# Best Practices for Improving Your Machine Learning and Deep Learning Models

Machine learning and deep learning models are everywhere around us in modern organizations. The number of AI use cases has been increasing exponentially with the rapid development of new algorithms, cheaper compute, and greater availability of data. Every industry has appropriate machine learning and deep learning applications, from banking to healthcare to education to manufacturing, construction, and beyond.

One of the biggest challenges in all of these ML and DL projects in different industries is model improvement. So, in this article, we're going to explore ways to improve machine learning models built on structured data (time-series, categorical data, tabular data) and deep learning models built on unstructured data (text, images, audio, video, or multi-modal).

## The strategy for improving ML / DL models

At this point, implementing ML and DL applications in business is still in its early days, and there is no single structured process that can guarantee success. However, there are some best practices that can minimize the likelihood of a failed AI project [1, 2, 3].

One of the main keys to success is model accuracy and performance. Model performance is mainly a technical factor, and for a number of machine learning and deep learning use cases, deployment doesn't make sense if the model isn't accurate enough for the given business use case.

| FACTORS | EXAMPLES |
|---|---|
| What is the business use case? | Recommendation, Search, Forecasting |
| What are the business metrics? | NPS, ARPU, CTR, MRR, LTV |
| What are the technical metrics? | F1-score = >0.95; Latency = <15ms |

| FACTORS | EXAMPLES |
|---|---|
| What kind of data is being used? | Structured – Numerical, categorical, temporal<br>Unstructured – Image, Text, Audio, Video |
| What is the machine learning problem formulation? | Regression, Classification – multi-class or multi-label, Clustering |
| What are the relevant ML and DL models? | Random Forest, XGBoost, CNNs, DNNs, Transformers |
| What hyperparameter optimization techniques to consider? | Grid search, Bayesian optimization |
| Additional constraints | Explainability, Fairness, Bias, Privacy |

***Table 1. A list of factors that govern the scope of improvement for a machine/deep learning model***

In the context of improving existing machine learning and deep learning models, there's no one-size-fits-all strategy that can be consistently applied. I will review a set of guidelines and best practices that can be evaluated to systematically identify potential sources of improvement in accuracy and model performance.

Table 1, above, shows a set of high-level factors that should be considered before starting to debug and improve ML and DL models. It highlights the crucial set of factors that underlie the business and technical constraints within which the machine learning or deep learning model has to be improved.

For example, a machine learning model for predicting credit rating of new retail banking customers should also be able to explain its decision in case the credit card application is rejected. Here, simply optimizing for the technical metric isn't enough if the model doesn't offer explainability and guidance for the customer to understand and improve their credit score.

For clarity's sake, in this article, I assume that your machine learning or deep learning model has already been trained on in-house data for a specific business use case, and the challenge is to improve the model performance on the same test set to meet the required acceptance criteria.

We're going to explore several methods to improve model performance, so you'll surely find one or two relevant to your use case. Ultimately, practice and experience working on a wide variety of models leads to better intuition about the best approaches to improve model accuracy and prioritize these techniques over others.

## Preliminary analysis

The first step in improving machine learning models is to carefully review the underlying hypotheses for the model in the context of the business use case, and evaluate the performance of the current models.

### (1) Review initial hypotheses about the dataset and the choice of algorithms

In an ideal scenario, any machine learning modeling or algorithmic work is preceded by careful analysis of the problem at hand including a precise definition of the use case and the business and technical metrics to optimize [1].

It's far too common to lose sight of the pre-defined data annotation guidelines, dataset creation strategies, metrics and success criteria once the exciting stage of building machine learning or deep learning models begins. However, keeping the larger picture in mind is beneficial to streamline and prioritize the iterative process of improving machine learning and deep learning models.

### (2) Is the model overfitting or underfitting?

This can be visualized as in Figure 1, below, by plotting the model prediction error as a function of model complexity or number of epochs. The difference between the training and test error curves shows overfitting, i.e., high variance and low bias or underfitting, i.e., high bias and low variance, and provides a useful proxy to understand the current state of the machine learning model.

If the model is overfitting, it can be improved by :

- using more training data,
- reducing model complexity,
- regularization methods including Ridge and Lasso regularization,
- dropout,
- early stopping.

If the model is underfitting, it can be addressed by making the model more complex, i.e., adding more features or layers, and training the model for more epochs.

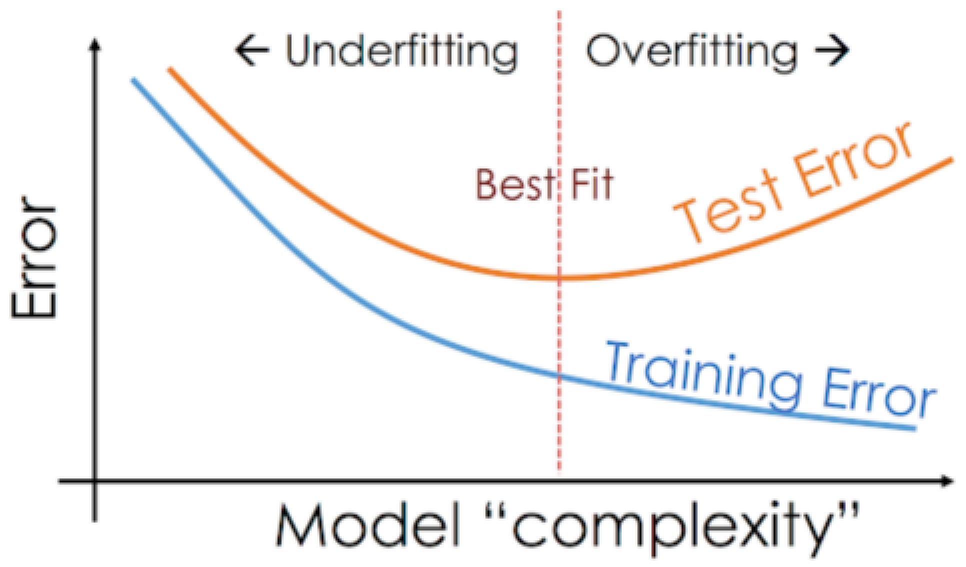


*Figure 1. Overfitting vs. Underfitting* | Source

## Learn more

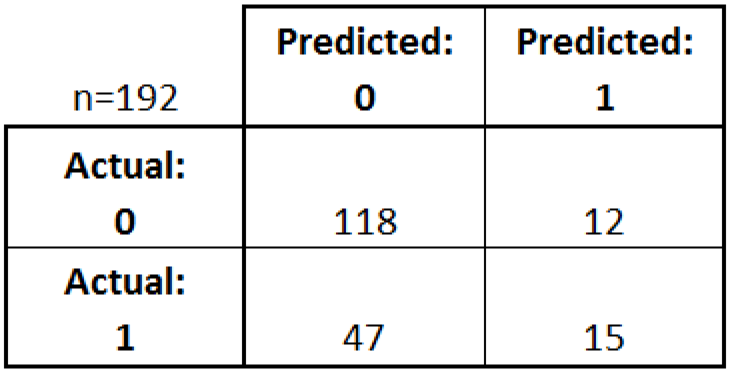
Overfitting vs Underfitting in Machine Learning – Everything You Need to Know

## (3) What kind of errors is the model making?

For a typical classification problem, this can be visualized using plots like the Confusion Matrix, which illustrates the proportion of Type 1 (false positive), and Type 2 (false negative) errors. Figure 2 shows a confusion matrix for a representative binary classification problem. In this example, we have 15 True Positives, 12 False Positives, 118 True Negatives, 47 False Negatives. So:

- Precision = 15/(15+12) = 15/27 = 0.56
- Recall = 15/(15+47) = 15/62 = 0.24
- F1-score = 2 * 0.56 * 0.34 / (0.56 + 0.34) = 0.42

Here, both precision and recall are on the lower side, and depending on the acceptance criteria, there is ample scope to improve both. Depending on the business use case and domain, it might make more sense to focus on improving recall compared to precision. This applies to several machine learning problems in domains like healthcare, finance, and education.



| n=192 | Predicted: 0 | Predicted: 1 |
|---|---|---|
| Actual: 0 | 118 | 12 |
| Actual: 1 | 47 | 15 |

*Figure 2. Illustration of Type 1 and Type 2 errors made by machine learning models.*

Figure 3 shows another representative confusion matrix for a multi-class classification problem, a common use case in industry applications. At first glance, it's clear to see that the model is confusing classes 1-5 with class 0, and in certain cases, it's predicting class 0 more often than the true class. This suggests that there's a systematic error in the model, most likely to do with class 0.

Armed with this insight, the first step to improve this model would be to check the labeled training examples for potential annotation errors or for the degree of similarity between the examples belonging to class 0 vs. classes 1-5. Potentially, this error analysis might show relevant evidence like the labels from a particular annotator being systematically mislabeled, accounting for the high confusion rate between the corresponding classes or categories.

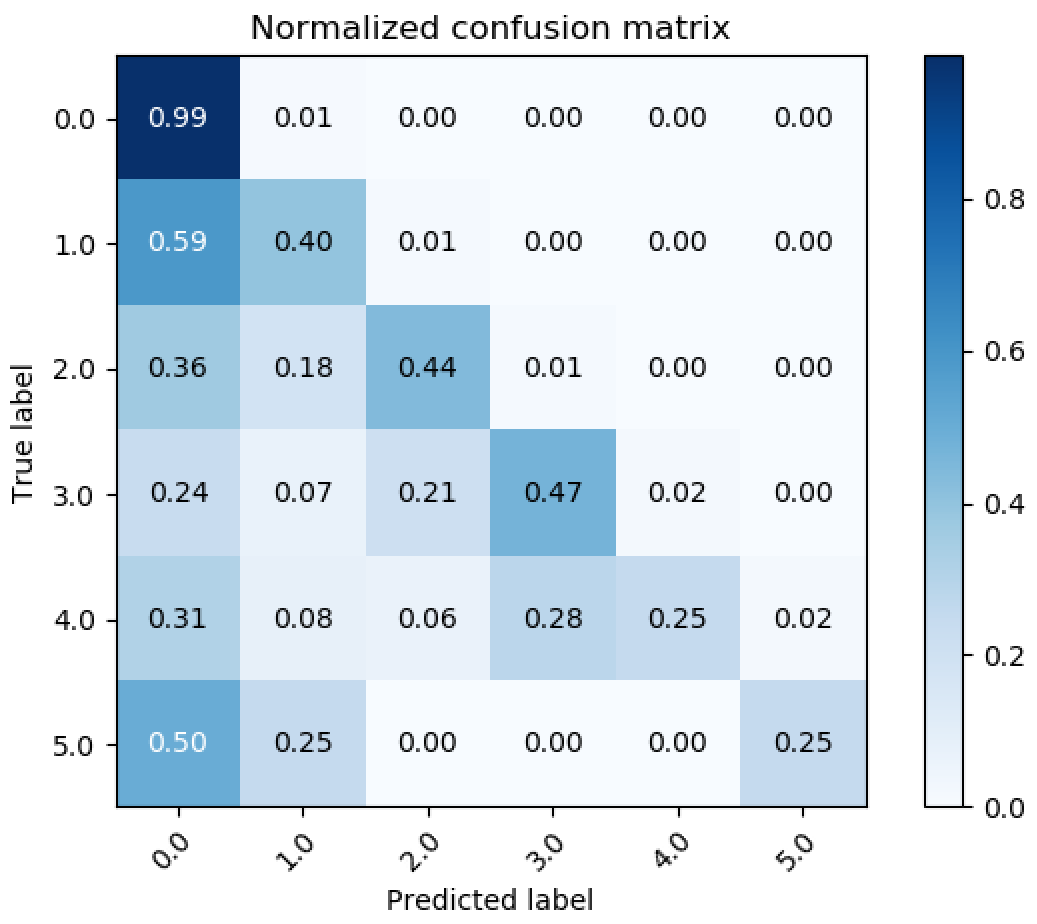


*Figure 3. Normalized confusion matrix for a multi-class classification problem* | Source

## Model optimization

After initial analysis and evaluation of model accuracy, visualization of key metrics to diagnose the errors, you should see if you can extract additional performance from the current model by retraining it with a different set of hyperparameters.

The assumption underlying a trained machine learning or deep learning model is that the current set of model weights and biases correspond to a local minima during the convex optimization process. Gradient descent should ideally yield a global minima that corresponds to the most optimal set of model weights.

However, gradient descent is a stochastic process that varies as a function of several parameters including how the weights are initialized, the learning rate schedule, the number of training epochs, any regularization method used to prevent overfitting, and a range of other hyperparameters specific to the training process and the model itself.

Each machine learning and deep learning model is based on a unique algorithm and intrinsic parameters. The goal of machine learning is to learn the best set of weights to approximate complex nonlinear functions from data. It's often the case that the first trained model is suboptimal and finding the optimal combination of hyperparameters can yield additional accuracy.

Hyperparameter tuning involves training separate versions of the models, each trained on a different combination of hyperparameters. Typically, for smaller machine learning models, it's a quick process and helps identify the model with the highest accuracy. For more complex models including deep neural networks, running several iterations of the same model on different combinations of hyperparameter values may not be feasible. In such cases, it's prudent to limit the range and choice of individual hyperparameter values based on prior knowledge or existing literature to find the most optimal model.

Three methods of hyperparameter tuning are most commonly used:

## (1) Grid Search

Grid search is a common hyperparameter optimization method that involves finding an optimal set of hyperparameters by evaluating all their possible combinations. It's most useful when the optimal range of relevant hyperparameters are known in advance, either based on empirical experiments, previous work, or published literature.

For instance, if you have identified 6 key hyperparameters and 5 possible values for each hyperparameter within a specific range, then grid search will evaluate 5 * 6 = 30 different models for each unique combination of hyperparameters. This ensures that our prior knowledge about the hyperparameter range is captured into a finite set of model evaluations.

The downside of this method is it's computationally expensive and it only samples from well-defined spaces in the high-dimensional hyperparameter grid. Therefore, as shown in Figure 4, it's more likely to miss the local minima associated with optimal hyperparameter values outside the pre-defined range. To alleviate these limitations of grid search, random search is recommended.

## (2) Random Search

Random search essentially involves taking random samples of the hyperparameter values, and is better at identifying optimal hyperparameter values that one may not have a strong hypothesis about [4]. The random sampling process is more efficient and usually returns a set of optimal values based on fewer model iterations. Therefore, random search is the first choice for hyperparameter optimization in many cases.

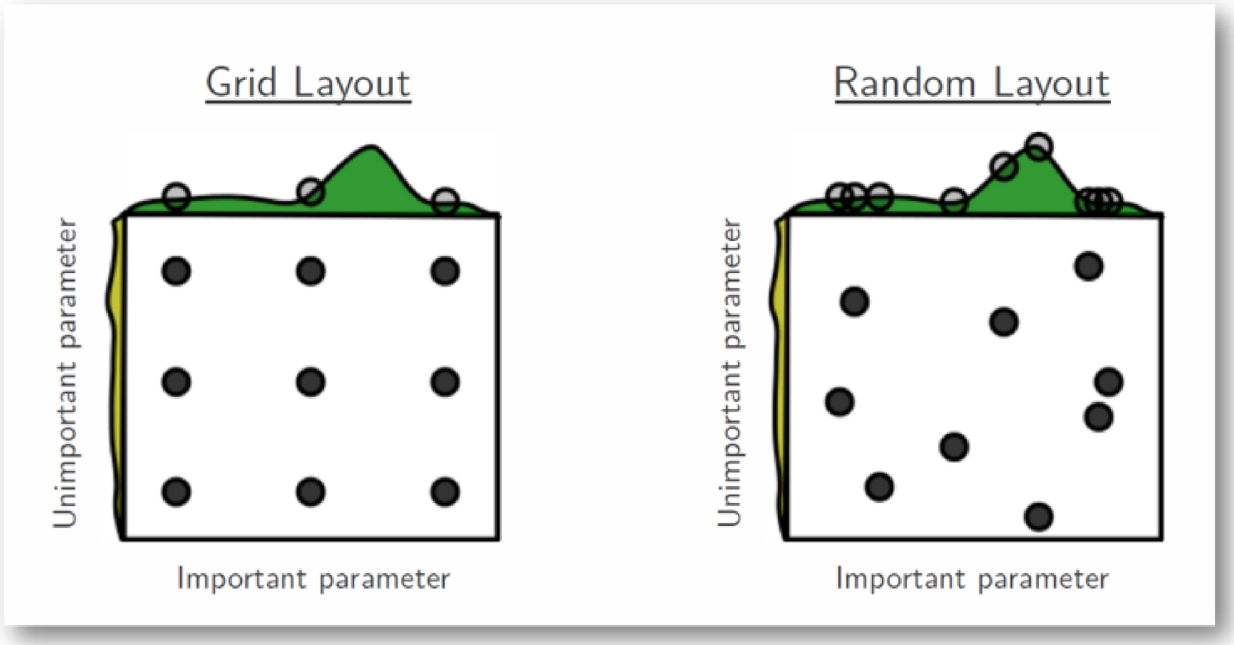


*Figure 4. A comparison of Grid search and Random search |* *Source*

## (3) Bayesian Search

Bayesian search is a sophisticated hyperparameter optimization method based on the Bayes Theorem [5]. It works by building a probabilistic model of the objective function, called the surrogate function, that is then searched efficiently with an acquisition function before candidate samples are chosen for evaluation on the real objective function. Bayesian Optimization is often able to yield more optimal solutions than random search as shown in Figure 5, and is used in applied machine learning to tune the hyperparameters of a given well-performing model on a validation dataset.

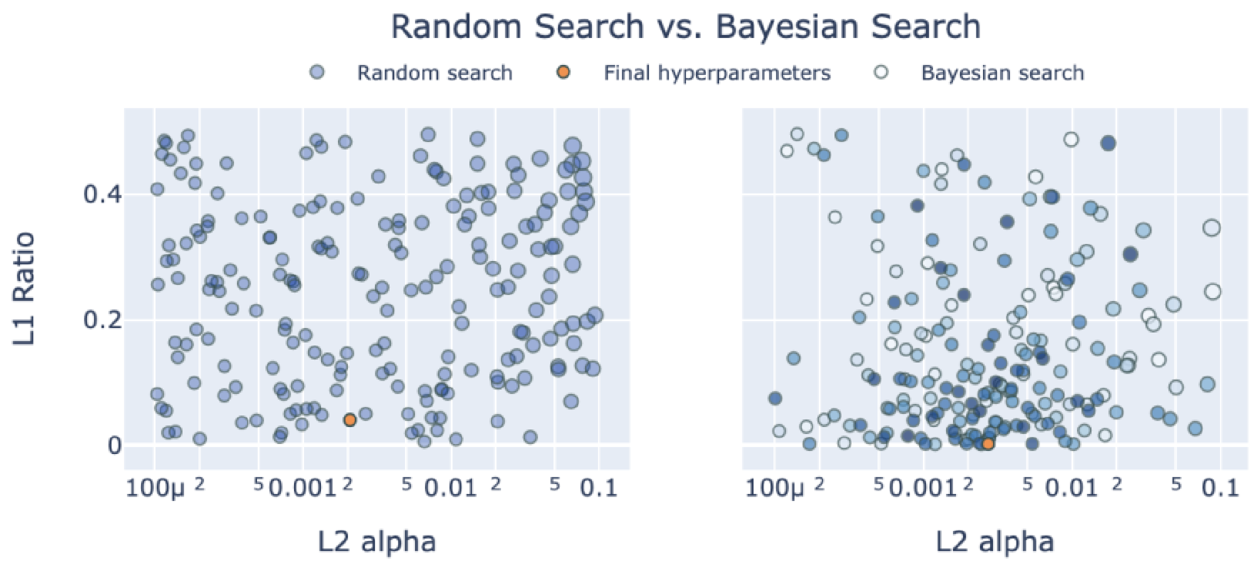


*Figure 5. A comparison of Random search and Bayesian search |* *Source*

## Models & algorithms

### (1) Establish a strong baseline model

To improve your machine learning or deep learning model, it’s important to establish a strong baseline model. A good baseline model incorporates all the business and technical requirements, tests the data engineering and model deployment pipelines, and serves as a benchmark for subsequent model development.

The choice of the baseline model is influenced by the particular application, the kind of dataset, and the business domain. For instance, for a forecasting application for time-series data from the financial domain, an XGBoost model is a strong baseline model. In fact, for several regression and classification based applications, Gradient Boosted Decision Trees are commonly used in production. Therefore, it makes sense to start with a model that is known to produce robust performance in production settings.

For unstructured data like images, text, audio, video, deep learning models are commonly employed across applications like object classification, image segmentation, sentiment analysis, chatbots, speech recognition, emotion recognition amongst others. Given the rapid advancement in the state-of-the-art performance of deep learning models, it’s prudent to use a more sophisticated model compared to an older one. For instance, for object classification, deep convolutional network models like VGG-16 or ResNet-50 should be the baseline, instead of a single layer convolutional neural network. As an example, for a face recognition application for CCTV image data from the security domain, a ResNet-50 is a strong baseline contender.

### (2) Use pre-trained models and cloud APIs

Instead of training a baseline model yourself, in certain cases, you can save valuable time and energy by evaluating pre-trained models. There are a variety of sources like Github, Kaggle, or APIs from cloud companies like AWS, Google Cloud, Microsoft Azure, specialized startups like Scale AI, Hugging Face, Primer.ai amongst others.

The advantage of using pretrained models or APIs is ease of use, faster evaluation, and savings in time and resources. However, an important caveat is that such pretrained models are often not directly applicable for your use cases, less flexible, and tricky to customize.

Using Transfer Learning, however, pretrained models can be applied to your use case by not retraining complex models afresh, and instead fine-tuning model weights on your specific dataset. For example, the intrinsic knowledge of an object classification model like ResNet-50 trained on several image categories from the ImageNet dataset can be leveraged to accelerate model development for your custom dataset and use case.

APIs are available for numerous use cases like forecasting, fraud, search, optical character recognition for processing documents, personalization, chat and voice bots for customer service, and others [6].

## (3) Try AutoML

While pretrained models are readily available, you can also investigate state-of-the-art AutoML technology for creating custom machine learning and deep learning models. AutoML is a good solution for companies that have limited organizational knowledge and resources to deploy machine learning at scale to meet their business needs.

AutoML solutions are provided by cloud services like Google Cloud Platform [7] as well as a number of niche companies and startups like H2O.ai. The promise of AutoML is yet to be seen at scale, but it represents an exciting opportunity to rapidly build and prototype a baseline machine learning or deep learning model for your use case and fast-track model development and deployment lifecycle.

## (4) Model improvements

Algorithmic and model-based improvements require greater technical expertise, intuition and understanding of the business use case. Given the limited supply of data scientists who combine all the above skills, it’s not common for most businesses to invest significant resources and allocate the necessary time and bandwidth for innovative machine learning and deep learning research and development.

As most business use cases and organizational data ecosystems are unique, a one-size-fits-all strategy is often not feasible nor advisable. This necessitates the requirement for original work to adapt existing or related applications to fit the businesses’ particular needs.

Model improvements can come from distinct sources:

- Choice of machine learning or deep learning model
- Hyperparameter tuning as described above
- Custom loss functions to prioritize metrics as per business needs
- Ensembling of models to combine relative strengths of individual models
- Novel optimizers that outperform standard optimizers like ReLu

## (5) Case Study: from BERT to RoBERTa

In this section, I will describe a case study in large-scale model improvement for a state-of-the-art deep learning model for natural language processing. BERT, developed in 2018 by Google [8], has become the de-facto deep learning model to use for a range of NLP applications and has accelerated NLP research and use cases across the board. It yielded state-of-the-art performance on benchmarks like GLUE, which evaluate models on a range of tasks that simulate human language understanding.

However, BERT's tenure at the top of the GLUE leaderboard was soon replaced by RoBERTa, developed by Facebook AI, which was fundamentally an exercise in optimizing the BERT model further, as evidenced by its full name – Robustly Optimized BERT PreTraining Approach [9].

RoBERTA surpassed BERT in terms of performance on the basis of simple modifications including training the model for more epochs, feeding more data to the model, training the model on different data (longer sequences) with bigger batch size, and optimizing the model and design choices. These simple model improvement techniques increased the model score on the GLUE benchmark from 80.5% for BERT to 88.5% for RoBERTa, a highly significant outcome.

## Data

In earlier sections, I discussed hyperparameter optimization and select model improvement strategies. In this section, I will describe the importance of focusing on the data to improve the performance of machine learning and deep learning models. In business, more often than not, improving the quality and quantity of training data yields stronger model performance. There are several techniques for a data-centric approach to machine learning and deep learning model improvement.

### (1) Data Augmentation

The lack of gold standard annotated training data is a common bottleneck for developing and improving large-scale supervised machine learning and deep learning models. The cost of annotation in terms of time, expense, and subject matter expertise is a limiting factor to create massive labeled training datasets. More often than not, machine learning models suffer from overfitting and their performance can be improved by using more training data.

Data augmentation techniques can be leveraged to expand the training dataset in a scalable fashion. The choice of data augmentation techniques depends on the kind of data. For instance, synthetic time-series data can be created by sampling from a generative model or probability distribution that is similar in summary statistics to the observed data. Images can be augmented by altering image characteristics like brightness, color, hue, orientation, cropping, etc.

Text can be augmented by a number of methods including regex patterns, templates, substitution by synonyms and antonyms, backtranslation, paraphrase generation, or using a language model to generate text.

Audio data can be augmented by modifying fundamental acoustic attributes like pitch, timbre, loudness, spatial location, and other spectrotemporal features. For specific applications, pretrained models can also be used to expand the original training dataset.

Recent methods based on weak supervision, semi-supervised learning, student-teacher learning, and self-supervised learning can also be leveraged to generate training data with noisy labels. These methods are based on the premise that augmenting gold standard labeled data with unlabeled or noisy labeled data provides a significant lift in model performance. It’s now possible to leverage a combination of rule-based and model-based data augmentation techniques that can be engineered at scale using data augmentation platforms like Snorkel [10].

Another common scenario where models underperform is in the context of imbalanced data across categories of interest. In such scenarios with skewed data distribution, upsampling and downsampling of data and techniques like SMOTE are helpful in correcting the modeling results.

The concept of having a training dataset, validation dataset, and test dataset is common in machine learning research. Cross-validation helps in shuffling the exact composition of these three datasets so that statistically robust inference can be made about the model performance.

While classical approaches focus on three datasets with a single validation dataset, it’s good to have two different validation datasets, one drawn from the same distribution as the training data and the other drawn from the same distribution as the test data. This way you can better diagnose bias-variance tradeoff and use the right set of model improvement strategies as described above.

## (2) Feature engineering & selection

Typical machine learning models are trained on data with numerous features. Another common technique to improve machine learning models is to engineer new features and select an optimal set of features that better improve model performance. Feature engineering requires significant domain expertise to devise new features that capture aspects of the complex nonlinear function that the machine learning model is learning to approximate. So, this method is not always feasible if the baseline model already captures a diverse set of features.

Feature selection via programmatic approaches can help remove some correlated or redundant features that don't contribute much to model performance. Methods to iteratively build and evaluate a model with a progressively increasing set of features, or iteratively reducing one feature at a time from a model trained with the entire set of features, help in identifying robust features.

### (3) Active learning

Analysis of model errors can shed light on the kind of mistakes that the machine learning model makes. Reviewing these errors helps understand whether there are any characteristic patterns that can be addressed by some of the techniques described above.

Additionally, active learning methods that focus on model mistakes that are closer to the decision boundary can provide a significant boost in performance once the model is already in production. In active learning, the new examples that the model is confused about and predicts incorrectly are sent for annotation to domain experts who provide the correct labels. This dataset that is reviewed and annotated by experts is incorporated back into the training dataset to help the retrained model learn from its previous errors.

## Conclusion

Machine learning and deep learning modeling requires significant subject matter expertise, access to high-quality labeled data, as well as computational resources for continuous model training and refinement.

Improving machine learning models is an art that can be perfected by systematically addressing the deficiencies of the current model. In this article, I have reviewed a set of methods focused on models, their hyperparameters, and the underlying data to improve and update models to attain the required performance levels for successful deployment.

### References

- [1] https://neptune.ai/blog/building-ai-ml-projects-for-business-best-practices
- [2] https://www.sundeepteki.org/blog/why-corporate-ai-projects-fail-part-14
- [3] https://www.sundeepteki.org/blog/why-corporate-ai-projects-fail-part-24
- [4] Bergstra and Bengio (2012) Random search for hyper-parameter optimization.
- *The Journal of Machine Learning Research* 13: pp 281-305
- [5] Snoek et al. (2012) Practical Bayesian Optimization of Machine Learning Algorithms. NIPS '12. 2: pp 2951-2959
- [6] https://aws.amazon.com/machine-learning/ml-use-cases/

- [7] https://cloud.google.com/automl
- [8] Devlin et al. (2018) BERT: Pre-Training of Deep Bidirectional Transformers for
- Language Understanding. https://arxiv.org/abs/1810.04805
- [9] Liu et al. (2019) RoBERTa: A Robustly Optimized BERT PreTraining Approach. https://arxiv.org/abs/1907.11692
- [10] Ratner et al. (2017). Snorkel: Rapid Training Data Creation with Weak Supervision. https://arxiv.org/abs/1711.10160