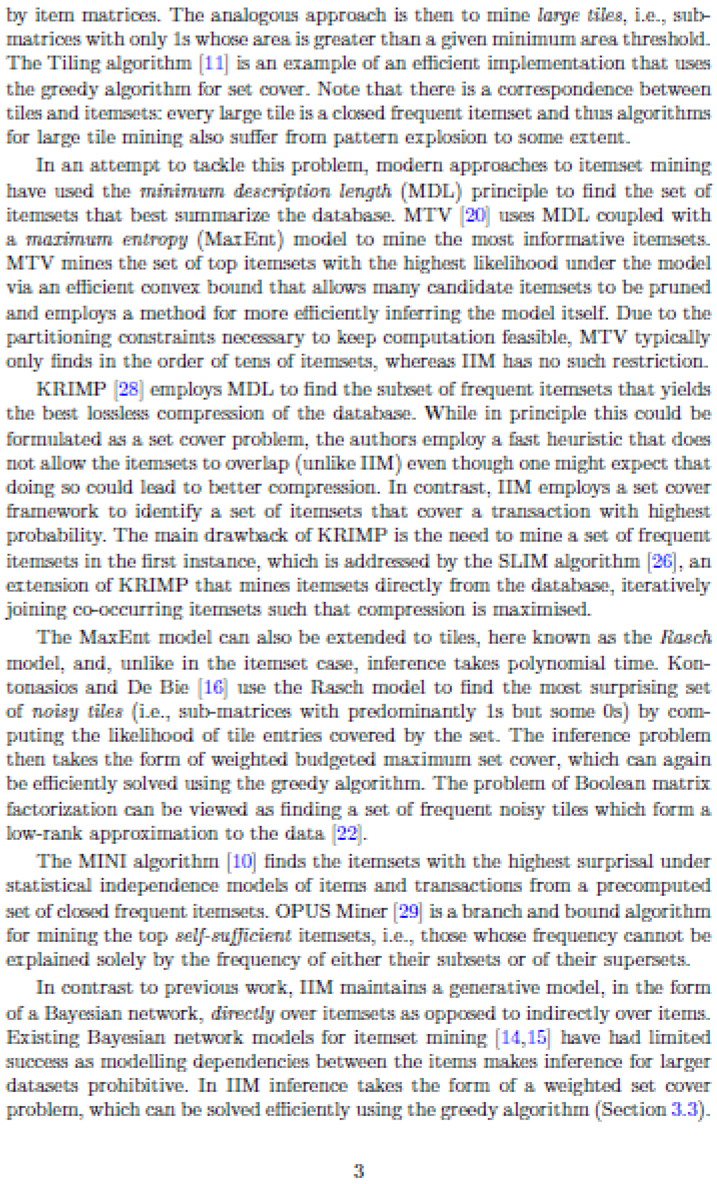
The collection of interesting itemsets under IIM can be inferred efficiently using a structural EM framework [9]. One can think of our model as a probabilistic relative of some of the early work on itemset mining that formulates the task of finding interesting patterns as a covering problem [11,28], except that in our work, the set cover problem is used to identify itemsets that cover a transaction *with maximum probability*. The set cover problem arises naturally within the E step of the EM algorithm. On real-world datasets we find that the interesting itemsets seem to capture meaningful domain structure, e.g. representing phrases such as *anomaly detection* in a corpus of research papers, or regions such as *western US states* in geographical data. Notably, we find that IIM returns a much more diverse list of itemsets than current state of the art algorithms (Table 2), which seem to be of similar quality. Overall, our results suggest that the interesting itemsets found by IIM are suitable for manual examination during exploratory data analysis.

2 Related Work

Itemset mining was first introduced by Agrawal and Srikant [2], along with the Apriori algorithm, in the context of market basket analysis which led to a number of other algorithms for frequent itemset mining including Eclat and FPGrowth. Frequent itemset mining suffers from *pattern explosion*: a huge number of highly redundant frequent itemsets are retrieved if the given minimum support threshold is too low. One way to address this is to mine *compact representations* of frequent itemsets such as maximal frequent, closed frequent and non-derivable itemsets with efficient algorithms such as CHARM [31]. However, even mining such compact representations does not fully resolve the problem of pattern explosion (see Chapter 2 of [1] for a survey of frequent itemset mining algorithms).

An orthogonal research direction has been to mine *tiles* instead of itemsets, i.e., subsets of rows *and columns* of the database viewed as binary transaction

[1] https://github.com/mast-group/itemset-mining



by item matrices. The analogous approach is then to mine *large tiles*, i.e., submatrices with only 1s whose area is greater than a given minimum area threshold. The Tiling algorithm [11] is an example of an efficient implementation that uses the greedy algorithm for set cover. Note that there is a correspondence between tiles and itemsets: every large tile is a closed frequent itemset and thus algorithms for large tile mining also suffer from pattern explosion to some extent.

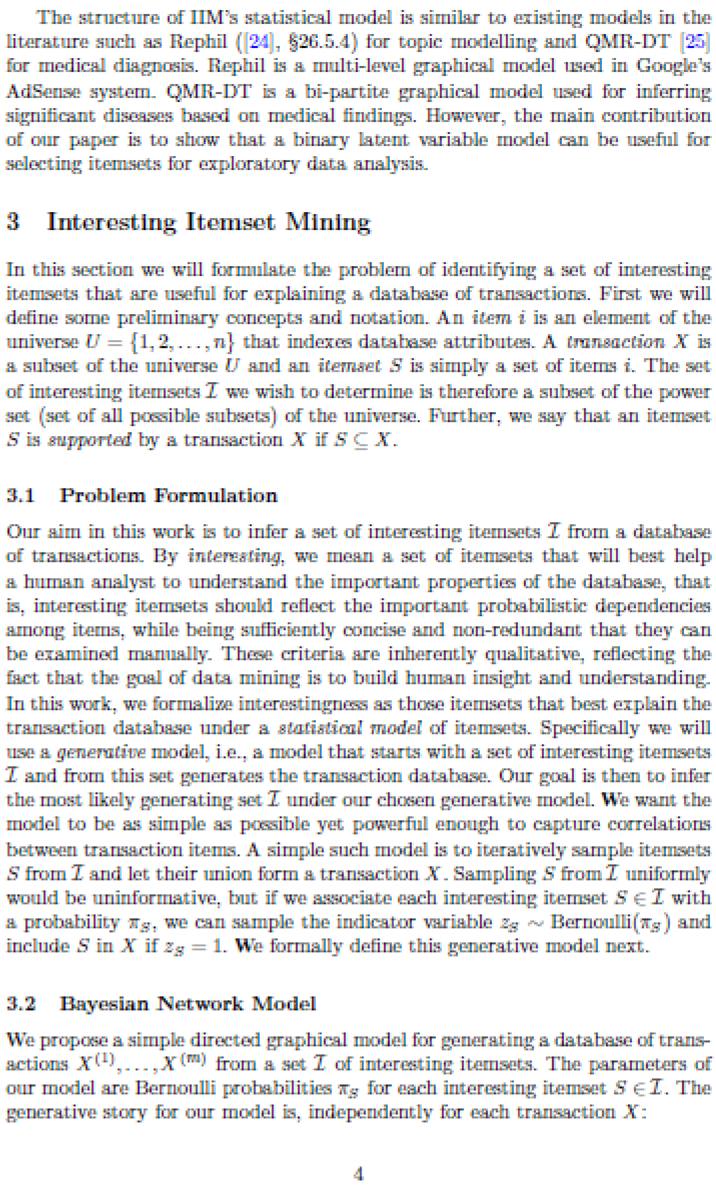
In an attempt to tackle this problem, modern approaches to itemset mining have used the *minimum description length* (MDL) principle to find the set of itemsets that best summarize the database. MTV [20] uses MDL coupled with a *maximum entropy* (MaxEnt) model to mine the most informative itemsets. MTV mines the set of top itemsets with the highest likelihood under the model via an efficient convex bound that allows many candidate itemsets to be pruned and employs a method for more efficiently inferring the model itself. Due to the partitioning constraints necessary to keep computation feasible, MTV typically only finds in the order of tens of itemsets, whereas IIM has no such restriction.

KRIMP [28] employs MDL to find the subset of frequent itemsets that yields the best lossless compression of the database. While in principle this could be formulated as a set cover problem, the authors employ a fast heuristic that does not allow the itemsets to overlap (unlike IIM) even though one might expect that doing so could lead to better compression. In contrast, IIM employs a set cover framework to identify a set of itemsets that cover a transaction with highest probability. The main drawback of KRIMP is the need to mine a set of frequent itemsets in the first instance, which is addressed by the SLIM algorithm [26], an extension of KRIMP that mines itemsets directly from the database, iteratively joining co-occurring itemsets such that compression is maximised.

The MaxEnt model can also be extended to tiles, here known as the *Rasch* model, and, unlike in the itemset case, inference takes polynomial time. Kontonasios and De Bie [16] use the Rasch model to find the most surprising set of *noisy tiles* (i.e., sub-matrices with predominantly 1s but some 0s) by computing the likelihood of tile entries covered by the set. The inference problem then takes the form of weighted budgeted maximum set cover, which can again be efficiently solved using the greedy algorithm. The problem of Boolean matrix factorization can be viewed as finding a set of frequent noisy tiles which form a low-rank approximation to the data [22].

The MINI algorithm [10] finds the itemsets with the highest surprisal under statistical independence models of items and transactions from a precomputed set of closed frequent itemsets. OPUS Miner [29] is a branch and bound algorithm for mining the top *self-sufficient* itemsets, i.e., those whose frequency cannot be explained solely by the frequency of either their subsets or of their supersets.

In contrast to previous work, IIM maintains a generative model, in the form of a Bayesian network, *directly* over itemsets as opposed to indirectly over items. Existing Bayesian network models for itemset mining [14,15] have had limited success as modelling dependencies between the items makes inference for larger datasets prohibitive. In IIM inference takes the form of a weighted set cover problem, which can be solved efficiently using the greedy algorithm (Section 3.3).



The structure of IIM's statistical model is similar to existing models in the literature such as Rephil ([24], §26.5.4) for topic modelling and QMR-DT [25] for medical diagnosis. Rephil is a multi-level graphical model used in Google's AdSense system. QMR-DT is a bi-partite graphical model used for inferring significant diseases based on medical findings. However, the main contribution of our paper is to show that a binary latent variable model can be useful for selecting itemsets for exploratory data analysis.

3 Interesting Itemset Mining

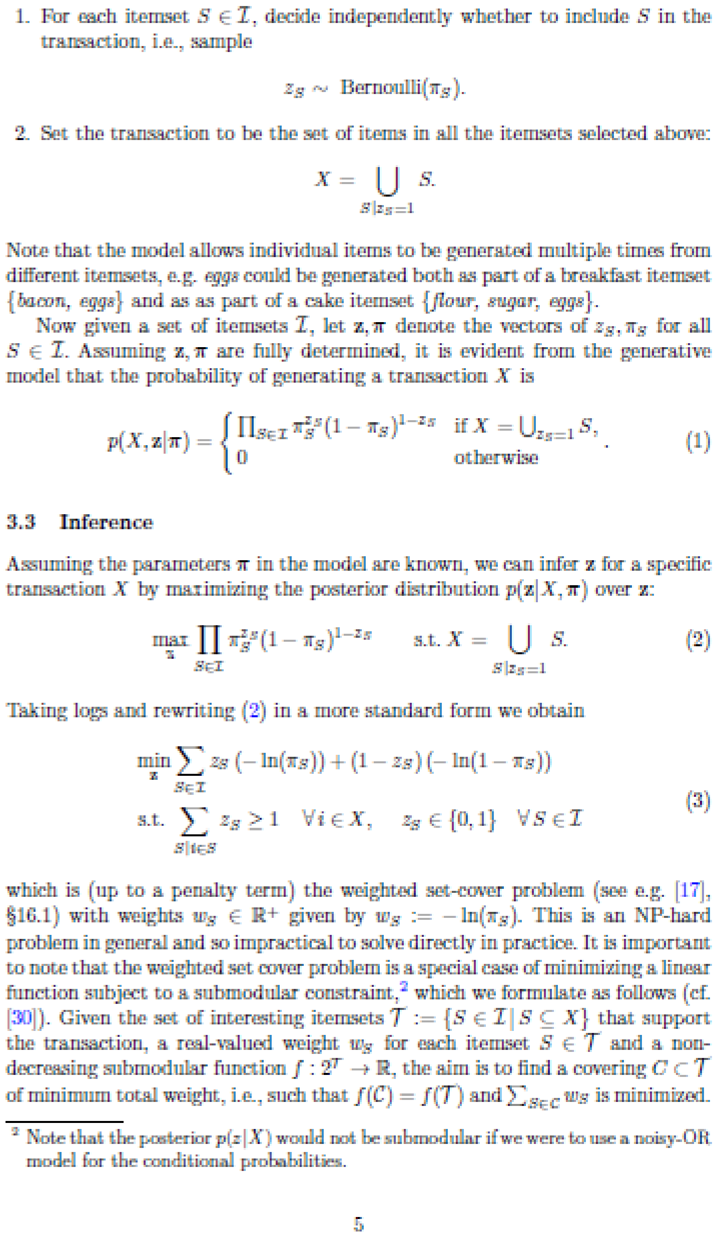
In this section we will formulate the problem of identifying a set of interesting itemsets that are useful for explaining a database of transactions. First we will define some preliminary concepts and notation. An *item* i is an element of the universe $U = \{1, 2, \ldots, n\}$ that indexes database attributes. A *transaction* X is a subset of the universe U and an *itemset* S is simply a set of items i. The set of interesting itemsets $\mathcal{I}$ we wish to determine is therefore a subset of the power set (set of all possible subsets) of the universe. Further, we say that an itemset S is *supported* by a transaction X if $S \subseteq X$.

3.1 Problem Formulation

Our aim in this work is to infer a set of interesting itemsets $\mathcal{I}$ from a database of transactions. By *interesting*, we mean a set of itemsets that will best help a human analyst to understand the important properties of the database, that is, interesting itemsets should reflect the important probabilistic dependencies among items, while being sufficiently concise and non-redundant that they can be examined manually. These criteria are inherently qualitative, reflecting the fact that the goal of data mining is to build human insight and understanding. In this work, we formalize interestingness as those itemsets that best explain the transaction database under a *statistical model* of itemsets. Specifically we will use a *generative* model, i.e., a model that starts with a set of interesting itemsets $\mathcal{I}$ and from this set generates the transaction database. Our goal is then to infer the most likely generating set $\mathcal{I}$ under our chosen generative model. We want the model to be as simple as possible yet powerful enough to capture correlations between transaction items. A simple such model is to iteratively sample itemsets S from $\mathcal{I}$ and let their union form a transaction X. Sampling S from $\mathcal{I}$ uniformly would be uninformative, but if we associate each interesting itemset $S \in \mathcal{I}$ with a probability π_S, we can sample the indicator variable $z_S \sim \text{Bernoulli}(\pi_S)$ and include S in X if $z_S = 1$. We formally define this generative model next.

3.2 Bayesian Network Model

We propose a simple directed graphical model for generating a database of transactions $X^{(1)}, \ldots, X^{(m)}$ from a set $\mathcal{I}$ of interesting itemsets. The parameters of our model are Bernoulli probabilities π_S for each interesting itemset $S \in \mathcal{I}$. The generative story for our model is, independently for each transaction X:



1. For each itemset $S \in \mathcal{I}$, decide independently whether to include S in the transaction, i.e., sample
$$z_S \sim \text{ Bernoulli}(\pi_S).$$
2. Set the transaction to be the set of items in all the itemsets selected above:
$$X = \bigcup_{S \mid z_S = 1} S.$$

Note that the model allows individual items to be generated multiple times from different itemsets, e.g. *eggs* could be generated both as part of a breakfast itemset $\{$*bacon, eggs*$\}$ and as as part of a cake itemset $\{$*flour, sugar, eggs*$\}$.

Now given a set of itemsets $\mathcal{I}$, let $\mathbf{z}, \boldsymbol{\pi}$ denote the vectors of z_S, π_S for all $S \in \mathcal{I}$. Assuming $\mathbf{z}, \boldsymbol{\pi}$ are fully determined, it is evident from the generative model that the probability of generating a transaction X is

$$p(X, \mathbf{z} | \boldsymbol{\pi}) = \begin{cases} \prod_{S \in \mathcal{I}} \pi_S^{z_S} (1 - \pi_S)^{1 - z_S} & \text{if } X = \bigcup_{z_S = 1} S, \\ 0 & \text{otherwise} \end{cases}. \tag{1}$$

3.3 Inference

Assuming the parameters $\boldsymbol{\pi}$ in the model are known, we can infer $\mathbf{z}$ for a specific transaction X by maximizing the posterior distribution $p(\mathbf{z} | X, \boldsymbol{\pi})$ over $\mathbf{z}$:

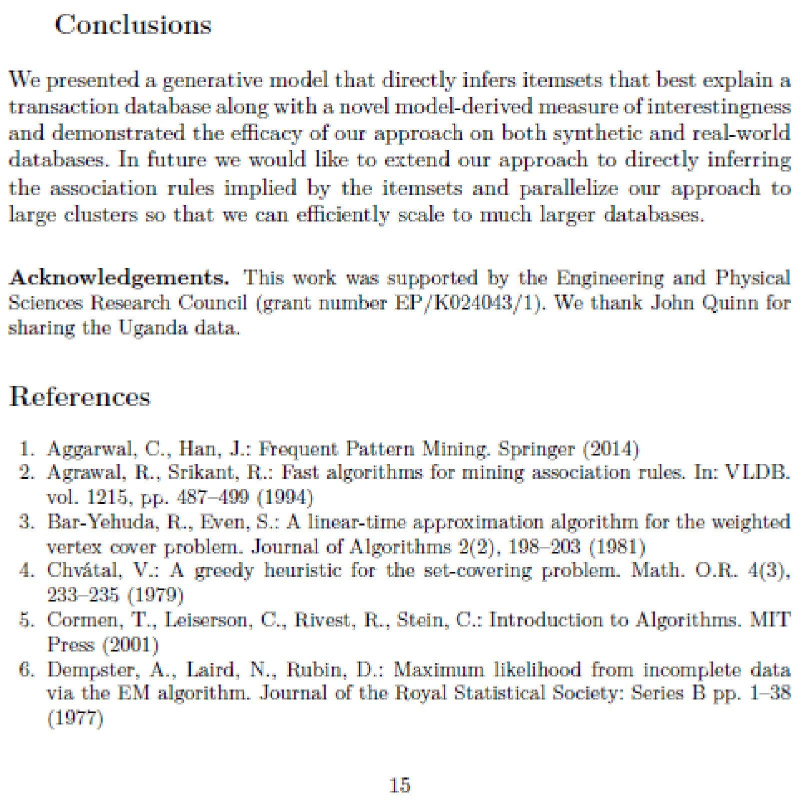
$$\max_{\mathbf{z}} \prod_{S \in \mathcal{I}} \pi_S^{z_S} (1 - \pi_S)^{1 - z_S} \qquad \text{s.t. } X = \bigcup_{S \mid z_S = 1} S. \tag{2}$$

Taking logs and rewriting (2) in a more standard form we obtain

$$\begin{aligned} &\min_{\mathbf{z}} \sum_{S \in \mathcal{I}} z_S \left(-\ln(\pi_S)\right) + (1 - z_S)\left(-\ln(1 - \pi_S)\right) \\ &\text{s.t.} \sum_{S \mid i \in S} z_S \geq 1 \quad \forall\, i \in X, \quad z_S \in \{0, 1\} \quad \forall\, S \in \mathcal{I} \end{aligned} \tag{3}$$

which is (up to a penalty term) the weighted set-cover problem (see e.g. [17], §16.1) with weights $w_S \in \mathbb{R}^{+}$ given by $w_S := -\ln(\pi_S)$. This is an NP-hard problem in general and so impractical to solve directly in practice. It is important to note that the weighted set cover problem is a special case of minimizing a linear function subject to a submodular constraint,[2] which we formulate as follows (cf. [30]). Given the set of interesting itemsets $\mathcal{T} := \{S \in \mathcal{I} \mid S \subseteq X\}$ that support the transaction, a real-valued weight w_S for each itemset $S \in \mathcal{T}$ and a non-decreasing submodular function $f : 2^{\mathcal{T}} \to \mathbb{R}$, the aim is to find a covering $C \subset \mathcal{T}$ of minimum total weight, i.e., such that $f(\mathcal{C}) = f(\mathcal{T})$ and $\sum_{S \in \mathcal{C}} w_S$ is minimized.

[2] Note that the posterior $p(z|X)$ would not be submodular if we were to use a noisy-OR model for the conditional probabilities.



Conclusions

We presented a generative model that directly infers itemsets that best explain a transaction database along with a novel model-derived measure of interestingness and demonstrated the efficacy of our approach on both synthetic and real-world databases. In future we would like to extend our approach to directly inferring the association rules implied by the itemsets and parallelize our approach to large clusters so that we can efficiently scale to much larger databases.

Acknowledgements. This work was supported by the Engineering and Physical Sciences Research Council (grant number EP/K024043/1). We thank John Quinn for sharing the Uganda data.

References

1. Aggarwal, C., Han, J.: Frequent Pattern Mining. Springer (2014)
2. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: VLDB. vol. 1215, pp. 487–499 (1994)
3. Bar-Yehuda, R., Even, S.: A linear-time approximation algorithm for the weighted vertex cover problem. Journal of Algorithms 2(2), 198–203 (1981)
4. Chvátal, V.: A greedy heuristic for the set-covering problem. Math. O.R. 4(3), 233–235 (1979)
5. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms. MIT Press (2001)
6. Dempster, A., Laird, N., Rubin, D.: Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society: Series B pp. 1–38 (1977)

3. METHODOLOGY

As PYTHON is the one responsible of all the machine learning algorithms all this process was done on its servers, we didn’t need hardware for it. However, for the feature engineering task, we needed it.

The software used in this project was the programming language Python. In addition, we used one of the most powerful libraries used nowadays for Data Science, Pandas. The environment to programme used was Jupyter Notebook.

The environmental specification specifies the hardware and software requirements for carrying out this project.
The following are the hardware and the software requirements.

Software

- Software Name: JUPYTER NOTEBOOK
- Windows XP Service Pack 2 / Windows 7
- Installed memory (RAM): high requirement(RAM)
- System type: 64-bit Operating System, x64 based processor

Hardware

- Windows Edition: Windows 7 PROFESSIONAL
- Processor: Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz 2.40GHz
- Installed memory (RAM): 8.00GB
- System type: 64-bit Operating System, x64 based processor

4.Design Details

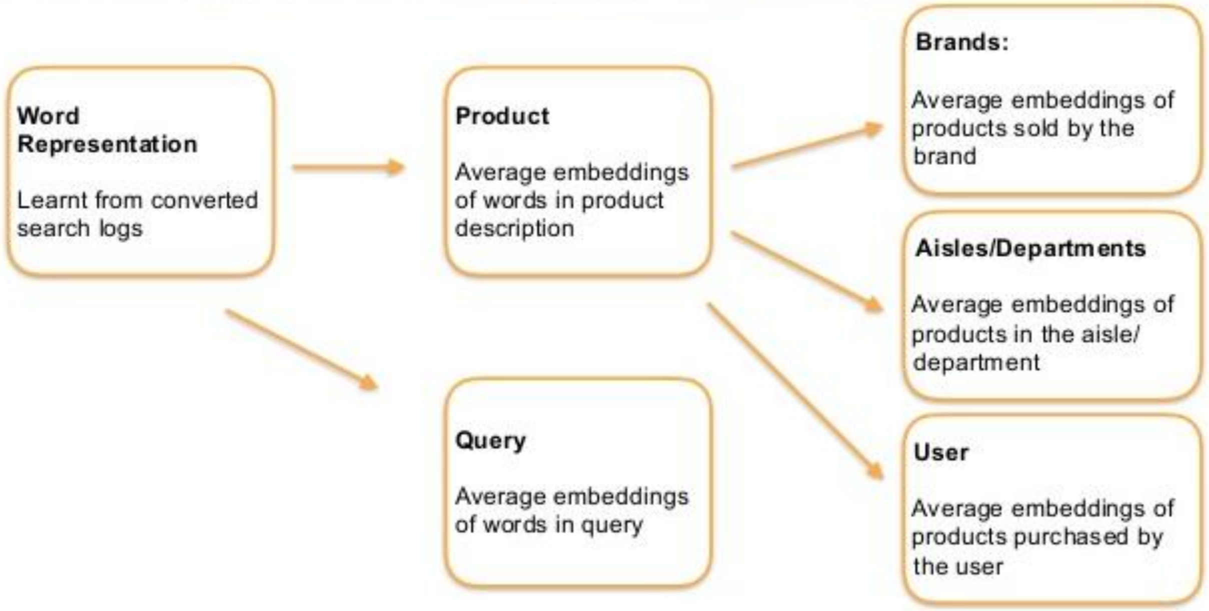
4.1Dataset Source

Instacart is an American company that operates as a same-day grocery delivery service. Customers select groceries through a web application from various retailers and delivered by a personal shopper. Up until the end of 2017, Instacart only had operations and services in the United States. In November 2017, the company announced plans to begin delivery in Toronto and Vancouve

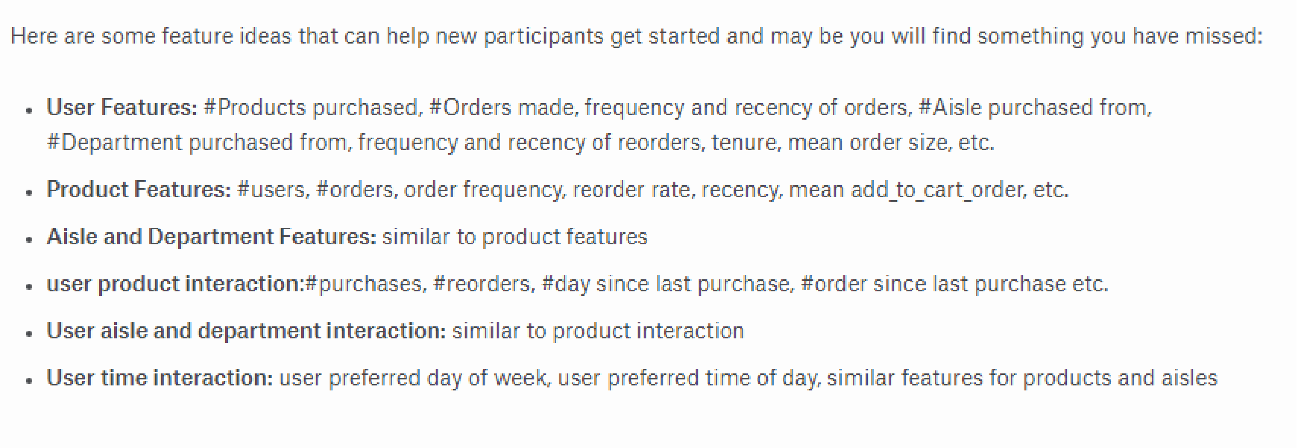
Whether you shop from meticulously planned grocery lists or let whimsy guide your grazing, our unique food rituals define who we are. Instacart, a grocery ordering and delivery app, aims to make it easy to fill your refrigerator and pantry with your personal favorites and staples when you need them. After selecting products through the Instacart app, personal shoppers review your order and do the in-store shopping and delivery for you.

Instacart's data science team plays a big part in providing this delightful shopping experience. Currently they use transactional data to develop models that predict which products a user will buy again, try for the first time, or add to their cart next during a session. Recently, Instacart open sourced this data 3 Million Instacart Orders, Open Sourced.

Instacart is challenging the Kaggle community to use this anonymized data on customer orders over time to predict which previously purchased products will be in a user's next order. They're not only looking for the best model, Instacart's also looking for machine learning engineers to grow their team


Word Representation
Learnt from converted search logs
Product
Average embeddings of words in product description
Query
Average embeddings of words in query
Brands:
Average embeddings of products sold by the brand
Aisles/Departments
Average embeddings of products in the aisle/ department
User
Average embeddings of products purchased by the user

Dataset Tables



Here are some feature ideas that can help new participants get started and may be you will find something you have missed:

- **User Features:** #Products purchased, #Orders made, frequency and recency of orders, #Aisle purchased from, #Department purchased from, frequency and recency of reorders, tenure, mean order size, etc.
- **Product Features:** #users, #orders, order frequency, reorder rate, recency, mean add_to_cart_order, etc.
- **Aisle and Department Features:** similar to product features
- **user product interaction:**#purchases, #reorders, #day since last purchase, #order since last purchase etc.
- **User aisle and department interaction:** similar to product interaction
- **User time interaction:** user preferred day of week, user preferred time of day, similar features for products and aisles

Aisles Dataset

aisle_id	aisle
1	prepared soups salads
2	specialty cheeses
3	energy granola bars
4	instant foods
5	marinades meat preparation
6	other
7	packaged meat
8	bakery desserts
9	pasta sauce
10	kitchen supplies
11	cold flu allergy
12	fresh pasta
13	prepared meals
14	tofu meat alternatives
15	packaged seafood
16	fresh herbs
17	baking ingredients
18	bulk dried fruits vegetables
19	oils vinegars
20	oral hygiene
21	packaged cheese

22	hair care
23	popcorn jerky
24	fresh fruits
25	soap
26	coffee
27	beers coolers
28	red wines
29	honeys syrups nectars
30	latino foods
31	Refrigerated
32	packaged produce
33	kosher foods
34	frozen meat seafood
35	poultry counter
36	Butter
37	ice cream ice
38	frozen meals
39	seafood counter
40	dog food car
41	cat food care
42	frozen vegan vegetarian
43	buns rolls
44	eye ear care
45	candy chocolate
46	mint gum
47	vitamins supplements
48	breakfast bars pastries
49	packaged poultry
50	fruit vegetable snacks
51	preserved dips spreads
52	frozen breakfast
53	cream
54	paper goods
55	shave needs
56	diapers wipes
57	granola
58	frozen breads doughs

Departments Dataset

department_id	Department
1	Frozen
2	Other
3	Bakery
4	Produce
5	Alcohol
6	International
7	Beverages
8	Pets
9	dry goods pasta
10	Bulk
11	personal care
12	meat seafood
13	Pantry
14	Breakfast
15	canned goods
16	dairy eggs
17	Household
18	Babies
19	Snacks
20	Deli
21	Missing

order_products__prior DataSet

order_id	product_id	add_to_cart_order	Reordered
2	33120	1	1
2	28985	2	1
2	9327	3	0
2	45918	4	1
2	30035	5	0
2	17794	6	1
2	40141	7	1
2	1819	8	1
2	43668	9	0
3	33754	1	1
3	24838	2	1
3	17704	3	1
3	21903	4	1
3	17668	5	1
3	46667	6	1
3	17461	7	1
3	32665	8	1
4	46842	1	0
4	26434	2	1
4	39758	3	1
4	27761	4	1
4	10054	5	1
4	21351	6	1
4	22598	7	1
4	34862	8	1
4	40285	9	1
4	17616	10	1
4	25146	11	1
4	32645	12	1
4	41276	13	1
5	13176	1	1
5	15005	2	1
5	47329	3	1
5	27966	4	1

Order Products-train Dataset

order_id	product_id	add_to_cart_order	reordered
1	49302	1	1
1	11109	2	1
1	10246	3	0
1	49683	4	0
1	43633	5	1
1	13176	6	0
1	47209	7	0
1	22035	8	1
36	39612	1	0
36	19660	2	1
36	49235	3	0
36	43086	4	1
36	46620	5	1
36	34497	6	1
36	48679	7	1
36	46979	8	1
38	11913	1	0
38	18159	2	0
38	4461	3	0
38	21616	4	1
38	23622	5	0
38	32433	6	0
38	28842	7	0
38	42625	8	0
38	39693	9	0
96	20574	1	1
96	30391	2	0
96	40706	3	1
96	25610	4	0
96	27966	5	1
96	24489	6	1
96	39275	7	1
98	8859	1	1
98	19731	2	1
98	43654	3	1

Orders Dataset

order_id	user_id	eval_set	order_number	order_dow	order-hour-of-day	days-since-prior-order
2539329	1	prior	1	2	8	
2398795	1	prior	2	3	7	15
473747	1	prior	3	3	12	21
2254736	1	prior	4	4	7	29
431534	1	prior	5	4	15	28
3367565	1	prior	6	2	7	19
550135	1	prior	7	1	9	20
3108588	1	prior	8	1	14	14
2295261	1	prior	9	1	16	0
2550362	1	prior	10	4	8	30
1187899	1	train	11	4	8	14
2168274	2	prior	1	2	11	
1501582	2	prior	2	5	10	10
1901567	2	prior	3	1	10	3
738281	2	prior	4	2	10	8
1673511	2	prior	5	3	11	8
1199898	2	prior	6	2	9	13
3194192	2	prior	7	2	12	14
788338	2	prior	8	1	15	27
1718559	2	prior	9	2	9	8
1447487	2	prior	10	1	11	6
1402090	2	prior	11	1	10	30
3186735	2	prior	12	1	9	28
3268552	2	prior	13	4	11	30
1447487	2	prior	10	1	11	6
1402090	2	prior	11	1	10	30
3186735	2	prior	12	1	9	28
3268552	2	prior	13	4	11	30
839880	2	prior	14	3	10	13
1492625	2	train	15	1	11	30
1374495	3	prior	1	1	14	
444309	3	prior	2	3	19	9
3002854	3	prior	3	3	16	21
2037211	3	prior	4	2	18	20

Product Dataset

product_id	product_name	aisle_id	department_id
1	Chocolate Sandwich Cookies	61	19
2	All-Seasons Salt	104	13
3	Robust Golden Unsweetened Oolong Tea	94	7
4	Smart Ones Classic Favorites Mini Rigatoni With Vodka Cream Sauce	38	1
5	Green Chile Anytime Sauce	5	13
6	Dry Nose Oil	11	11
7	Pure Coconut Water With Orange	98	7
8	Cut Russet Potatoes Steam N' Mash	116	1
9	Light Strawberry Blueberry Yogurt	120	16
10	Sparkling Orange Juice & Prickly Pear Beverage	115	7
11	Peach Mango Juice	31	7
12	Chocolate Fudge Layer Cake	119	1
13	Saline Nasal Mist	11	11
14	Fresh Scent Dishwasher Cleaner	74	17
15	Overnight Diapers Size 6	56	18
16	Mint Chocolate Flavored Syrup	103	19
17	Rendered Duck Fat	35	12
18	Pizza for One Suprema Frozen Pizza	79	1
19	Gluten Free Quinoa Three Cheese & Mushroom Blend	63	9
20	Pomegranate Cranberry & Aloe Vera Enrich Drink	98	7
21	Small & Medium Dental Dog Treats	40	8
22	Fresh Breath Oral Rinse Mild Mint	20	11
23	Organic Turkey Burgers	49	12
24	Tri-Vi-SolÂ® Vitamins A-C-and D Supplement Drops for Infants	47	11
25	Salted Caramel Lean Protein & Fiber Bar	3	19
26	Fancy Feast Trout Feast Flaked Wet Cat Food	41	8
27	Complete Spring Water Foaming Antibacterial Hand Wash	127	11
28	Wheat Chex Cereal	121	14
29	Fresh Cut Golden Sweet No Salt Added Whole Kernel Corn	81	15
30	Three Cheese Ziti, Marinara with Meatballs	38	1
31	White Pearl Onions	123	4
32	Nacho Cheese White Bean Chips	107	19
33	Organic Spaghetti Style Pasta	131	9
34	Peanut Butter Cereal	121	14
35	Italian Herb Porcini Mushrooms Chicken Sausage	106	12
36	Traditional Lasagna with Meat Sauce Savory Italian Recipes	38	1

4.2 Project Implementation Code

This project aims to predict which previously purchased products will be in a user’s next order.

```
import pandas as pd
import numpy as np
import warnings # current version of seaborn generates a bunch of warnings that will be
ignore
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
import seaborn as sns
color = sns.color_palette()
%matplotlib inline
pd.options.mode.chained_assignment = None  # default='warn'
import operator
# reading data
# directory = 'instacart_2017_05_01/'
directory = './instacart_2017_05_01/'
print('Loading prior orders')
prior_orders = pd.read_csv(directory + 'order_products__prior.csv', dtype={
        'order_id': np.int32,
        'product_id': np.int32,
        'add_to_cart_order': np.int16,
        'reordered': np.int8} , nrows=652000 )
print('Loading orders')
orders = pd.read_csv(directory + 'orders.csv', dtype={
        'order_id': np.int32,
        'user_id': np.int32,
        'eval_set': 'category',        'order_number': np.int16,
        'order_dow': np.int8,
        'order_hour_of_day': np.int8,
        'days_since_prior_order': np.float32}, nrows=68000)
print('Loading aisles info')
aisles = pd.read_csv(directory + 'products.csv', engine='c',
                           usecols = ['product_id','aisle_id'],
                       dtype={'product_id': np.int32, 'aisle_id': np.int32}, nrows=1000)
pd.set_option('display.float_format', lambda x: '%.3f' % x)
print("\n Checking the loaded CSVs")
print("Prior orders:", prior_orders.shape)
print("Orders", orders.shape)
print("Aisles:", aisles.shape)
```

out:

```
Loading prior orders
Loading orders
Loading aisles info

 Checking the loaded CSVs
('Prior orders:', (652000, 4))
('Orders', (68000, 7))
('Aisles:', (1000, 2))
```

```
import os
os.getcwd()
```

out:

```
'C:\\Users\\priyanka\\Desktop\\InstacartProductPrediction-master'
```

```
os.chdir('C:\\Users\\priyanka\\Desktop\\InstacartProductPrediction-master')

os.getcwd()
```

out:

```
'C:\\Users\\priyanka\\Desktop\\InstacartProductPrediction-master'
```

```
os.listdir('C:\\Users\\priyanka\\Desktop\\InstacartProductPrediction-master')
```

out:

```
['.ipynb_checkpoints',
 'bayes.log',
 'CustomerRankingAndPrediction.ipynb',
 'instacart_2017_05_01',
 'order_products__prior.csv',
 'order_products__train.csv',
 'predictions.csv',
 'README.md']
```

```
predictions=pd.read_csv
order_products_train_df= pd.read_csv('order_products__train.csv')    # Supply the file name (path)
```

Visualizing tables

```
order_products_train_df.head(6)
```

out:

	order_id	product_id	add_to_cart_order	reordered
0	1	49302	1	1
1	1	11109	2	1
2	1	10246	3	0
3	1	49683	4	0
4	1	43633	5	1
5	1	13176	6	0

```
prior_orders.head()
```

out:

order_id	product_id	add_to_cart_order	Reordered	
0	2	33120	1	1
1	2	28985	2	1
2	2	9327	3	0
3	2	45918	4	1
4	2	30035	5	0

orders.head()

output:

order_id	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior_order	
0	*2539329*	*1*	*prior*	*1*	*2*	*8*	*nan*
1	*2398795*	*1*	*prior*	*2*	*3*	*7*	*15.000*
2	*473747*	*1*	*prior*	*3*	*3*	*12*	*21.000*
3	*2254736*	*1*	*prior*	*4*	*4*	*7*	*29.000*
4	*431534*	*1*	*prior*	*5*	*4*	*15*	*28.000*

aisles.head()

out:

	product_id	aisle_id
0	*1*	*61*
1	*2*	*104*
2	*3*	*94*
3	*4*	*38*
4	*5*	*5*

```
cnt_srs = orders.eval_set.value_counts()
plt.figure(figsize=(12,8))
sns.barplot(cnt_srs.index, cnt_srs.values, alpha=0.8, color=color[1])
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Eval set type', fontsize=12)
plt.title('Count of rows in each dataset', fontsize=15)
plt.xticks(rotation='vertical')
plt.show()
```

out:

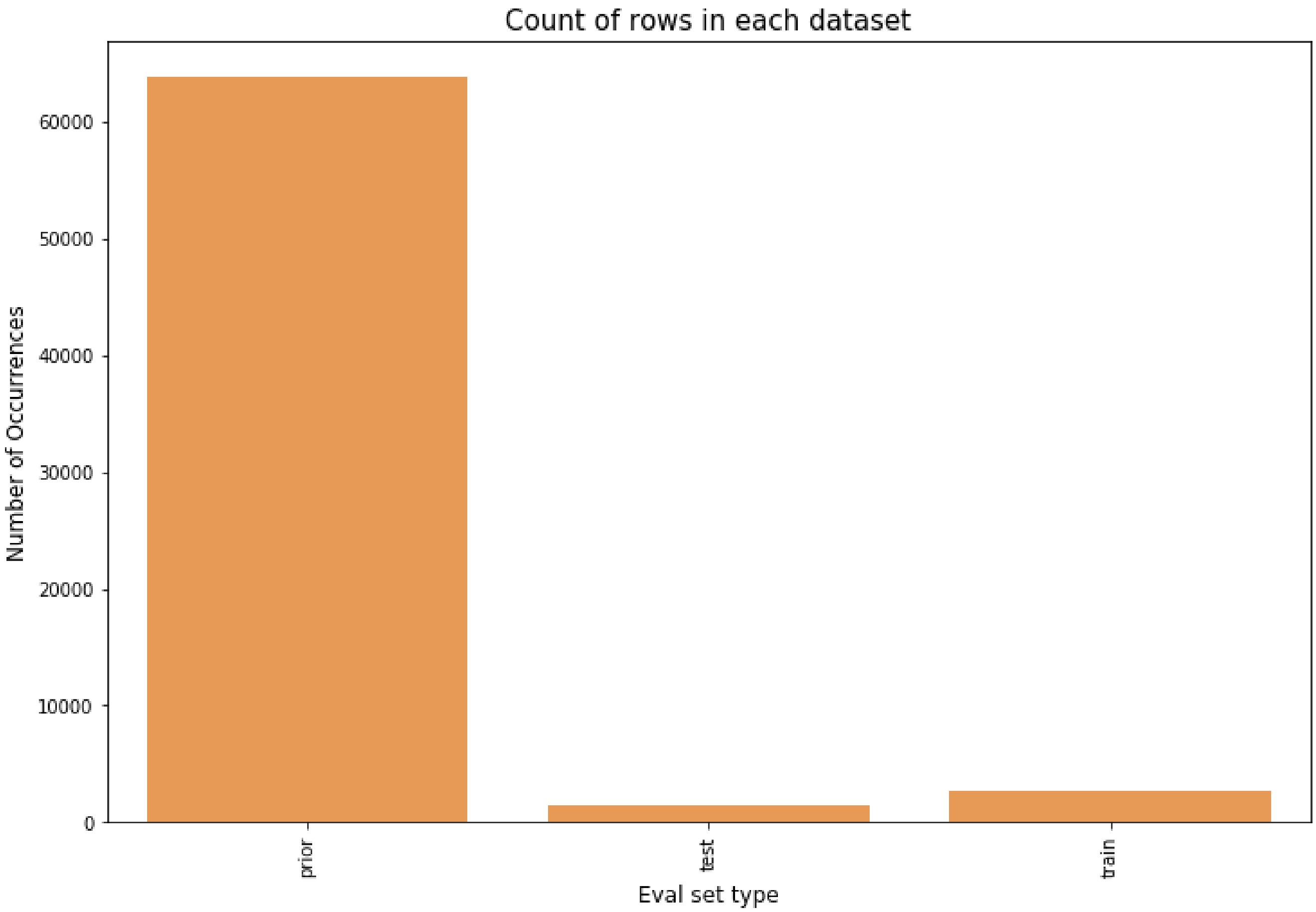


Fig-7(Count of Rows in Each dataset)

```
def get_unique_count(x):
    return len(np.unique(x))
cnt_srs = orders.groupby("eval_set")["user_id"].aggregate(get_unique_count)
cnt_srs
```

out:

```
eval_set
prior    4215
test     1492
train    2722
Name: user_id, dtype: int32
```

```
cnt_srs = orders.groupby("user_id")["order_number"].aggregate(np.max).reset_index()
cnt_srs = cnt_srs.order_number.value_counts()

plt.figure(figsize=(12,8))
sns.barplot(cnt_srs.index, cnt_srs.values, alpha=0.8, color=color[2])
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Maximum order number', fontsize=12)
plt.xticks(rotation='vertical')
plt.show()
```

out:

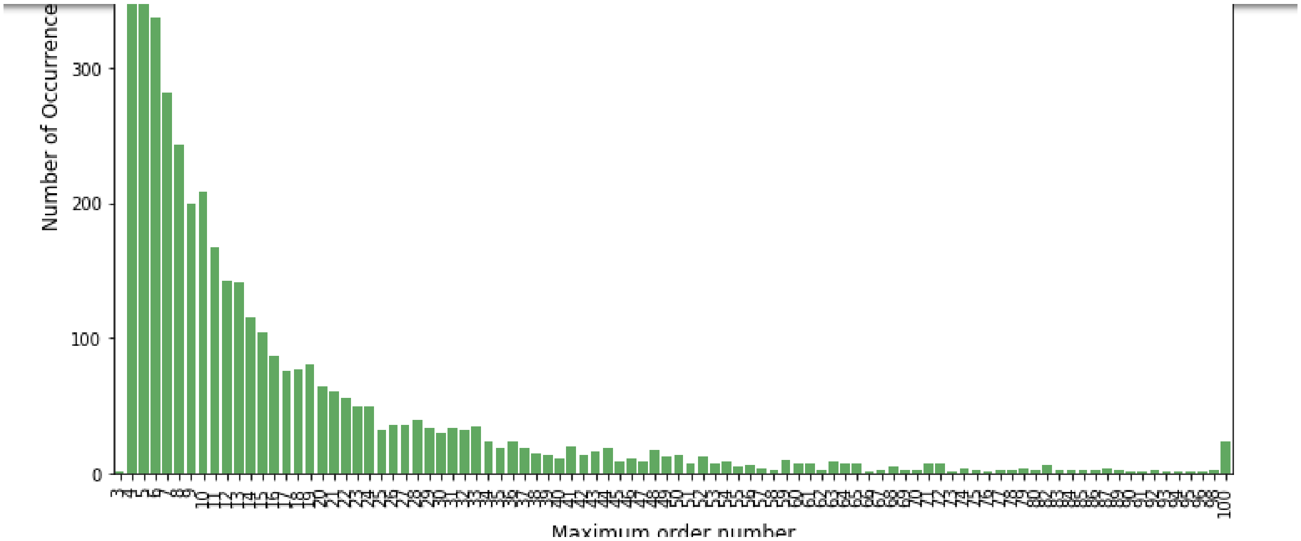


Fig-8(Graph for Maximum Order Number Per Occurrences)

```
plt.figure(figsize=(12,8))
sns.countplot(x="order_dow", data=orders, color=color[0])
plt.ylabel('Count', fontsize=12)
plt.xlabel('Day of week', fontsize=12)
plt.xticks(rotation='vertical')
plt.title("Frequency of order by week day", fontsize=15)
plt.show()
```

out:

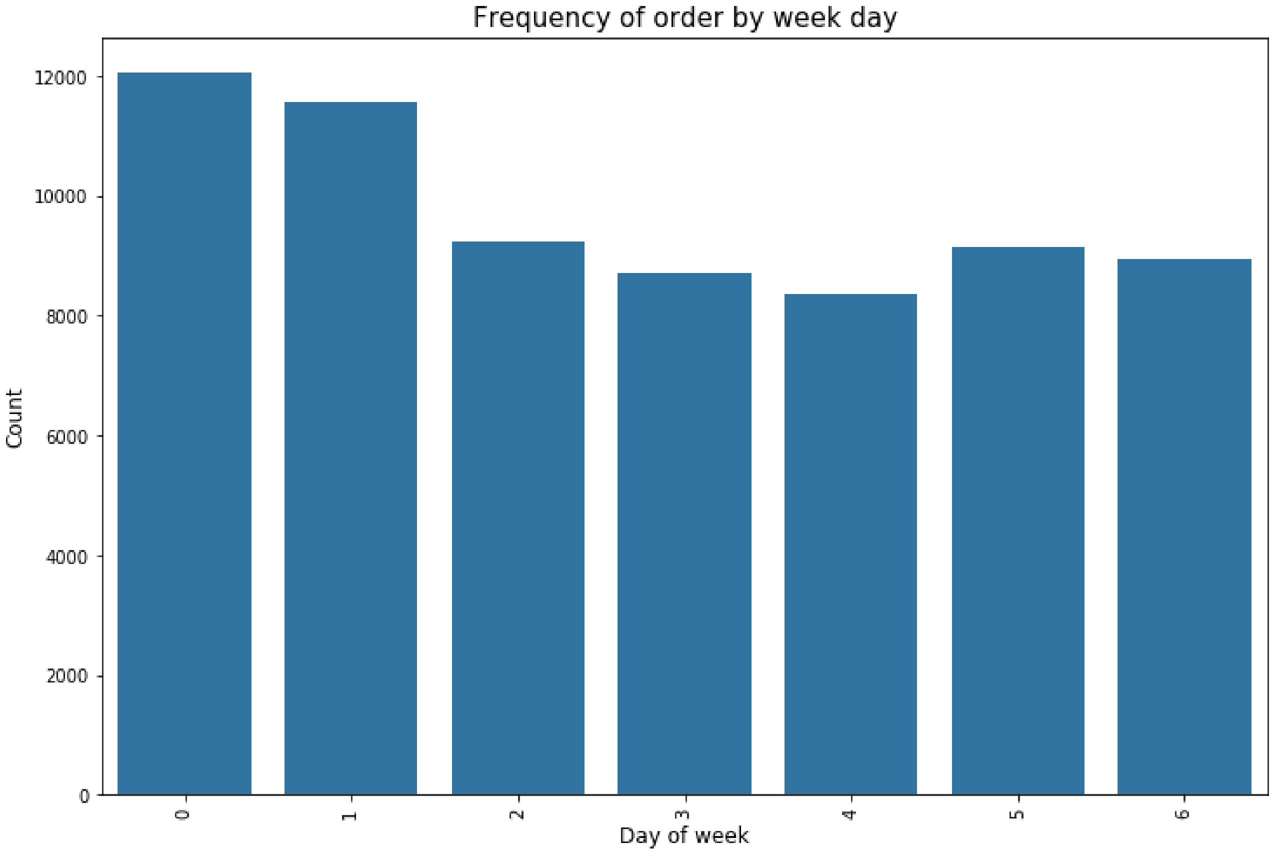


Fig-9(Frequence by Order day of Week)

```
order_products_train_df = pd.merge(order_products_train_df, orders, on='order_id', how='left')
grouped_df=order_products_train_df.groupby(["order_dow"])["reordered"].aggregate("mean").reset_index()
plt.figure(figsize=(12,8))
sns.barplot(grouped_df['order_dow'].values,grouped_df['reordered'].values,alpha=0.8,
color=color[3])
plt.ylabel('Reorder ratio', fontsize=12)
plt.xlabel('Day of week', fontsize=12)
plt.title("Reorder ratio across day of week", fontsize=15)
plt.xticks(rotation='vertical')
plt.ylim(0.5, 0.7)
plt.show()
```

out:

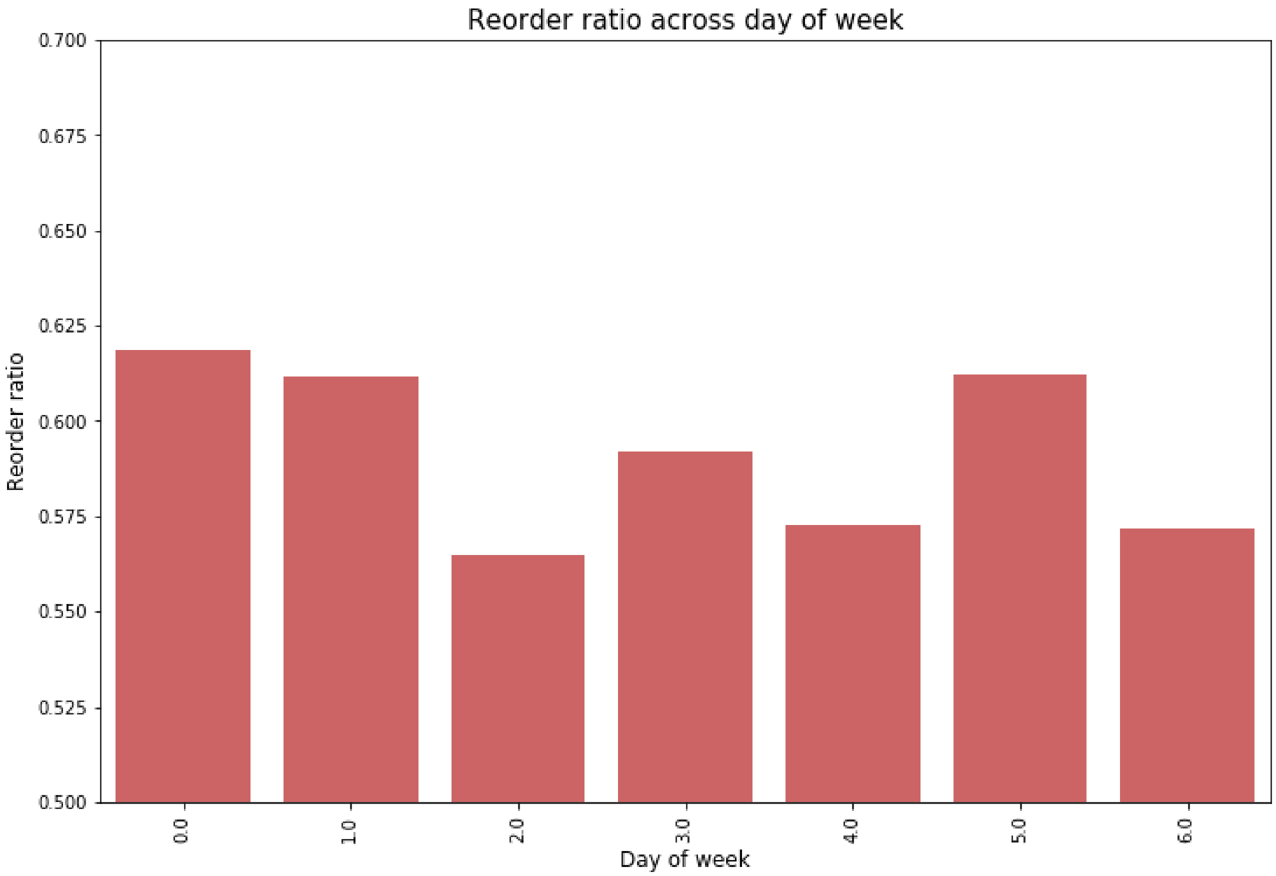


Fig-10(Reorder ratio across day of week)

```
plt.figure(figsize=(12,8))
sns.countplot(x="order_hour_of_day", data=orders, color=color[1])
plt.ylabel('Count', fontsize=12)
plt.xlabel('Hour of day', fontsize=12)
plt.xticks(rotation='vertical')
plt.title("Frequency of order by hour of day", fontsize=15)
plt.show()
```

out:

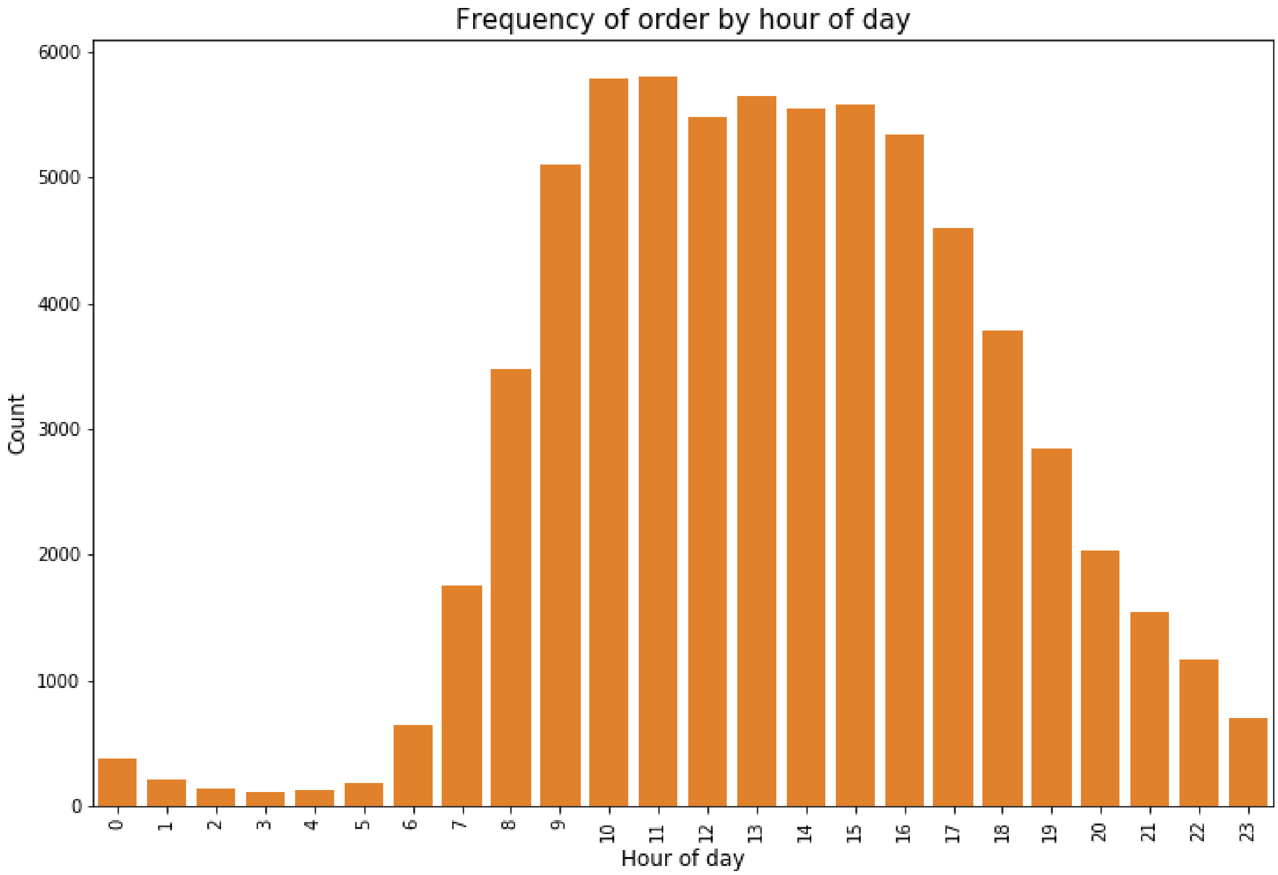


Fig-11(Frequency order by hour of day)

```
grouped_df=orders.groupby(["order_dow","order_hour_of_day"])["order_number"].aggregate("count")
.reset_index()
grouped_df = grouped_df.pivot('order_dow', 'order_hour_of_day', 'order_number')
plt.figure(figsize=(12,6))
sns.heatmap(grouped_df)
plt.title("Frequency of Day of week Vs Hour of day")
plt.show()
```

out:

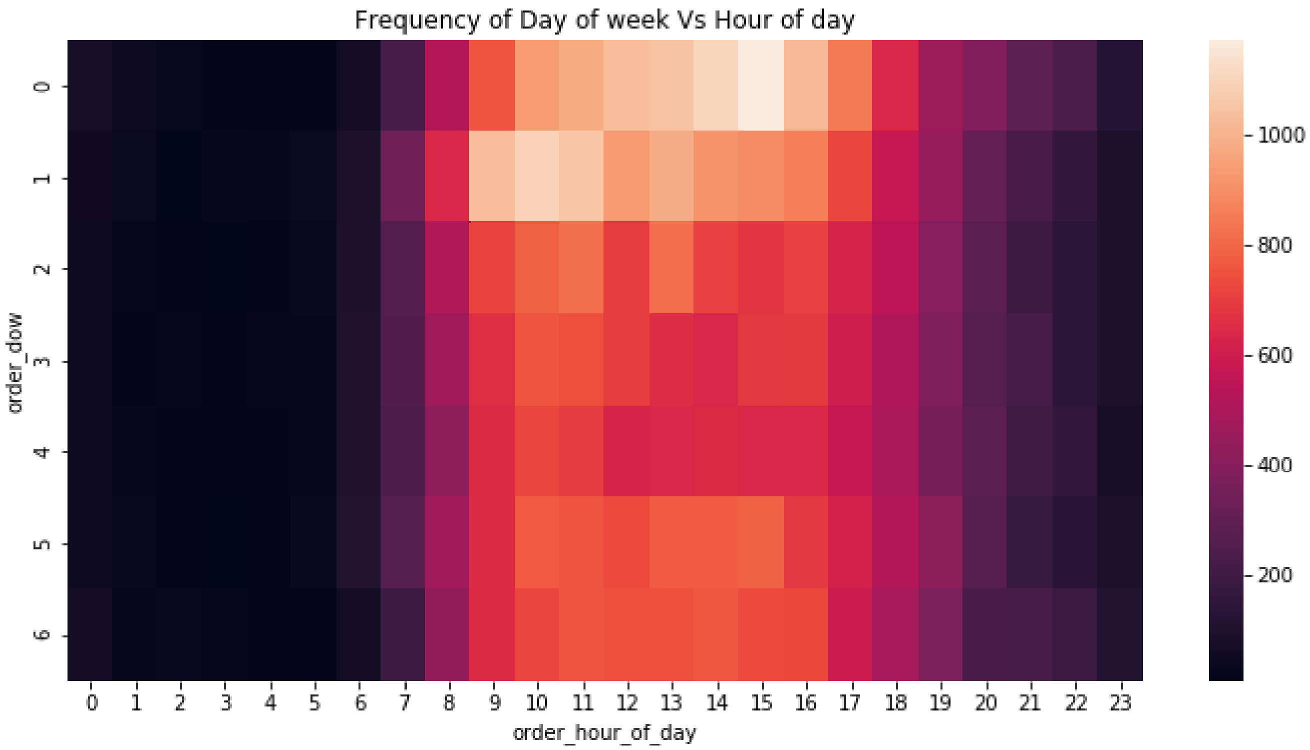


Fig-12(Frequency of Day of Week VS Hour of Day)

```
plt.figure(figsize=(12,8))
sns.countplot(x="days_since_prior_order", data=orders, color=color[3])
plt.ylabel('Count', fontsize=12)
plt.xlabel('Days since prior order', fontsize=12)
plt.xticks(rotation='vertical')
plt.title("Frequency distribution by days since prior order", fontsize=15)
plt.show()
```

out:

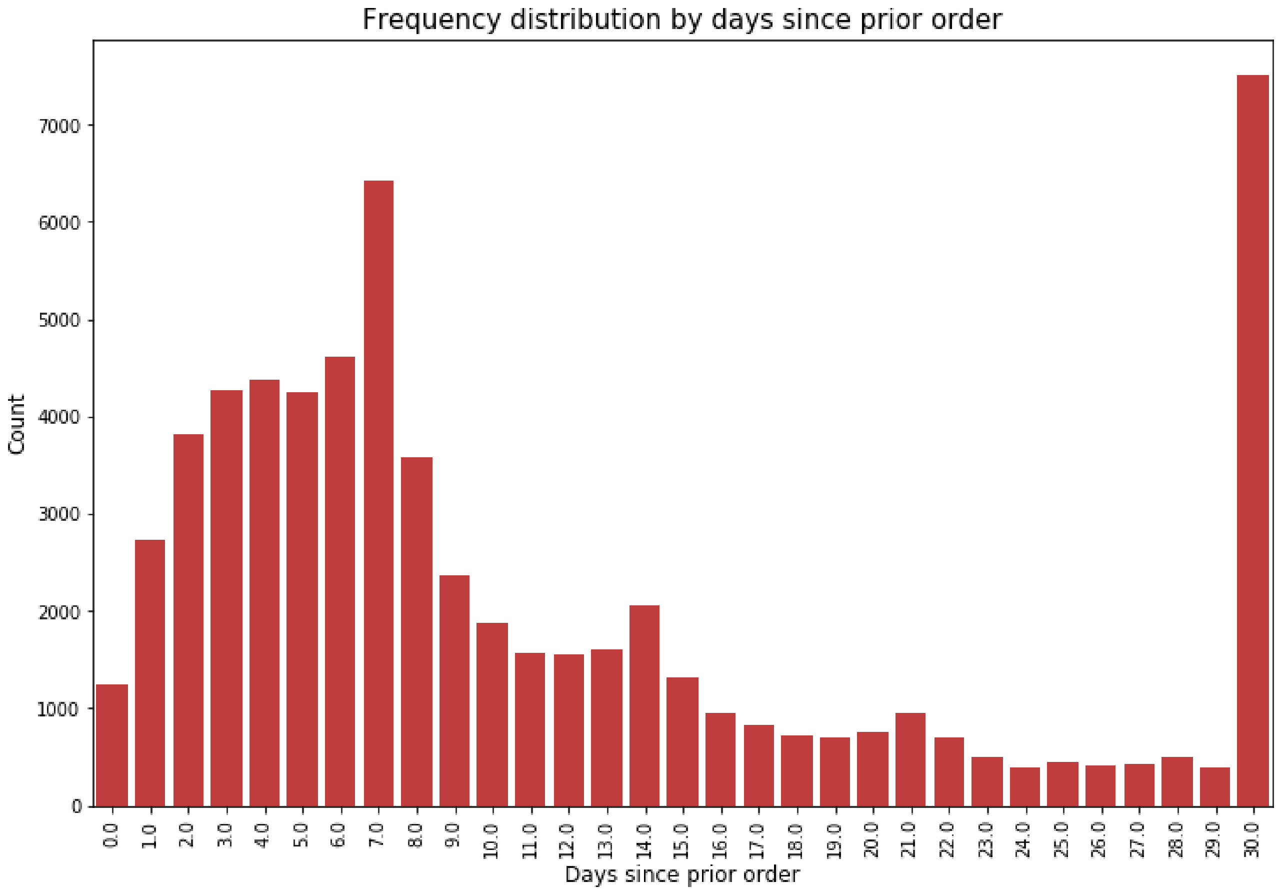


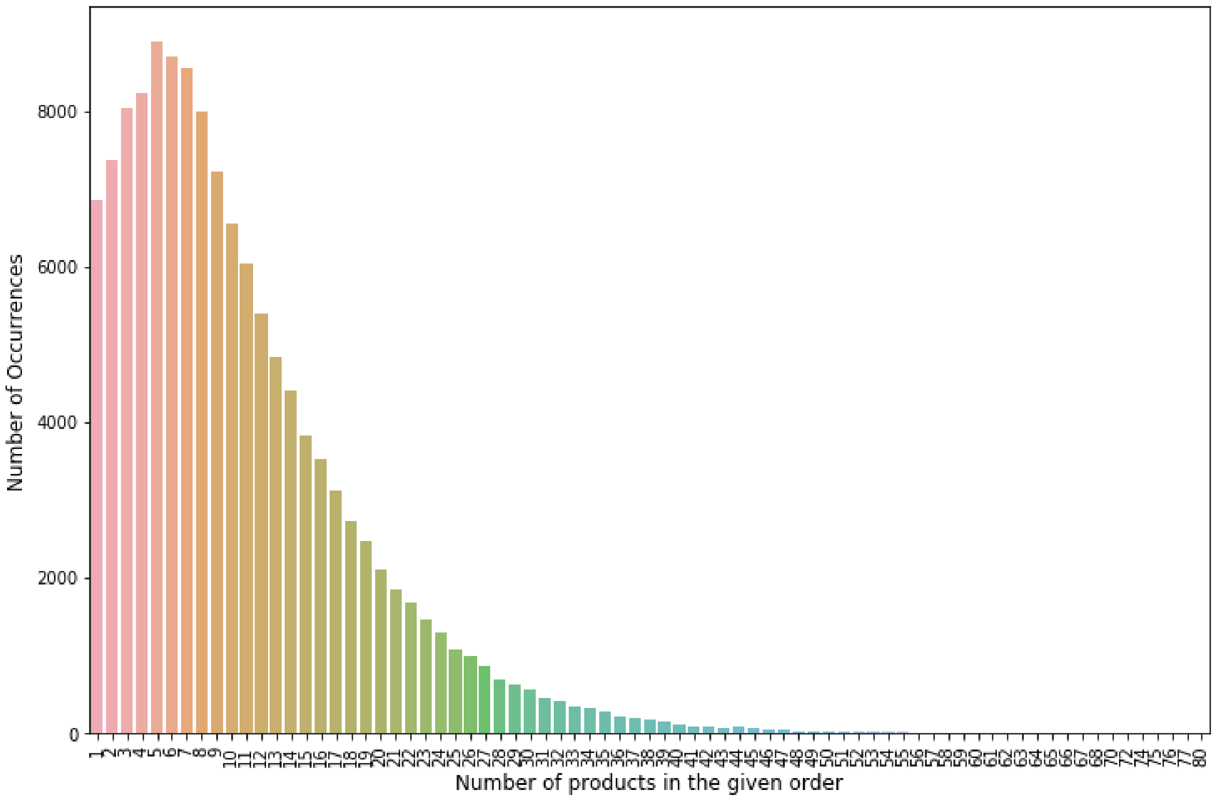
Fig-13(Frequency Distribution By Days Since Prior order)

```
grouped_df = prior_orders.groupby("order_id")["reordered"].aggregate("sum").reset_index()
grouped_df["reordered"].ix[grouped_df["reordered"]>1] = 1
grouped_df.reordered.value_counts()/grouped_df.shape[0]grouped_df=
order_products_train_df.groupby("order_id")["reordered"].aggregate("sum").reset_index()
grouped_df["reordered"].ix[grouped_df["reordered"]>1] = 1
grouped_df.reordered.value_counts() / grouped_df.shape[0]
```

About 12% of the orders in prior set has no re-ordered items while in the train set it is 6.6%.
Now let us see the number of products bought in each order.

```
grouped_df                                                    =
order_products_train_df.groupby("order_id")["add_to_cart_order"].aggregate("max").reset_index()
cnt_srs = grouped_df.add_to_cart_order.value_counts()
plt.figure(figsize=(12,8))
sns.barplot(cnt_srs.index, cnt_srs.values, alpha=0.8)
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Number of products in the given order', fontsize=12)
plt.xticks(rotation='vertical')
plt.show()
```

out:



A right tailed distribution with the maximum value at 5.!

Fig-14(No of Products in given order per Occurrences)

Calculate the initial Informed Prior

p(reordered | product_id)

```
# removing all user_ids not in the test set from both files to save memory
# the test users present ample data to make models. (and saves space)

test  = orders[orders['eval_set'] == 'test' ]
user_ids = test['user_id'].values
orders = orders[orders['user_id'].isin(user_ids)]
test.shape
```

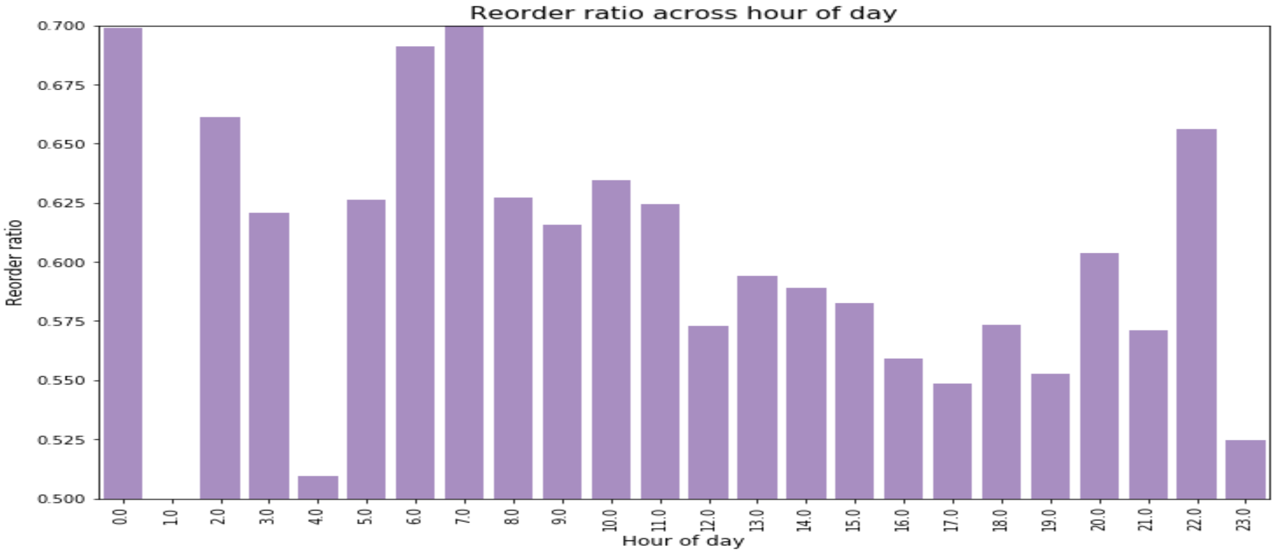
out:

(1492,7)

```
grouped_df=order_products_train_df.groupby(["order_hour_of_day"])["reordered"].aggregate("mean").reset_index()
plt.figure(figsize=(12,8))
sns.barplot(grouped_df['order_hour_of_day'].values, grouped_df['reordered'].values, alpha=0.8, color=color[4])
plt.ylabel('Reorder ratio', fontsize=12)
plt.xlabel('Hour of day', fontsize=12)
plt.title("Reorder ratio across hour of day", fontsize=15)
plt.xticks(rotation='vertical')
plt.ylim(0.5, 0.7)
plt.show()
```

out:

Fig-15(Reorder Ratio Across Hour of Day)



```
# Calculate the Prior : p(reordered|product_id)

prior=pd.DataFrame(prior_orders.groupby('product_id')['reordered']
\.agg([('number_of_orders',len),('sum_of_reorders','sum')]))
prior['prior_p'] = (prior['sum_of_reorders']+1)/(prior['number_of_orders']+2) # Informed Prior
# prior['prior_p'] = 1/2  # Flat Prior
# prior.drop(['number_of_orders','sum_of_reorders'], axis=1, inplace=True)

print('Here is The Prior: our first guess of how probable it is that a product be reordered once it has been ordered.')
prior.head()
```

out:
```
Here is The Prior: our first guess of how probable it is that a product
be reordered once it has been ordered.
```

number_of_orders	sum_of_reorders	prior_p	
product_id			
1	42	29.000	0.682
2	2	0.000	0.250
3	3	2.000	0.600
4	6	2.000	0.375
8	1	0.000	0.333

```
# merge everything into one dataframe and save any memory space

comb = pd.DataFrame()
comb = pd.merge(prior_orders, orders, on='order_id', how='right')

# slim down comb -
comb.drop(['eval_set','order_dow','order_hour_of_day'], axis=1, inplace=True)
del prior_orders
del orders
comb = pd.merge(comb, aisles, on ='product_id', how = 'left')
del aisles
prior.reset_index(inplace = True)
comb = pd.merge(comb, prior, on ='product_id', how = 'left')
del prior
print('combined data in DataFrame comb')
comb.head()
```

out:

combined data in DataFrame comb

	order_id	product_id	add_to_cart_order	Reordered	user_id	order_number	days_since_prior_order	aisle_id	number_of_orders	sum_of_reorders	prior_p
0	85	16797.000	1.000	1.000	4041	25	7.000	nan	2844.000	1999.000	0.703
1	85	19691.000	2.000	0.000	4041	25	7.000	nan	253.000	152.000	0.600
2	85	32259.000	3.000	0.000	4041	25	7.000	nan	15.000	7.000	0.471
3	85	20217.000	4.000	0.000	4041	25	7.000	nan	9.000	2.000	0.273
4	85	21349.000	5.000	0.000	4041	25	7.000	nan	13.000	9.000	0.667

Building factors

Build the factors needed for a model of probability of reordered. This model forms our hypothesis H and allows the calculation of each Bayes Factor:

$$BF=p(e|H)/(1-p(e|H))$$

where e is the test user product buying history. See DAG of model above.

We discretize reorder count into categories, 9 buckets, being sure to include 0 as bucket.
These bins maximize mutual information with ['reordered'].

```
recount = pd.DataFrame()
recount['reorder_c'] = comb.groupby(comb.order_id)['reordered'].sum().fillna(0)
bins = [-0.1, 0, 2,4,6,8,11,14,19,71]
cat =  ['None','<=2','<=4','<=6','<=8','<=11','<=14','<=19','>19']
recount['reorder_b'] = pd.cut(recount['reorder_c'], bins, labels = cat)
recount.reset_index(inplace = True)
comb = pd.merge(comb, recount, how = 'left', on = 'order_id')
del recount
comb.head(50)
```

Discretize 'add_to_cart_order' (atco) into categories, 8 buckets. These bins maximize mutual information with ['recount'].

```
bins = [0,2,3,5,7,9,12,17,80]
cat = ['<=2','<=3','<=5','<=7','<=9','<=12','<=17','>17']
comb['atco1'] = pd.cut(comb['add_to_cart_order'], bins, labels = cat)
del comb['add_to_cart_order']
print('comb')
comb.head(50)
```

These are the children Nodes of reordered:atco, aisle, recount.
Build occurrence tables first, then calculate probabilities. Then merge to add atco into comb

```
atco_fac = pd.DataFrame()
atco_fac = comb.groupby(['reordered', 'atco1'])['atco1'].agg(np.count_nonzero).unstack('atco1')
tot = pd.DataFrame()
tot = np.sum(atco_fac,axis=1)
atco_fac = atco_fac.iloc[:,:].div(tot, axis=0)
atco_fac = atco_fac.stack('atco1')
atco_fac = pd.DataFrame(atco_fac)
atco_fac.reset_index(inplace = True)
atco_fac.rename(columns = {0:'atco_fac_p'}, inplace = True)
comb = pd.merge(comb, atco_fac, how='left', on=('reordered', 'atco1'))
comb.head(50)
```

```
aisle_fac = pd.DataFrame()
aisle_fac=comb.groupby(['reordered','atco1','aisle_id'])['aisle_id']\.agg(np.count_nonzero).unstack('aisle_id')
tot = np.sum(aisle_fac,axis=1)
aisle_fac = aisle_fac.iloc[:,:].div(tot, axis=0)
aisle_fac = aisle_fac.stack('aisle_id')
aisle_fac = pd.DataFrame(aisle_fac)
aisle_fac.reset_index(inplace = True)
aisle_fac.rename(columns = {0:'aisle_fac_p'}, inplace = True)
comb = pd.merge(comb, aisle_fac, how = 'left', on = ('aisle_id','reordered','atco1'))
comb.head(50)
```

```
recount_fac = pd.DataFrame()
recount_fac = comb.groupby(['reordered', 'atco1', 'reorder_b'])['reorder_b']\
                .agg(np.count_nonzero).unstack('reorder_b')

tot = pd.DataFrame()
tot = np.sum(recount_fac,axis=1)

recount_fac = recount_fac.iloc[:,:].div(tot, axis=0)
recount_fac.stack('reorder_b')
recount_fac = pd.DataFrame(recount_fac.unstack('reordered').unstack('atco1')).reset_index()
recount_fac.rename(columns = {0:'recount_fac_p'}, inplace = True)
comb = pd.merge(comb, recount_fac, how = 'left', on = ('reorder_b', 'reordered', 'atco1'))
recount_fac.head(50)
```

We use the factors in comb + the prior_p to update a posterior for each product purchased.(table not shown here)

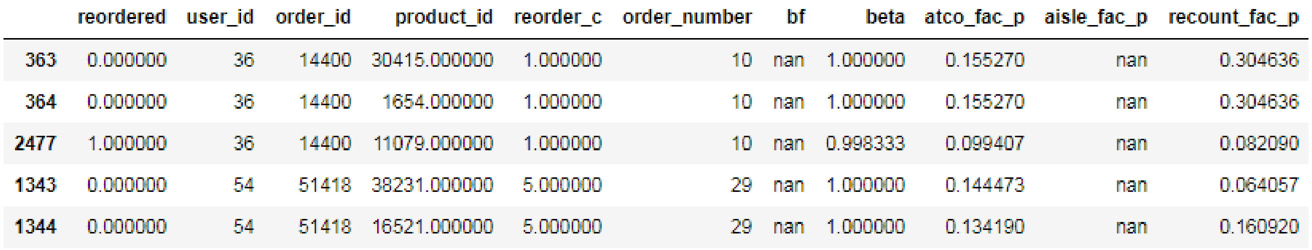
```
p = pd.DataFrame()
p = (comb.loc[:,'atco_fac_p'] * comb.loc[:,'aisle_fac_p'] * comb.loc[:,'recount_fac_p'])
p.reset_index()
comb['p'] = p
comb.head(30)
```

Filing Bayess Factors for intermediate orders

We now split into three dataframes. Two of them are reordered == 1 and == 0 This is done because Bayes Factor (BF) is calculated differently for each case. We then append this to a table called 'comb_last' which will be using for modelling Bayes Factor depending on whether the product was ordered or not.

```
# Calculate bf0 for products when first purchased aka reordered=0

comb0 = pd.DataFrame()
comb0 = comb[comb['reordered']==0]
comb0.loc[:,'first_order'] = comb0['order_number']
# now every product that was ordered has a posterior in usr.
comb0.loc[:,'beta'] = 1
comb0.loc[:,'bf'] = (comb0.loc[:,'prior_p'] * comb0.loc[:,'p']/(1 - comb0.loc[:,'p'])) # bf1

# Small 'slight of hand' here. comb0.bf is really the first posterior and second prior.

# Calculate beta and BF1 for the reordered products

comb1 = pd.DataFrame()
comb1 = comb[comb['reordered']==1]
comb1.loc[:,'beta'] = (1 - .05*comb1.loc[:,'days_since_prior_order']/30)
comb1.loc[:,'bf'] = (1 - comb1.loc[:,'p'])/comb1.loc[:,'p'] # bf0
comb_last = pd.DataFrame()
comb_last = pd.concat([comb0, comb1], axis=0).reset_index(drop=True)
comb_last = comb_last[['reordered', 'user_id', 'order_id', 'product_id','reorder_c','order_number',
                    'bf','beta','atco_fac_p', 'aisle_fac_p', 'recount_fac_p']]
comb_last = comb_last.sort_values((['user_id', 'order_number', 'bf']))
pd.set_option('display.float_format', lambda x: '%.6f' % x)
comb_last.head()
```

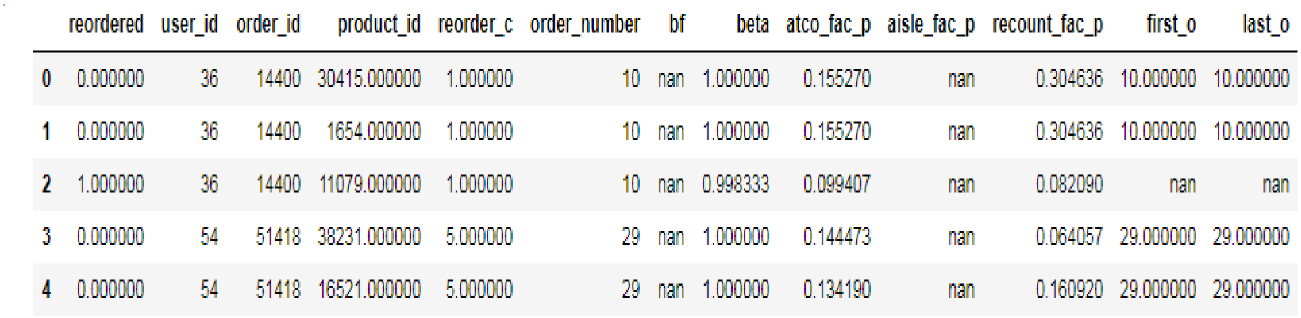
out:



```
      reordered  user_id  order_id    product_id  reorder_c  order_number   bf      beta  atco_fac_p  aisle_fac_p  recount_fac_p
363    0.000000       36     14400  30415.000000   1.000000            10  nan  1.000000    0.155270          nan       0.304636
364    0.000000       36     14400   1654.000000   1.000000            10  nan  1.000000    0.155270          nan       0.304636
2477   1.000000       36     14400  11079.000000   1.000000            10  nan  0.998333    0.099407          nan       0.082090
1343   0.000000       54     51418  38231.000000   5.000000            29  nan  1.000000    0.144473          nan       0.064057
1344   0.000000       54     51418  16521.000000   5.000000            29  nan  1.000000    0.134190          nan       0.160920
```

```
first_order = pd.DataFrame()
first_order = comb_last[comb_last.reordered == 0]
first_order.rename(columns = {'order_number':'first_o'}, inplace = True)
first_order.loc[:,'last_o'] = comb_last.groupby(['user_id'])['order_number'].transform(max)
first_order = first_order[['user_id','product_id','first_o','last_o']]
comb_last = pd.merge(comb_last, first_order, on = ('user_id', 'product_id'), how = 'left')
comb_last.head()
#com = pd.DataFrame()
#com = comb_last[(comb_last.user_id == 3) & (comb_last.first_o < comb_last.order_number)]
#com.groupby([('order_id', 'product_id', 'order_number')])['bf'].agg(np.sum).head(50)
```

Out:



	reordered	user_id	order_id	product_id	reorder_c	order_number	bf	beta	atco_fac_p	aisle_fac_p	recount_fac_p	first_o	last_o
0	0.000000	36	14400	30415.000000	1.000000	10	nan	1.000000	0.155270	nan	0.304636	10.000000	10.000000
1	0.000000	36	14400	1654.000000	1.000000	10	nan	1.000000	0.155270	nan	0.304636	10.000000	10.000000
2	1.000000	36	14400	11079.000000	1.000000	10	nan	0.998333	0.099407	nan	0.082090	nan	nan
3	0.000000	54	51418	38231.000000	5.000000	29	nan	1.000000	0.144473	nan	0.064057	29.000000	29.000000
4	0.000000	54	51418	16521.000000	5.000000	29	nan	1.000000	0.134190	nan	0.160920	29.000000	29.000000

```
temp = pd.pivot_table(comb_last[(comb_last.user_id == 3) & (comb_last.first_o ==
comb_last.order_number)],values = 'bf', index = ['user_id', 'product_id'],columns = 'order_number',
dropna=False)
temp.head(10)

temp = temp.fillna(method='pad', axis=1).fillna(1)
temp.head(10)


pd.pivot_table(comb_last[comb_last.first_o <= comb_last.order_number],
                    values = 'bf', index = ['user_id', 'product_id'],
                    columns = 'order_number').head(10)


temp.update(pd.pivot_table(comb_last[comb_last.first_o <= comb_last.order_number],
                    values = 'bf', index = ['user_id', 'product_id'],
                    columns = 'order_number'))
temp.head()
```

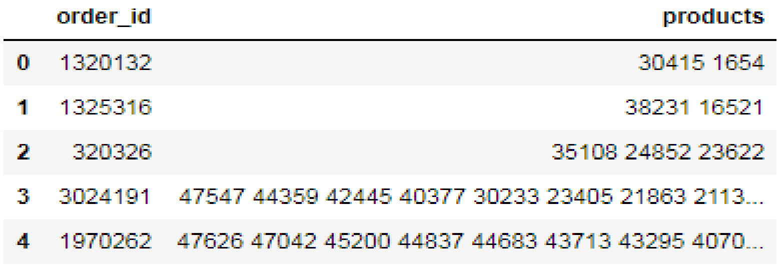
```
import logging
logging.basicConfig(filename='bayes.log',level=logging.DEBUG)
logging.debug("Started Posterior calculations")
print("Started Posterior calculations")

pred = pd.DataFrame(columns=['user_id', 'product_id'])
# comb_last_temp = pd.DataFrame()
# com = pd.DataFrame()

for uid in comb_last.user_id.unique():
    if uid % 1000 == 0:
        print("Posterior calculated until user %d" % uid)
        logging.debug("Posterior calculated until user %d" % uid)
del comb_last_temp
    comb_last_temp = pd.DataFrame()
    comb_last_temp = comb_last[comb_last['user_id'] == uid].reset_index()
#     del com    com = pd.DataFrame()
    com = pd.pivot_table(comb_last_temp[comb_last_temp.first_o ==
comb_last_temp.order_number], values = 'bf', index = ['user_id', 'product_id'],
                         columns = 'order_number', dropna=False)
    com = com.fillna(method='pad', axis=1).fillna(1)
   com.update(pd.pivot_table(comb_last_temp[comb_last_temp.first_o <=
comb_last_temp.order_number],  values = 'bf', index = ['user_id', 'product_id'],
  columns = 'order_number'))
   com.reset_index(inplace=True)
   com['posterior'] = com.product(axis=1)
pred = pred.append(com.sort_values(by=['posterior'], ascending=False).head(10)    \
                       .groupby('user_id')['product_id'].apply(list).reset_index())


print("Posterior calculated for all users")
logging.debug("Posterior calculated for all users")
pred = pred.rename(columns={'product_id': 'products'})
pred.head(10)
```

```
Started Posterior calculations
Posterior calculated for all users
```

	user_id	Products
0	36	[30415.0, 1654.0]
0	54	[38231.0, 16521.0]
0	57	[35108.0, 24852.0, 23622.0]
0	68	[47547.0, 44359.0, 42445.0, 40377.0, 30233.0, ...
0	75	[47626.0, 47042.0, 45200.0, 44837.0, 44683.0, ...
0	126	[41834.0, 14647.0, 3880.0]
0	129	[36929.0]
0	136	[47591.0, 45104.0, 45007.0, 41165.0, 40516.0, ...
0	153	[20120.0]
0	154	[44805.0, 44667.0, 37908.0, 12803.0]

```
pred = pred.merge(test, on='user_id', how='left')[['order_id', 'products']]
pred['products'] = pred['products'].apply(lambda x: [int(i) for i in x])\.astype(str).apply(lambda x: x.strip('[]').replace(',', ''))
pred.head(5)
```

Predicted Order:Final Result



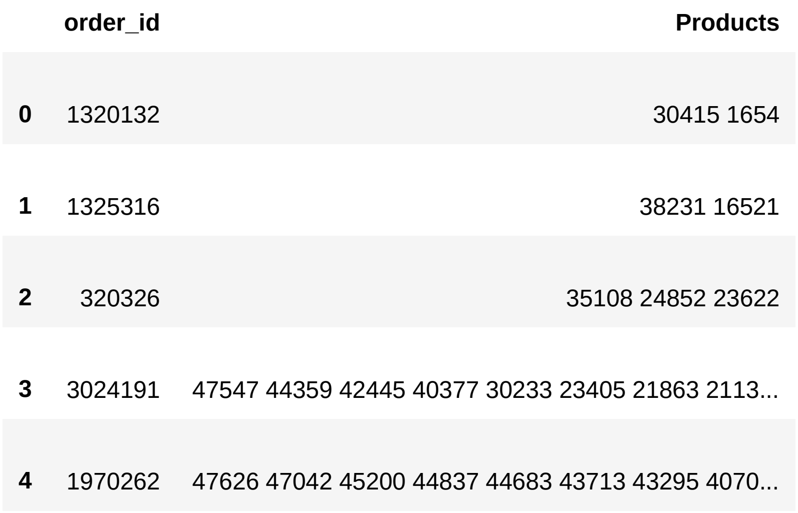
	order_id	products
0	1320132	30415 1654
1	1325316	38231 16521
2	320326	35108 24852 23622
3	3024191	47547 44359 42445 40377 30233 23405 21863 2113...
4	1970262	47626 47042 45200 44837 44683 43713 43295 4070...

5. RESULT

RESULT

A Bayesian Belief network to analyze the behavior of the customers,and predict their next purchase, along with the order in which they are bought in that transaction.So now we predict product along with order id's .

This is the final result.



	order_id	Products
0	1320132	30415 1654
1	1325316	38231 16521
2	320326	35108 24852 23622
3	3024191	47547 44359 42445 40377 30233 23405 21863 2113...
4	1970262	47626 47042 45200 44837 44683 43713 43295 4070...

6. CONCLUSION AND FUTURE WORK

6.1 Conclusion

The baysian rule effectively generates highly informative results for frequent itemsets and association rules for the data of the supermarket. The frequent data items are generated from the given input data and based on the frequent item stets strong association rules were generated.

6.2 Recommendations

The input data given to the application is used as the integer value mapped from the transaction database. The mapping is done manually. If database converter is made then the system will work effectively for any format of data. The application can be efficiently used by using more efficient algorithm rather that Apriori Algorithm in future.

6.3 Future Enhancements

In future We use better algorithms to modify model to prediction of products and predict more precise.

7. REFERENCES

References

[1]Afrati FN, Gionis A, Mannila H (2004) Approximating a collection of frequent sets. In: Proceedings of the 2004 ACM SIGKDD international conference knowledge discovery in databases (KDD'04), Seattle, WA, pp 12–19

[2] "A Fast Distributed Algorithm for Mining Association Rules",Proc. Parallel and Distributed Information
Systems; D.W. Cheung, et al., IEEE CS Press, 1996,pp. 31-42

[3] "Introduction: Recent Developments in Parallel and Distributed Data Mining",J. Distributed and Parallel Databases; M.J. Zaki and Y. Pin, vol. 11, no. 2, 2002,pp. 123-127

[4] "Efficient Mining of Association Rules in Distributed Databases",IEEE Trans. Knowledge and Data Eng.; D.W. Cheung , et al., vol. 8, no. 6, 1996,pp. 911-922

[5] "Communication-Efficient Distributed Mining of Association Rules"; A. Schuster and R. Wolff, Proc. ACM SIGMOD Int'l Conf. Management of Data, ACM Press, 2001,pp. 473-484

[6] "Mining Association Rules Between Sets of Items in Large Databases"; R. Agrawal, T. Imielinski, and A. Swami, Proc. ACMSIGMOD Int'l Conf. Management of Data, , May 1993

[7] "An Optimized Distributed Association Rule Mining Algorithm"; M.Z Ashrafi, Monash University ODAM, IEEE DISTRIBUTED SYSTEMS ONLINE 1541-4922 2004

[8] "The Data Warehouse Toolkit, The Complete Guide to Dimensional Modeling",2nd edn. John Wiley & Sons; Kimball, R., Ross, M., New York (2002)

[9] Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. In: Proceedings of the 1994 international conference on very large data bases (VLDB'94), Santiago, Chile, pp 487–499