Chapter 3

Training Data

When it comes to machine learning, most of us get excited about building the models and that's the fun part for us. But, it's challenging to make it work, seeing it improve, and watching it learn. But let's be real, the real work is in the data. Dealing with messy, incomplete, or poorly organized data that won't even fit into our machine's memory. That's the tough part. Data is unpredictable, and if we don't handle it right, it can ruin our whole project. That's why learning how to work with data is so important. A big challenge is getting the right training data. It's not just about picking a random data set and throwing it into our model. we must carefully choose the data for training, testing, and validating our model. we'll face a lot of problems like labels not matching up, missing labels, imbalanced data, or just not having enough data in the first place. Also, "training data" isn't the same as a "training dataset." A dataset sounds like something fixed, but real-world data isn't like that. It changes over time. So as our model improves, we'll probably need to update our training data too. One more thing data is full of biases. These can come from how data is collected, sampled, or labeled. Even if the data looks clean, it might still carry biases from the past or human decisions.

Sampling is a crucial but often overlooked part of the ML process. It comes up at many stages, like creating training data from real-world data, splitting data into training, validation, and test sets, or even monitoring your ML system. In this case, we're focusing on how sampling works when creating training data, but these methods apply to other parts of the ML workflow too. Sampling is necessary in a lot of situations. Understanding the different sampling methods helps in two ways: it can help avoid biases that might affect our results, and it lets us choose the right method to make our sampling process faster and more efficient. There are two main types of sampling: nonprobability sampling and random sampling. Nonprobability sampling is when we select data without using any probability-based method. There are some types of nonprobability sampling, like Convenience sampling, picking data that's easiest to get. It's fast and convenient but not necessarily representative.

Snowball sampling, starting with a small sample, and then using that to find more samples. Judgment sampling, Experts decide which samples are important or useful. Quota sampling, selecting data based on specific quotas.

Simple Random Sampling is the most basic type of random sampling, where every member of the population has an equal chance of being selected. Stratified Sampling, instead of just randomly selecting from the whole population, first we break the data into groups (called strata) that we care about, then sample each group separately. Weighted Sampling is a method where we give each sample a weight, which determines the chance of it being selected. Reservoir Sampling is a cool algorithm used when dealing with streaming data, like tweets, where we don’t know how much data we’ll get, and it’s too much to fit in memory. The goal is to sample a certain number (k) of items from a stream, ensuring each item has an equal chance of being selected, even though we don’t know the total number in advance.

Importance Sampling is a powerful method that helps us sample from a distribution when it's hard, slow, or expensive to sample directly from it. Labeling data is another major challenge in machine learning. Most models in production require labeled data to learn, and the quality of the model depends a lot on how good the labels are. Label Multiplicity happens when we have multiple conflicting labels for the same data because different data sources label things differently. Data Lineage is tracking where our data comes from and who labeled it.

Natural Labels it is one of the great methods in ML because they don’t require manual labeling. These are labels that come automatically from the system, often from the model's own predictions, such as Google Maps. Natural labels are commonly used in companies.

Feedback Loop Length refers to how long it takes for feedback to come in after a model makes a prediction. Short feedback loops mean we get labels quickly, such as recommender systems. Types of User Feedback differ in how quickly they come in and how strong the signal is. For long feedback loops, like fraud detection, labels might not appear until months later.

Getting high-quality labeled data can be tough, but there are several techniques to get it right, such as Weak Supervision, Semi-Supervision, Active Learning and Transfer Learning. weak supervision allows us to automate labeling using rules and heuristics, reducing the need for expensive hand-labeled data. If hand labeling is too difficult, weak supervision is a good method for it. While weak supervision relies on noisy heuristics for labeling, semi-supervision uses structural assumptions to generate new labels based on a small initial set of labeled data. Unlike weak supervision, semi-supervision requires a small set of labeled examples to start the learning process. Transfer learning is when we use a model trained for one task as a starting point for a new task. The base model is usually trained on a task with plenty of data, often without labels. Active learning is a technique to make labeling data more efficient. Instead of randomly labeling data, we focus on labeling the samples that will help the model learn best.

Class imbalance occurs when one class in our dataset has significantly more samples than another, making it harder for models to learn from the minority class. For instance, if we have data about cats and dogs, and nearly 95% of the data are cats, so this will lead to a huge problem later on, as the model will not figure out the data that is related to dogs, and it may miss predict them as cats as well. There are some challenges of Class Imbalance, such as nsufficient Signal for Minority Classes, Nonoptimal Solutions, Asymmetric Error Costs. In real-world tasks, class imbalance is common, especially when rare events are more crucial to detect, such as fraud detection, churn prediction, disease screening, and resume screening. Imbalance can also arise from bias in sampling or labeling errors.

Class imbalance is a common issue in real-world tasks, and many methods have been developed to address it. Its impact depends on the problem's complexity, simple, linearly separable problems are usually unaffected by class imbalance, while more complex ones are more sensitive. Binary classification problems are easier to handle than multiclass ones, and deeper neural networks like more than 10 layers tend to perform better on imbalanced data than shallower ones. There are some strategies for managing class imbalance, such as Choosing the Right Metrics, Data-Level Methods, Algorithm-Level Methods.

Data-level methods adjust the distribution of training data to address class imbalance, making it easier for models to learn. One common approach is resampling, which involves either oversampling the minority class which is adding more samples or undersampling the majority class which is removing samples to balance it.

Algorithm-level methods tackle class imbalance by modifying the learning algorithm itself, without changing the distribution of the training data. These methods often focus on adjusting the cost function, which guides the learning process.

There are some techniques to Handle Class Imbalance, such as Cost-Sensitive Learning, Class-Balanced and LossFocal Loss. These methods help mitigate the effects of class imbalance by making the algorithm more sensitive to the important.

Data augmentation is a set of techniques designed to increase the amount and variety of training data, making models more robust. While it’s traditionally used when training data is scarce, it’s also beneficial when we have plenty of data, as it helps models become more resilient to noise and adversarial attacks. Data augmentation is commonly used in computer vision and it is applied in natural language processing (NLP). There are three main types of augmentation, Simple Label-Preserving Transformations, Perturbation and Data Synthesis.

Training data is the core foundation of machine learning (ML). No matter how advanced the algorithms are, poor training data will lead to poor performance. It's crucial to invest time in curating and preparing good training data to help algorithms learn effectively. Finally, once our training data is ready, the next step is to extract features from the data to train your ML models.

Chapter 4

Feature Engineering

Feature engineering is critical in machine learning (ML) because it can significantly boost model performance. Even with advanced algorithms, having the right features often leads to better results than complex techniques like hyperparameter tuning. In fact, poor features can cause state-of-the-art models to underperform. While deep learning can automatically extract features from raw data, such as mages and text, traditional feature engineering is still needed for many tasks, especially non-deep learning ones or structured data problems like sentiment analysis, fraud detection, or recommendations. we still need to do feature engineering to our data before start working with it.

In text tasks, manual feature engineering used to involve steps like lemmatization and creating n-grams, such as breaking down a sentence like "I like food" into smaller parts, but this is something may take sometime to complter. However, deep learning now automates much of this through tokenization and word embeddings, reducing the need for manual steps, which will save our time. For images, deep learning also eliminates the need for manual feature extraction, such as detecting shapes, allowing raw images to be input directly into models like CNNs.

Feature engineering is an iterative process that requires domain knowledge and can be time-consuming. While deep learning has made some of the process easier, it's still essential for many applications, especially when dealing with complex or specialized data.

There are some common Feature Engineering Operations we can use, such as handling missing values, scaling, discretization, encoding categorical features, and generating cross or positional features. Missing values can be tricky and come in three types: *Missing Not at Random (MNAR)*, where the missingness is related to the true value; *Missing at Random (MAR)*, where missingness depends on another variable; and *Missing Completely at Random (MCAR)*, where no pattern exists.

For handling missing values, you can either delete data or impute missing values. Deletion may involve removing rows or columns with too many missing values, but this can lead to losing important information that we may need and be useful as well or bias. Also, imputation fills missing values with the mean, median, or mode, but it can introduce its own biases or errors. It's important to avoid using values that could confuse the model, like filling in missing data with an unlikely value, such as filling 0 for the age which is not acceptable.

Scaling is crucial in ML when features have different ranges or units, like age (20-40) and income ($10,000 to $150,000). If these features aren't scaled, the model might wrongly assign more importance to the larger values. Feature scaling standardizes the range of data, often scaling it to 0, 1 or -1, 1. Standardization, or normalization, adjusts features to have zero mean and unit variance, useful when data is normally distributed. A log transformation can also help reduce skewness in data, but it doesn't always work for all situations. We have to ensure that scaling doesn't lead to data leakage when statistics.

Discretization is a technique that turns continuous data (like income) into categorical bins, such as low, middle, high income. It simplifies learning by reducing the number of categories the model needs to consider. However, it can introduce issues at the category boundaries, where slight differences in data values might be treated as distinct, even if they are essentially the same. Discretization is more useful with limited data but requires careful selection of bin boundaries.

In ML, dealing with categorical data, like product brands or user accounts, can be tricky because the number of categories might change over time. Initially, categories might seem static, but in production, new categories, such as new brands on Amazon or new users keep appearing, which can cause models to fail or perform poorly. Also, the model may predict "UNKNOWN" for the new category for the unseen data. A more advanced solution is the hashing trick, where categories are hashed into a fixed range of values, such as 262,144 possible values, allowing models to handle new, unseen categories by assigning them random hash values. Although the hashing trick is often seen as a "hack" by academics, it's widely used in the industry for handling dynamic categories.

Feature crossing involves combining two or more features to create new ones that can help model nonlinear relationships. For example, combining "marital status" and "number of children" to create a new feature, "marriage and children," can capture interactions between these features. This technique is useful for models that struggle with nonlinear relationships. The main downside of feature crossing is that it can increase the number of features dramatically, potentially leading to overfitting and requiring much more data to train effectively.

Positional embeddings are used to encode the order of elements, which is especially important in models like transformers that process input data in parallel, such as words in a sentence. Fourier features which is using sine and cosine. It can be used for positional embedding, which has been shown to improve model performance. This technique is effective in domains like computer vision, where continuous positional information is required.

Data Leakage occurs when information from the label which is the thing we're trying to predict "leaks" into the features that the inputs to the model, which shouldn't happen because this information wouldn't be available during actual predictions. This leads to overfitting, where the model performs well during testing but fails in real-world production. For example, in Covid-19 Prediction, A model that predicts Covid-19 risk from medical scans learned to predict based on the patient's position, such as lying down vs. standing, which was not a valid feature.

There are some Common Causes of Data Leakage, such as plitting time-correlated data randomly, Scaling before splitting which we have to split first, Filling missing data with statistics from the entire dataset, Data duplication, Group leakage, Leakage from the data generation process.

Data leakage can occur at various stages of an ML project, from data generation to processing and feature engineering. It's crucial to monitor for leakage throughout the project lifecycle. There are some methods to detect Data Leakage, such as Check Feature Correlation, Ablation Studies, Monitor New Features, and Test Split Caution.

Adding more features doesn’t always improve model performance. Too many features can lead to several problems, such as Data leakage risk, Overfitting, Increased memory usage and inference latency, and Technical debt.

We have to onsider two main factors, importance and generalization, if a feature is useful. In conclusion, effective feature engineering requires careful detection of leakage, an understanding of feature importance, and ensuring that features generalize well across data splits.