Proceedings of the 3rd International Conference on Signal Processing and Machine Learning
DOI: 10.54254/2755-2721/6/20230812

Probalistic model of spam filter system

Yanzhi Xiong[1,3] , Tianlan Wei[2,4]

[1]Department of mathematics, Xi'an Jiaotong-Liverpool University, Suzhou, China
[2]Department of mathematics, Xi'an Jiaotong University, XiAn, China

[3]Yanzhi.Xiong20@student.xjtlu.edu.cn
[4]tinawei@stu.xjtu.edu.cn

Abstract. Numerous tasks have been carried out using probabilistic reasoning, including picture recognition, computer diagnosis, stock price prediction, movie recommendation, and cyber intrusion detection. But until recently, the breadth of probabilistic programming was constrained (partly because of the low processing power), and the majority of inference methods had to be created manually for every job. Nevertheless, in 2015, 3D representations of human faces were created from 2D photographs of such faces using a 50-line probabilistic computer vision software. The inference approach of the software, which was developed in Julia using the Picture package, was based on inverted graphics. "What used to require thousands of lines of code" might now be accomplished in just 50. Probabilistic computing is a method to create systems that help us make decisions in the face of uncertainty. In this paper we propose a spam-filtering system. The novelty of our spam-filtering system is that we utilize probabilistic programming to improve spam-filtering reasoning systems. We implement several factors from email to establish a model to get the output. Our preliminary results show that we successfully accomplish a spam-filtering system. More problem of classification of spam filter can be described in a probabilistic modelling language in our future work.

Keywords: spam filter, probablistic programming, probablistic model

1. Introduction

In the real world, few of our concerns have a clear, black or white answer. And many daily decisions involve judgment, but some related factors are not directly observed. So people use Probabilistic programming, which is a way to create a system which can help people make decisions in the face of uncertainty. And using a model to make decisions in the face of uncertainty is a technique known as probabilistic reasoning. In probabilistic reasoning, people construct a model that quantifies and probabilistically expresses all the pertinent general knowledge of our subject. Probabilistic inference, often known as inference, is the act of applying the model to provide answers to questions based on the available data[1-18].

Probabilistic reasoning systems are flexible. There are three types of reasoning in probabilistic reasoning systems in practice: (1) Foresee future events. (2) Infer the cause of events. (3) Learn from the past to predict the future more accurately. These three types of patterns offer options for reasoning about specific situations, providing evidence and using historical information to broaden knowledge base.

DOI: 10.54254/2755-2721/6/20230812

The first reasoning pattern predicts future events based on understanding of the current situation, while the second reasoning pattern infers the past situation based on the current results. When building a probabilistic model, the model itself usually follows a natural chronological order. However, there is no requirement that the direction of reasoning match the direction of the model, which is a major characteristic of probabilistic reasoning.

In addition to the third pattern of reasoning, there is also a method for improving the model itself—especially if people have a wealth of prior experiences to draw from. This is achieved by a learning algorithm. The learning algorithm's objective is to create a new model. To create a new model, it starts with the previous model and updates it in light of experience. Then revised model can be used to respond questions. It is speculated that the new model will produce a better answer than the original model.

To convey its probabilistic models, every probabilistic reasoning system uses a representation language, which include hidden Markov models and Bayesian networks (also known as belief networks). This form of language regulates which models the system can handle and how they appear.

A probabilistic reasoning system with programming as its representation language is known as a probabilistic programming system. All the characteristics of a standard programming language are contained in the programming language, including variables, a wide range of data types, control flow, functions, and so forth.

There are some relationships between probabilistic reasoning systems and probabilistic programming systems. The most significant modification is that models are now written as programs in programming languages rather than as mathematical structures like a Bayesian network. This modification affects how evidence, questions, and responses relate to program variables. Program variable values may be specifically stated in evidence, while program variable values are requested in questions, and answers are probabilities of various query variable values. A probabilistic programming system also frequently includes a collection of inference methods. The language's programs can use these algorithms.

Execution is a fundamental concept in programming languages. To produce output, you run a program. A probabilistic program is similar, but rather than having only one execution path, it may have several, each of which produces a distinct output. Random selections made throughout the program specify which execution path will be taken.

After knowing what probabilistic programming is, we will explain why we choose it to solve the problem.

One of the core components of machine learning is probabilistic reasoning. And applications as varied as stock price forecasting, movie recommendation, computer diagnosis, and cyber intrusion detection have all used probabilistic reasoning.

There are two compelling points: (1) To anticipate the future, infer the past, and improve future predictions, one can utilize probabilistic reasoning. (2) Probabilistic programming is the modeling of probabilistic reasoning using a Turing-complete computer language. Probabilistic programming is inspired by the fact that it combines two ideas that are both potent in their own right. As a result, using computers to assist in making decisions under uncertainty is made simpler and more adaptable.

In recent years, more sophisticated probabilistic programming has been created, some of which also include programming-language elements like recursion and arrays. Nowadays, under the help of many interacting entities and events, long-running processes may be modeled. Probabilistic programming is also a method to predict future. The crucial point insight in the development of probabilistic programming is that many of the inference algorithms that applicable to more basic modeling frameworks can also be applied to simulations. As a result, it is feasible to write a simulation and use the results to build a probabilistic model.

Based on the above knowledge about probabilistic programming, we will involve a complete design of spam filter and analyze the common architecture of probabilistic programming reflected in the process.

DOI: 10.54254/2755-2721/6/20230812

2. Problem & solution description

2.1. Learn about spam filters through the big picture

How the spam filter integrates into a large email app?

Email comes into the server. Before delivering an email to the user, the server passed it to the spam-filter reasoning component, which used probabilistic reasoning to assess the likelihood that the email was spam. The inference process is used to the email model and incoming emails as part of the reasoning component, which employs probabilistic reasoning. The reasoning component outputs the probability that the email is spam. According to policy, filter evaluates spam probability. The email is passed to the user when the probability is not too high.

When faced with evidence, a model-based inference method is used by a probabilistic reasoning system to produce a conclusion. The system, for instance, uses a model of legitimate and spam emails that is part of the email model. Is the email spam, that is the question. "Spam likelihood," is the response. The email model must be sourced in some way for the reasoning component to work. A learning component may be able to learn the model from training data. The process of learning involves gathering training data and creating a probabilistic reasoning model. We obviously gave a basic introduction to learning as a capability of a probabilistic reasoning system.

The model used for probabilistic reasoning is created through learning in real-world scenarios. The manual construction of the model utilizing your knowledge of the system we are modeling is an alternate strategy.

Think about how the learning component produces the email model as an example. Given that this offline component doesn't apply to every email pattern, it might take some time to tune and acquire the best model. Before the reasoning component may function, the learning component must complete its task and generate the email model. If the email model doesn't need to be updated in the future, the learning component is no longer required.

In this instance, the preceding email model includes the necessary structural presumptions for creating a spam filter.

This model specifically specifies that the presence or absence of specific terms determines whether an email is spam, however it does not specify which words signify spam and with what probability. These are discovered using the data. The previous model similarly claims that the quantity of odd terms in an email is related to be spam, but it doesn't specify how this is so. Again, the data is used to infer this association.

In a nutshell, what the learning component learns composes of the following information.

which terms are helpful in identifying spam from legitimate emails. The feature words are what we refer to.

A set of parameters is used to represent various probabilities, such as the likelihood that an email is spam before you even open it, the likelihood that a specific word will appear in the email given that it is either normal or spam, and the likelihood that an email contains a large number of unusual words given that it is either normal or spam.

When the data set is first read, the first item—the set of feature words—is instantly read. Learning the parameters is the core of the learning component. The reasoning component then makes use of these identical parameters.

2.2. The architecture of a spam filter

The spam filter has two parts. An online component classifies emails as spam or normal using probabilistic reasoning and filters or forwards them to the user as necessary. A training batch of emails is used by an offline component to learn the email model.

2.2.1. Reasoning component architecture. The first is a reasoning component. When determining the architecture of components, the first thing to be decided is the input, output and their relationships. We summarize the spam-filter reasoning component as:

DOI: 10.54254/2755-2721/6/20230812

A text file that serves as an email INPUT. A double reflecting the likelihood that the email is span is the OUTPUT. Then, we gradually build up the spam filter's architecture.

Step 1: Declare the high-level architecture to be a probabilistic reasoning system. The span filter reasoning component uses the email's text as evidence to evaluate if the email is normal or span. In the reasoning component, we also employ an inference technique and an email model.

Step 2: Improve the architecture of the program for spam-filtering. A feature extractor in machine learning is a component that converts raw data into evidence for model components. Features are the components of the data that serve as evidence.

Step 3: Clearly define the architecture.This model consists of five distinct components:

Templates defining which elements are in the model; The dependencies between the elements; The function forms of these dependencies; The numerical parameters of the model; The knowledge.

2.2.2. Learning component architecture. The second component is a learning component. To summarize the spam-filtering learning component.

A group of text files that represent emails and a file that identifies a portion of the emails as regular or spam make up INPUT.

Numerical parameters describing the generating process and the supplemental information for the model are the OUTPUT.

Although the inference component and the spam filter learning component share many components, there are some significant distinctions.

- The learning component is based on an old email template.
- The knowledge extractor extracts knowledge directly from training emails.
- Like with the reasoning component, the feature extractor takes features from every email and transforms them into evidence.
- It uses learning algorithm instead of inference algorithm.

2.3. Design an email model

Next content is how to design the probabilistic model. Since the process and knowledge of the two components are the same, the difference is only in the parameters, so all designs are applicable to both. This consists of four steps.

2.3.1. Choosing the elements. When designing a spam filter, we need to consider what factors can help us judge whether email is normal or spam. The items include:

- Certain phrases in the email's header or body
- Misspelling and other unusual words.
- The proximity of some words to other terms.
- Characteristics of natural language.
- Features of the header.
- To keep the application simple, we simplify design decisions:
- Use only the first two of these items.
- Lump header and body together to reason about.
- Use a Boolean element to create an element about a word in an email.
- Use different approach to find whether a word is well-defined or unusual.
- Then, the following step is to select the model's elements.
- Representing presence or absence of words.
- Representing the number of unusual words.

2.3.2. Defining the dependencies. Let's start by defining the dependency model for all the word elements and the isSpam element. The email's class and its words are dependent on one another in the model. Although the words are not independent in reality, we can make the assumption that they are

DOI: 10.54254/2755-2721/6/20230812

independent. We can draw a diagram called a Bayesian network that shows the dependencies in the model.

The class of the object determines the object's properties, therefore the properties rely on the class in the dependency model when constructing the dependencies of the probabilistic model, according to a common rule. The dependencies in the model range from email categories to words. Although these words are not independent in reality, we can assume that they are independent and draw a Bayesian network to show the dependencies in the model.

There are two elements for unusual words: hasManyUnusualWords and numUnusualWords. hasManyUnusualWords indicates whether the words in this email tend to be unusual, whereas numUnusualWords is the number. As an email with unusual words tends to have more unusual words than an email that doesn't, numUnusualWords relies on hasManyUnusualwords. And, hasManyUnusualWords, which is a property of the email, depends on isSpam, which represents the class of the email. Finally, continuing with the Naive Bayes' assumption, we assume that under the condition that the class is known, the existence of unusual words has nothing to do with the existence of any particular word

2.3.3. Explanation of the functional forms. Next, the functional forms are defined. They serve as the model's element class constructors.

2.3.4. Using numerical parameters. The parameters mentioned above will be provided by the learning results in the reasoning components. The code of LearnedParameters class and PriorParameters class are shown in the book.

2.3.5. Working with auxiliary knowledge. Which terms qualify as feature words and whether a particular word is unusual to construct a model for particular email must be known. The Dictionary data structure will offer all of these services.

3. Conclusion

The following is a summary of probabilistic programming. Making decisions involves reasoning and information. In probabilistic reasoning, an inference method encodes the logic while a probabilistic model communicates the knowledge. To predict future occurrences, understand the causes of the past, and learn from the past to improve forecasts, probabilistic reasoning can be utilized. (Probabilistic programming is probabilistic reasoning in which a programming language is used to express the probabilistic model. A probabilistic programming system offers inference methods to use the models and employs a Turing-complete programming language to represent the models. The steps for developing a probabilistic programming application are summarized here. The learning component makes use of prior parameters to determine values, which are subsequently employed by the reasoning component. In most cases, probability programs are used to learn the model from training data and then apply it to the application of specific instances as needed. Numerous applications of probabilistic programming share a common architecture, which includes learning and reasoning components. A model that includes the specification of the generative process, the process parameters, and auxiliary information is used by both the reasoning and learning components. Prior parameters are used by the learning component to learn values for those parameters, which are then used by the reasoning component. The problem of classification of spam filter can be described in a probalistic modelling language. So in the future work, we can improve the spam filters by proving and using the statistical knowledge.

Reference

[1] W. J. Anderson. Continuous-Time Markov Chains. Springer-Verlag, New York, 1991.
[2] R. J. Adler. The Geometry of Random Fields., Wiley, New York, 1981.

DOI: 10.54254/2755-2721/6/20230812

[3] P. Baldi, L. Mazliak and P. Priouret. Martingales and Markov Chains: Solved Exercises and Elements of Theory. Chapman-Hall/CRC, Boca Raton, 2002.
[4] I. V. Basawa and B. L. S. Prakasa Rao. Statistical Inference for Stochastic Processes. Academic Press, London.
[5] P. Billingsley. Convergence of Probability Measures, 2nd Ed., Wiley, New York, 2000.
[6] K. L. Chung. Markov Chains with Stationary Transition Probabilities, 2nd Ed. Springer-Verlag,
[7] New York, 1967.
[8] K. L. Chung and R. J. Williams. Introduction to Stochastic Integration, 2nd Ed. Birkhauser, Boston, 1990.
[9] P. Embrechts and M. Maejima. Selfsimilar Processes. Princeton University Press, Princeton, 2002.
[10] T. Hida and M. Hitsuda. Gaussian Processes. American Mathematical Society, Providence, R.I. , 1991.
[11] O. Kallenberg. Random Measures. Academic Press, London, 1976.
[12] I. Karatzas and S. E. Shreve. Brownian Motion and Stochastic Calculus. Springer-Verlag, New York, 1991.
[13] E. Lukacs. Characteristic Functions, 2nd Ed. Griffin, 1970.
[14] S. P. Meyn and R. L. Tweedie. Markov Chains and Stochastic Stability. Springer-Verlag, New York, 1993.
[15] V. V. Petrov. Limit Theorems of Probability Theory. Oxford University Press, Oxford, 1995.
[16] G. Samorodnitsky and M. S. Taqqu. Stable Non-Gaussian Random Processes. Chapman-Hall/CRC, Boca Raton, 1994.
[17] G. R. Shorack. Probability for Statisticians. Springer-Verlag, New York, 2000.
[18]]R. Durrett. Probability: theory and examples (Vol. 49). Cambridge university press, 2019.
[19] A. Pfeffer. Practical probabilistic programming. Simon and Schuster, 2016.