Machine Learning
ENGINEERING

Andriy Burkov

"In theory, there is no difference between theory and practice. But in practice, there is."

— Benjamin Brewster

"The perfect project plan is possible if one first documents a list of all the unknowns."

— Bill Langley

"When you're fundraising, it's AI. When you're hiring, it's ML. When you're implementing, it's linear regression. When you're debugging, it's printf()."

— Baron Schwartz

The book is distributed on the "read first, buy later" principle.

10 Conclusion

In 2020, machine learning has become a mature and popular tool for solving business problems. What previously was available only to a handful of organizations, and considered "magic" by others, can today be created and used by a typical organization.

Thanks to open-source code, crowdsourcing, easily accessible books, online courses, and publicly available datasets, many scientists, engineers, and even home enthusiasts may now train machine learning models. If you are lucky, your problem can be solved by writing several lines of code, as demonstrated in many online tutorials.

However, many things can go wrong in a machine learning project. Most are independent of the technology's maturity or the analyst's understanding of the machine learning algorithm.

Machine learning texts, online tutorials, and courses are devoted to explaining how machine learning algorithms work and how to apply them to a dataset. Your success will probably depend on other factors. What data you can get and whether you can get enough of it, how you prepare it for learning, what features you engineer, whether your solution is scalable, maintainable, cannot be manipulated by attackers, and doesn't make costly errors — these factors are much more important for an applied machine learning project.

Yet despite their magnitude, most modern machine learning books and courses often leave these aspects for self-study. Some provide only partial coverage, with just an application to solving a specific illustrative problem.

It's a significant gap in knowledge, and I tried to fill it with this book.

10.1 Takeaways

What do I hope the reader takes away after reading this book?

First of all, a strong understanding that all machine learning projects are unique. There's no single recipe that will always work. Most of the time, the greatest challenges must be solved before you type `from sklearn.linear_model import LogisticRegression`: you must define your goal, select a baseline, gather relevant data, get it labeled with quality labels, and transform labeled data into training, validation, and test sets. The rest of the problem is solved after you type `model.fit(X,y)`, by applying error analysis, evaluating the model, verifying it solves the problem, and works better than the existing solution.

The seasoned analyst or machine learning engineer understands that not all problems, business or otherwise, will be solved with machine learning. In fact, many problems can be solved more easily using a heuristic, a lookup in a database, or traditional software development. You probably should not use machine learning if the system's every action, decision, or behavior, must be explained. With rare exceptions, machine learning models are blackboxes. They will not tell you why they predicted what they predicted, nor why they didn't predict today what they predicted yesterday, nor how to fix these issues.

Furthermore, unless you can find a public dataset and an open-source solution providing exactly what you need, machine learning is not the right approach for the shortest time to market. Sometimes the data needed to train and maintain a model is too hard or even impossible to get.

On the other hand, training data may be synthetically generated by using oversampling and data augmentation. These techniques are often applied when the data exhibits imbalance.

Before you start collecting data, ask these questions: is your data accessible, sizeable, usable, understandable, and reliable? Good data contains enough information for modeling, has good coverage of production use cases and few biases, is big enough to allow generalization, and not a result of the model itself.

Or does your data come with high cost, bias, imbalance, missing attributes, and/or noisy labels? Data quality must be ensured before it's used for training.

The machine learning project life cycle consists of the following stages: goal definition, data collection and preparation, feature engineering, model training, evaluation, deployment, serving, monitoring, and maintenance. At most stages, data leakage may arise. The analyst must be able to anticipate and prevent it.

After data preparation, feature engineering is the second most important stage. For some data, such as natural language text, features may be generated in bulk by using techniques like bag-of-words. However, the most useful features are often handcrafted by the analyst domain knowledge. Put yourself into the "model's shoes."

Good features have high predictive power, can be computed fast, are reliable and uncorrelated. They are unitary, easy to understand and maintain. Feature extraction code is one of the most important parts of a machine learning system. It must be extensively and systematically tested.

Best practices are to scale features, store and document them in schema files or feature stores, and keep code, model, and training data in sync.

You can synthesize new features by discretizing existing features, clustering training examples, and applying simple transformations to existing features, or combining pairs of them.

Before starting to work on a model, make sure that data conforms to the schema, then split it into three sets: training, validation, and test. Define an achievable level of performance, and choose a performance metric. It should reduce the model performance to a single number.

Most machine learning algorithms, models, and pipelines have hyperparameters. They can significantly influence the result of learning. However, these are not learned from data. The analyst sets their values during the hyperparameter tuning. In particular, tweaking these values controls two important tradeoffs: precision-recall and bias-variance. By varying the complexity of the model, you can reach the so-called "zone of solutions," a situation where both bias and variance are relatively low. The solution that optimizes the performance metric is usually found in the neighborhood of the zone of solutions. Grid search is the simplest and most widely-used hyperparameter-tuning technique.

Instead of training a deep model from scratch, it can be useful to start with a pre-trained model. Using a pre-trained model to build your own is called transfer learning. The fact that deep models allow for transfer learning is one of its most important properties.

Training deep models can be tricky. Implementation errors can happen at many stages, from data preparation, to defining a neural network topology. It's recommended to start small. For example, implement a simple model using a high-level library. Apply the default hyperparameter values to a small normalized dataset fitting in memory. Once you have your first simplistic model architecture and dataset, temporarily reduce your training dataset even further, to the size of one minibatch. Then start the training. Make sure that your simple model is capable of overfitting this training minibatch.

Your machine learning system's performance may benefit from model stacking. Ideally, base models used for stacking are obtained from algorithms or models of a different nature, such as random forests, gradient boosting, support vector machines, and deep models. Many real-world production systems are based on stacked models.

Machine learning model errors can be either uniform, and apply to all use cases with the same rate, or focused, and apply to certain use cases more frequently. By fixing a focused error, you fix it once for many examples.

Model performance can be improved using the following simple, iterative process:

1. Build the model using the best values of hyperparameters identified so far.
2. Test the model by applying it to a small subset of the validation set.
3. Find the most frequent error patterns on that small validation set.
4. Generate new features, or add more training data to fix the observed error patterns.
5. Repeat until no frequent error patterns are observed.

The model must be carefully evaluated before deployment, and continuously afterwards. Perform an offline model evaluation when the model is initially trained, based on the historical data. An online model evaluation consists of testing and comparing models in the production environment, using online data. Two popular techniques of online model evaluation are A/B testing and multi-armed bandit. They allow us to determine whether the new model is better than the old one.

A model may be deployed following several patterns: statically (as a part of an installable software package), dynamically on the user's device or a server, or via model streaming. In addition, choose among strategies such as single deployment, silent deployment, canary deployment, and multi-armed bandit. Each pattern and strategy has its pros and cons, and should be chosen depending on your business application.

Algorithmic efficiency is also an important consideration for model deployment. Scientific Python packages like NumPy, SciPy, and scikit-learn were built by experienced scientists and engineers with efficiency in mind. They have many methods implemented in C for maximum efficiency. Avoid writing your own production code, when you can reuse a popular and mature library or package. For high efficiency, choose appropriate data structures and caching.

For some applications, the prediction speed is critical. In such cases, the production code is written in a compiled language, such as Java or C/C++. If a data analyst has built a model in Python or R, there are several options for production deployment: rewrite the code in a compiled programming language of the production environment, use a model representation standard such as PMML or PFA, or use a specialized execution engine such as MLeap.

Machine learning models are served in either batch or on-demand mode. When served on-demand, a model is usually wrapped into a REST API. Serving a machine is often done by using a streaming architecture.

When a software system is exposed to the real world, its architecture must be ready to effectively react to errors, change, and human nature. A model must be constantly monitored. Monitoring must allow us to make sure that the model is served correctly and its performance remains within acceptable limits.

It is important to log enough information to reproduce any erratic system behavior during future analysis. If the model is served to a front-end user, it's important to save the user's context at the moment of the model serving.

Some users can try to abuse your model to reach their own business goals. To prevent abuse, don't trust the data coming from a user, unless similar data comes from multiple users. Assign a reputation score to each user and don't trust the data obtained from users with low reputations. Classify user behavior as normal or abnormal, and make progressively longer pauses or block some users, if necessary.

Regularly update your model by analyzing users' behavior and input data, to make it more robust. Afterwards, run the new model against the end-to-end and confidence test sets. Make sure that the outputs are as before, or that the changes are as expected. Validate that the new model doesn't make significantly more costly errors. Ensure errors are distributed uniformly across user categories. It's undesirable if the new model affects negatively most users from a minority or specific location.

* * *

The book stops here, but your learning doesn't. Machine learning engineering is a relatively new field of software engineering. Thanks to online publications and open source, I'm sure that new best practices, libraries, and frameworks simplifying or solidifying the stages of data preparation, model evaluation, deployment, serving, and monitoring, will appear during the upcoming years. Subscribe to my mailing list on this book's companion website http://www.mlebook.com. You will regularly receive relevant links.

Please keep in mind that, like its predecessor The Hundred-Page Machine Learning Book, this book is distributed on the "read-first, buy-later" principle. This means that you may download the entire text of the book from its companion website and read before buying. If you're reading these concluding words from a PDF file, and cannot remember having paid for it, please consider buying the book. You may buy it from Amazon, Leanpub, and other major online book sellers.

10.2 What to Read Next

There are many great books on machine learning and artificial intelligence. Here, I will give you only several recommendations.

If you would like to get hands-on experience with practical machine learning in Python, there are two books:

- "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow" (2nd edition) by Aurélien Géron (O'Reilly Media, 2019), and
- "Python Machine Learning" (3rd edition) by Sebastian Raschka (Packt Publishing, 2019).

For R, the best choice is "Machine Learning with R" by Brett Lantz (Packt Publishing, 2019).

To get a deeper understanding of the underlying math behind various machine learning algorithms, I recommend,

- "Pattern Recognition and Machine Learning" by Christopher Bishop (Springer, 2006), and
- "An Introduction to Statistical Learning" by Gareth James et al. (Springer, 2013).

To get more detailed understanding of deep learning, I recommend,

- "Neural Networks and Deep Learning" by Michael Nielsen (online, 2005), and
- "Generative Deep Learning" by David Foster (O'Reilly Media, 2019).

If your ambitions go far beyond machine learning and you want to sweep the whole field of artificial intelligence, then "Artificial Intelligence: A Modern Approach" (4th Edition) by Stuart Russell and Peter Norvig (Pearson, 2020), known as AIMA, is your best book.

10.3 Acknowledgements

The high quality of this book would be impossible without volunteering editors. I especially thank the following readers for their systematic contributions: Alexander Sack, Ana Fotina, Francesco Rinarelli, Yonas Mitike Kassa, Kelvin Sundli, Idris Aleem, and Tim Flocke.

I thank scientific advisors, Veronique Tremblay and Maximilian Hudlberger, for the review and correction of the Model Evaluation chapter. I'm also grateful to Cassie Kozyrkov for her attentive and critical eye that allowed solidifying the section on statistical tests.

Other wonderful people to whom I am grateful for their help are Jean Santos, Carlos Azevedo, Zakarie Hashi, Tridib Dutta, Zakariya Abu-Grin, Suhel Khan, Brad Ezard, Cole Holcomb, Oliver Proud, Michael Schock, Fernando Hannaka, Ayla Khan, Varuna Eswer, Stephen Fox, Brad Klassen, Felipe Duque, Alexandre Mundim, John Hill, Ryan Volpi, Gaurish Katlana, Harsha Srivatsa, Agrita Garnizone, Shyambhu Mukherjee, Christopher Thompson, Sylvain Truong, Niklas Hansson, Zhihao Wu, Max Schumacher, Piers Casimir, Harry Ritchie, Marko Peltojoki, Gregory V., Win Pet, Yihwa Kim, Timothée Bernard, Marwen Sallem, Daniel

Bourguet, Aliza Rubenstein, Alice O., Juan Carlo Rebanal, Haider Al-Tahan, Josh Cooper, Venkata Yerubandi, Mahendren S., Abhijit Kumar, Mathieu Bouchard, Yacin Bahi, Samir Char, Luis Leopoldo Perez, Mitchell DeHaven, Martin Gubri, Guillermo Santamaría, Mustafa Murat Arat, Rex Donahey, Nathaniel Netirungroj, Aliza Rubenstein , Rahima Karimova, Darwin Brochero, Vaheid Wallets, Bharat Raghunathan, Carlos Salas, Ji Hui Yang, Jonas Atarust, Siddarth Sampangi, Utkarsh Mittal, Felipe Antunes, Larysa Visengeriyeva, Sorin Gatea, Mattia Pancerasa, and James Hoover.