Project 1　American Express - Default Prediction

1. Problem statement

Credit card consumption is one of the most mainstream consumption modes in the United States, and convenience is an important reason for its popularity. People don't have to bear the inconvenience of carrying cash. You can also buy the goods you want in advance.

However, such convenience also carries risks. Card issuers must determine to the maximum extent that the user is able to repay before the deadline, otherwise it will cause credit default, which is the last thing card issuers want to see. A large-scale credit default will not only lead to the bankruptcy of card issuers, but also destroy the economy and cause a social crisis.

Therefore, credit default prediction is very important for card issuers. It must predict whether customers will default in the future based on existing data. A good prediction model can not only help card issuers better control risks, but also enable users to have a good customer experience.

2. Applications

To make such predictions with modern computers, we need machine learning. You may have read a lot of reports about deep learning or machine learning, although many times they are given a broader Name: artificial intelligence. In fact, or fortunately, most programs do not need deep learning or artificial intelligence technology in a broader sense. For example, if we want to write a user interface for a microwave oven, we can design more than a dozen buttons and a series of rules that can accurately describe the performance of the microwave oven in various situations with only a little effort. For another example, suppose we want to write an e-mail client. Such a program is more complicated than a microwave oven, but we can still think step by step: the user interface of the client will need several input boxes to accept the recipient, subject, email body, etc. the program will listen to keyboard input and write them into a buffer, and then display them in the corresponding input boxes. When the user clicks the "send" button, we need to check whether the format of the recipient's email address is correct, and check whether the subject of the email is empty, or warn the user when the subject is empty, and then use the corresponding protocol to transmit the email.

It is worth noting that in the above two examples, we do not need to collect real-world data, nor do we need to systematically extract

the characteristics of these data. As long as there is enough time, our common sense and programming skills are enough for us to complete the task.

At the same time, we can easily find some simple problems that even the best programmers in the world can't solve with programming skills alone. For example, suppose we want to write a program to determine whether there is a cat in an image. It sounds simple, doesn't it? The program only needs to output "true" (meaning there is a cat) or "false" (meaning there is no cat) to each input image. But it is surprising that even the best computer scientists and programmers in the world do not know how to write such programs.

Where should we start? Let's simplify this problem further: if we assume that the height and width of all images are the same size of 400 pixels, and a pixel is composed of three values of red, green and blue, then an image is represented by nearly 500000 values. So what values hide the information we need? Is it the average of all the values, the values of the four corners, or a special point in the image? In fact, in order to interpret the content of the image, we need to look for features that only appear when thousands of values are combined, such as edges, texture, shape, eyes, nose, etc., and finally judge whether there is a cat in the

image.

One way of thinking to solve the above problems is reverse thinking. Instead of designing a program to solve the problem, it is better to start with the final requirements to find a solution. In fact, this is also the common core idea of current machine learning and deep learning applications: we can call it "programming with data". Instead of sitting in a room thinking about how to design a program to recognize cats, it is better to use the ability of human naked eyes to recognize cats in images. We can collect some real images with and without cats, and then our goal is to get a function that can infer whether there is a cat in the image from these images. The form of this function is usually selected according to our knowledge for a specific problem. For example, we use a quadratic function to determine whether there is a cat in the image, but the specific value of function parameters such as the coefficient value of the quadratic function is determined by data.

For credit default prediction, the customer data we have is the "picture", and the judgment of whether it will be credit default is the judgment of whether the picture is a cat or not. Humans can't find features from a large amount of data to make judgments, but machine learning can.

We are now in an era when program design is getting more and more help from deep learning. This can be said to be a watershed in the history of computer science. For example, deep learning is already in your mobile phone: spelling correction, voice recognition, recognizing friends in social media photos, etc. Thanks to excellent algorithms, fast and cheap computing power, unprecedented amounts of data and powerful software tools, most software engineers today have the ability to build complex models to solve problems that even the best scientists found difficult a decade ago.

3. List of papers

1. Goldberg, David E.. "Genetic Algorithms in Search Optimization and Machine Learning." (1988).
2. Neal, Radford M.. "Pattern Recognition and Machine Learning." *Technometrics* 49 (2007): 366 - 366.
3. Rasmussen, Carl Edward and Christopher K. I. Williams. "Gaussian Processes for Machine Learning." *Adaptive computation and machine learning* (2009).
4. Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Gilles Louppe, Peter Prettenhofer, Ron Weiss, Ron J. Weiss, J. Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot and E. Duchesnay. "Scikit-learn: Machine Learning in Python." *J. Mach. Learn. Res.* 12 (2011): 2825-2830.
5. Cho, Kyunghyun, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk and Yoshua Bengio. "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation." *EMNLP* (2014).

6. Bahdanau, Dzmitry, Kyunghyun Cho and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate." *CoRR* abs/1409.0473 (2015): n. pag.
7. Abadi, Martín, Paul Barham, Jianmin Chen, Z. Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu and Xiaoqiang Zhang. "TensorFlow: A system for large-scale machine learning." *ArXiv* abs/1605.08695 (2016): n. pag.
8. **สืบสิงห์, อนิรุธ**. "Data Mining Practical Machine Learning Tools and Techniques." *Journal of management science* 3 (2014): 92-96.
9. Dietterich, Thomas G.. "Machine learning." *ACM Comput. Surv.* 28 (1996): 3.
10. Quinlan, J. Ross. "C4.5: Programs for Machine Learning." (1992).