Building Data-Centric Systems for Machine Learning Development and Operations

DISS. ETH NO. 28434

Building Data-Centric Systems for Machine Learning Development and Operations

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH

(Dr. sc. ETH Zurich)

presented by

CÉDRIC BENJAMIN RENGGLI

MSc in Computer Science, ETH Zurich

born on 03.02.1991

citizen of Malters, LU, Switzerland

accepted on the recommendation of

Prof. Dr. Ce Zhang (ETH Zurich), examiner
Prof. Dr. Gustavo Alonso (ETH Zurich), co-examiner
Prof. Dr. James Zou (Stanford University), co-examiner
Prof. Dr. Sebastian Schelter (University of Amsterdam), co-examiner

2022

DS3Lab
Institute for Computing Platforms
ETH Department of Computer Science

CÉDRIC BENJAMIN RENGGLI

A dissertation submitted to
ETH Zurich
for the degree of Doctor of Sciences

DISS. ETH NO. 28434

examiner:
Prof. Dr. Ce Zhang
co-examiners:
Prof. Dr. Gustavo Alonso
Prof. Dr. James Zou
Prof. Dr. Sebastian Schelter

Examination date: May 20, 2022

Building Data-Centric Systems for Machine Learning Development and Operations

Abstract

Developing machine learning (ML) models can be seen as a process similar to the one established for traditional software development. Over the last years, practitioners have started adopting well-established concepts of classical software engineering to ML projects. One prominent example is MLOps, where tools and techniques from development and operations (DevOps) are transferred to ML to shorten the system development life cycle and provide continuous delivery into production while ensuring high quality of the artifacts. A key difference between classical software and ML development lies in the strong dependence of ML models on the data used to train or evaluate their performance. Therefore, many tools and best-practices of DevOps can not be directly applied to ML workloads or could lead to lower quality if blindly taken over. In this thesis, we provide three novel data-centric solutions to support data scientists and ML engineers to handle ML workloads in a principled and efficient way:

1. We present `Ease.ML/Snoopy`, designed to perform a systematic and theoretically founded *feasibility study* before building ML applications. We approach this problem by estimating the irreducible error of the underlying task, also known as the Bayes error rate (BER), which stems from data quality issues in datasets used to train or evaluate ML model artifacts. We design a practical Bayes error estimator by aggregating over a collection of 1NN-based estimator over publicly available pre-trained feature transformations. Furthermore, by including our systematic feasibility study with additional signals into the iterative label cleaning process, we demonstrate in end-to-end experiments how users are able to save substantial labeling time and monetary efforts.

2. We propose `SHiFT`, the first task-aware (i.e., taking the dataset into account), flexible, and efficient model search engine for transfer learning, which can be seen as a data- and compute-efficient alternative to training models from scratch, in analogy to *code-reuse* in classical software engineering. The emergence of rich model repositories, such as Tensor-

Flow Hub, enables practitioners and researchers to unleash the potential of these models across a wide range of tasks. As these repositories keep growing exponentially, efficiently selecting a good model for the task at hand becomes paramount. By carefully comparing various selection and search strategies, we realize that no single method outperforms the others, and hybrid or mixed strategies can be beneficial. The flexibility and efficiency of `SHiFT` are enabled by a custom query language `SHiFT-QL` together with a cost-based decision maker. Motivated by the iterative nature of machine learning development, we further support efficient incremental executions of our queries, which requires a careful implementation when jointly used with our optimizations.

3. We introduce `Ease.ML/CI`, the first *continuous integration* system for machine learning with statistical guarantees. The challenge of building `Ease.ML/CI` is to provide these rigorous guarantees (e.g., single accuracy point error tolerance with 0.999 reliability) with a practical amount of labeling effort (e.g., 2000 labels per test). We design a declarative scripting language that allows users to specify integration conditions with reliability constraints, and develop simple novel optimizations that can lower the number of labels required by up to two orders of magnitude for test conditions commonly used in real production systems.

ZUSAMMENFASSUNG

Die Entwicklung von Modellen für maschinelles Lernen (ML) ist ein ähnlicher Prozess wie in der traditionellen Softwareentwicklung. In den letzten Jahren haben Praktiker damit begonnen, etablierte Konzepte der klassischen Softwareentwicklung für ML-Projekte zu übernehmen. Ein prominentes Beispiel ist MLOps, bei welchem Tools und Techniken für die Entwicklung und den Betrieb (DevOps) von Code auf ML Artefakte übertragen werden. Damit wird der Lebenszyklus der Systementwicklung verkürzt, und eine kontinuierliche Bereitstellung für die Produktion bei gleichzeitiger Sicherstellung einer hohen Qualität der Artefakte gewährleistet. Ein wesentlicher Unterschied zwischen der Entwicklung von klassischer Software und ML Modellen liegt in der starken Datenabhängigkeit des ML Modells zum Trainieren oder um seine Leistung zu bewerten. Daher sind viele Tools und empfohlene Vorgehensweisen von DevOps entweder nicht auf ML-Workloads anwendbar, oder könnten bei blinder Übernahme zu geringerer Qualität führen. In dieser Arbeit identifizieren bieten wir drei neuartige datenzentrische Lösungen zur prinzipientreuen und effizienten Unterstützung an Data Scientists und ML-Ingenieure:

1. Wir präsentieren `Ease.ML/Snoopy`, welches erlaubt eine systematische und theoretisch fundierte *Machbarkeitsstudie* vor dem Entwickeln von ML-Anwendungen durchzuführen. Dabei schätzen wir den irreduziblen Fehler des zugrunde liegenden Problems, der auch als Bayes-Fehlerrate (BER) bekannt ist. Dieser Fehler ist auf Datenqualitätsprobleme in Datensätzen zurückzuführen, welche das hochwertige Trainieren oder Evaluieren von ML Modell Artefakten verhindert. Wir entwerfen einen praktischen Schätzer für die Bayes-Fehlerrate, indem wir über eine Zusammenstellung von 1NN-basierten Schätzern aggregieren, welche frei verfügbar vor-trainierte Transformationen verwenden. Darüber hinaus, indem wir unsere systematische Machbarkeitsstudie mit zusätzlichen Signalen ausstatten, reduzieren wir Labeling und Geldaufwand für unsere Anwender erheblich.

2. Wir schlagen SHiFT vor, die erste problembewusste (d.h. unter Berücksichtigung des Datensatzes), flexible und effiziente Modell-Suchmaschine für Transfer Learning, was eine daten- und rechen-effiziente Alternative zum Training von ML Modellen von Grund auf representiert, eine Analogie zur *Code-Wiederverwendung* in der klassischen Software-Entwicklung. Das Aufkommen umfangreicher Modell-Repositories wie TensorFlow Hub ermöglicht Praktikern und Forschern das Potenzial dieser Modelle für eine breite Palette von Problemen auszuschöpfen. Da diese Repositories weiterhin exponentiell wachsen, ist eine effiziente Auswahl von guten Modellen für die anstehende Aufgabe von grösster Bedeutung. Durch den sorgfältigen Vergleich verschiedener Auswahl- und Suchstrategien erkennen wir, dass keine einzelne Methode die anderen übertrifft, und hybride oder gemischte Strategien sehr vorteilhaft sein können. Diese Flexibilität und Effizient von SHiFT werden durch eine benutzerdefinierte Abfragesprache SHiFT-QL zusammen mit einem kostenbasierten Entscheidungsträger für Optimierungen ermöglicht. Motiviert durch die iterative Natur der Entwicklung von ML Modellen, unterstützen wir ausserdem effiziente inkrementelle Ausführungen unserer Abfragen, was eine sorgfältige Implementierung erfordert, wenn sie gemeinsam mit unseren Optimierungen verwendet wird.

3. Wir stellen Ease.ML/CI vor, das erste kontinuierliche Integrationssystem für maschinelles Lernen mit statistischen Garantien. Die Herausforderung Ease.ML/CI zu bauen, liegt in den rigorosen Garantien (z.B. Fehlertoleranzen mit einer Genauigkeit von 0,999) mit praktikablem Labeling Aufwand (z.B. 2000 Labels pro Evaluierung). Wir entwerfen eine deklarative Skriptsprache, die es unseren Benutzern ermöglicht Integrationsbedingungen zu spezifizieren. Wir entwickeln dafür einfache neuartige Optimierungen, welche die Anzahl der erforderlichen Labels zum Überprüfen von Testbedingungen um bis zu zwei Größenordnungen verringern.

Acknowledgments

First and foremost, I would like to sincerely thank Ce for all the coaching and support that enabled me to become a much better researcher. Ce gave me the all the freedom and guidance needed to fulfill this successful PhD. He also helped me with career advice and always supported me in whatever professional goals I envisioned. I could not have asked for a better advisor.

I am also grateful for all the other members of the committee for taking the time to assess this thesis and their valuable feedback. I am grateful for James giving me feedback and advice for our ongoing project to write a book on the foundation of MLOps. I thank Gustavo for the advice given numerous times during my PhD and Sebastian for the in-depth pass on my thesis.

During my PhD, I was lucky to collaborate with many people outside ETH. Thanks to André, Carlos, Joan, Mario, Neil, Noé, and Jeremiah for the great time I could spend at Google and the fruitful collaboration. A special thank goes to Diogo from the University of Waterloo, and Wentao from Microsoft which supported me on many projects.

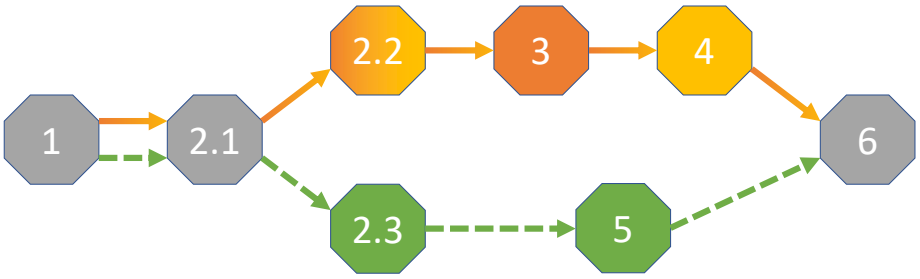
I also want to thank my colleagues from ETH, which became some of my best friends. Luka, Nora, Merve, Thomas, and Fabio, as well as Bojan and Vojislav, thank you for proofreading this thesis and all the fun time we had during and off work. I also had the privilege of supervising very talented students, especially Alexander, Luka, Thórhildur, and Xiaozhe, which managed to write papers out of their work.

I am very grateful for my many friends and family, for their support and, possibly even more importantly, for distracting my mind from research when I needed it most. Lastly, but certainly not the least, I would like to thank Sabrina, for her endless love, and support during my PhD and my future career. Looking forward to the journey ahead of us.

Zurich, May 2022

The results outlined in this dissertation stem from various previous works published as first or co-first author (Renggli et al., 2019a,b, 2020, 2021a,b, 2022a,b,c; Rimanic et al., 2020). Some descriptions are taken directly from these papers. Although this dissertation bears my name, the papers are joint efforts with different authors, most notably: B. Ding, N. M. Gürel, N. Hollenstein, B. Karlaš, A. Klimovic, L. Kolar, B. Li, F. Liu, M. Lucic, A. S. Pinto, J. Puigcerver, L. Rimanic, C. Riquelme, W. Wu, X. Yao, and C. Zhang. This collection of work would not have been possible without the contributions of all these collaborators.

Reading Threads



The figure above shows the possible reading threads of this thesis. If the reader is more interested in the development part of ML models, please follow the solid line. If the reader wants to focus on the operations part with focus on continuous testing, please follow the dashed line. Otherwise, the thesis can simply be read in linear order.

Contents

Chapter 1

Introduction

Machine learning (ML) has matured and found its successful application in many domains over the last decades (LeCun et al., 2015; Pouyanfar et al., 2018). Years of research have focused on speeding up the training process (Liu et al., 2020) and making it more automatic (He et al., 2021). It is now much easier to feed data into an automated machine learning (AutoML) system and get trained ML models at a reasonable price. However, this runs the risk of *wasting* resources, often linked to monetary expenses, if noisy data is blindly fed into AutoML systems, or the trained models are put into production without care.

Figure 1.1: Stages in a machine learning lifecycle (i.e., MLOps).

To alleviate these risks, practitioners have started adopting well-established concepts from traditional software engineering to ML workload. One such prominent example is *MLOps*. In a nutshell, by following the principles of DevOps (Ebert et al., 2016), one strives to ensure fast and robust development and operational cycles.[1] Figure 1.1 illustrates the four classical parts of an ML lifecycle. Admittedly, there is a major difference between classical software and ML.

[1] We refer to MLOps as both the development an operational phase of the ML lifecycle in this thesis.

In contrast to classical software, ML models are artifacts compiled from data, carrying the inherent randomness of ML tasks and its finite-sample data dependency (Karpathy, 2017). Not surprisingly, many challenges of getting high-accuracy ML models, or using them in a reliable and statistically rigorous manner, originate from improper data-management or usage (Polyzotis et al., 2018). Unfortunately, existing software engineering principles including DevOps tools and techniques, which are purposely designed for classical software artifacts, do not encompass the ML specific challenges and can thus be ill-suited for ML development and operations. Alternatively, blindly using these systems and neglecting the data dependency can lead to lower quality or catastrophic failures in production.

There is surprisingly little work focusing on the support of the lifecycle of ML models. Some prominent exceptions are systems such as TFX (Baylor et al., 2017), Overton (Ré et al., 2020), or our own Ease.ML (Aguilar Melgar et al., 2021).[2] These systems are typically much more complex than just a single training or AutoML instance. The architecture of these systems is modular and consists of many different components responsible for very specific parts in an ML lifecycle. At the core of this thesis, we focus on a subset of such tasks and take a purely data-centric view by asking ***How can we use our data during the entire ML lifecycle in an efficient, yet principled and theoretically founded manner?***

Clearly, this broad question can not be addressed in a single doctoral thesis alone, but is likely to remain a central question for the foreseeable future. Focusing on how to *use* a fixed dataset to support data scientists (i.e., ignoring the question on how to *improve* the datasets for ML purposes), we formulate the following three high-level questions targeting first-of-their-kind ML lifecycle specific tasks:

> *Q1: What is the best accuracy any ML model can achieve on a given task, represented by a training and test dataset?*

Answering this question reliably helps our users to check whether the data quality meets the requirements for training a high-accuracy ML model. If not, users can refine their task, or change the datasets (e.g., by cleaning labels or gathering more data) before proceeding to the model development phase (e.g., by starting costly AutoML runs). Drawing an analogy to classical software engineering, the answer to this question enables our users to perform a *feasibility study* of the ML task at hand (Van Vliet et al., 2008).

[2] We contributed to two components inside the Ease.ML system with `Ease.ML/Snoopy` and `Ease.ML/CI`.

Q2: Using the training dataset, which pre-trained model(s) should be taken to achieve the best accuracy?

Code-reuse (Frakes and Kang, 2005) and software factories (Greenfield and Short, 2003) are well-established concepts that allow experienced software engineers to quickly get started with their project. In ML, using large pre-trained models available online, and especially doing transfer learning, represents a cost-efficient way for users with potentially small datasets to quickly obtain high-accuracy neural networks. The ever-growing number of available, mostly large models are typically not part of today's AutoML system's search space. The reason lies in the computational burden of evaluating even a small subset of these models, and depending on user's dataset and the pre-filtered models, different model search or selection strategies need to be applied. Understanding and efficiently supporting various model search strategies is therefore crucial for users applying transfer learning.

Q3: How large does the test dataset need to be to provide worst-case guarantees for repetitively evaluating a trained model?

Continuous integration is a key technique in classical DevOps (Ebert et al., 2016). It consists of two stages: (1) reliably test the new artifacts, (2) seamlessly integrate them into production. In order to prevent some catastrophic failures when continuously integrating a new ML model, or pushing a replacement for an existing model into production, one needs to take the inherent randomness of ML into account. Furthermore, users typically do not want to provide a new test set after every model evaluation, yet still want to be protected against overfitting.

1.1 Contributions

We provide three major contributions that address the previously raised questions (Q1 - Q3).

(C1) We present `Ease.ML/Snoopy`, a system to perform a systematic feasibility study for ML development. By feeding a dataset and target accuracy to the system, users get a binary signal, stating whether the target accuracy is feasible or not. Being aware of failures, both false positives and false negatives, the system returns additional guidance in the form of convergence curves, an extrapolation number, and an accuracy gap, to support user's decision whether to trust `Ease.ML/Snoopy`'s signal or not. At the core, `Ease.ML/Snoopy` uses a powerful 1NN-based

Bayes error rate (BER) estimator, for which we further analyze the theoretical properties such as convergence rate. The choice of this estimator stems from a large-scale study of existing estimators on real-world data, which lead to the definition of `FeeBee`, the first principled framework for analyzing and comparing BER estimators on any modern real-world dataset with unknown probability distribution.

(C2) When carefully analysing various pre-trained model search strategies, we realize that no single strategy outperforms all others for any combination of downstream task and subset of pre-trained models . Guided by these insights, we propose novel hybrid model search strategies combining prominent task-agnostic and task-aware approaches. To further efficiently support more complex and generic search queries, we propose `SHiFT`, the first downstream data aware, flexible model search engine for transfer learning. With targeted optimizations and caching mechanisms, we enable scalability to increasing number of pre-trained models, as well as efficient support for incremental query executions.

(C3) Finally, with `Ease.ML/CI`, we present the first continuous integration engine for ML models with strong statistical guarantees. `Ease.ML/CI` allows our users to re-use the same test set for a fixed number of evaluations. A custom declarative scripting language enables users to state a rich class of test conditions, which we evaluate with rigorous probabilistic guarantees. Based on various optimization techniques, we further reduce the number of samples and labels required for very specific test conditions.

1.2 Organization of the Thesis

This thesis is organized as follows: We start by giving a short ML primer and background on ML development and operations in Chapter 2. We then proceed by describing `Ease.ML/Snoopy` and the theory used to build this feasibility study for ML system in Chapter 3. `Ease.ML/Snoopy` leverages pre-trained models available online as powerful feature transformations, while the following Chapter 4 compares different model search strategies for transfer learning and presents `SHiFT`, a generic yet efficient model search engine for these pre-trained models. Chapter 5 focuses on `Ease.ML/CI`, our continuous integration engine with strong theoretical guarantees and a reduced number of samples required. We conclude this thesis in Chapter 6 summarizing the contributions and limitations of this thesis, along with outlining potential future work.

1.3 Author's Publications

This dissertation is largely based on the following publications presented in the order of appearance of this thesis (* equal contributions):

- **Cedric Renggli**, Luka Rimanic, Nezihe Merve Gürel, Bojan Karlaš, Wentao Wu and Ce Zhang. "A Data Quality-Driven View of MLOps". *IEEE Data Engineering Bulletin*. 2021.
- **Cedric Renggli***, Luka Rimanic*, Nora Hollenstein and Ce Zhang. "Evaluating Bayes Error Estimators on Read-World Datasets with FeeBee". *Advances in Neural Information Processing Systems (Datasets and Benchmarks)*. 2021.
- Luka Rimanic*, **Cedric Renggli***, Bo Li and Ce Zhang. "On Convergence of Nearest Neighbor Classifiers over Feature Transformations". *Advances in Neural Information Processing Systems*. 2020.
- **Cedric Renggli***, Luka Rimanic*, Luka Kolar*, Wentao Wu and Ce Zhang. "Ease.ML/Snoopy: Towards Automatic Feasibility Studies for ML via Quantitative Understanding of Data Quality for ML". *Under Review*. 2022.
- **Cedric Renggli**, Andre Susano Pinto, Luka Rimanic, Joan Puigcerver, Carlos Riquelme, Ce Zhang and Mario Lucic. "Which Model to Transfer? Finding the Needle in the Growing Haystack" *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022.
- **Cedric Renggli**, Xiaozhe Yao, Luka Kolar, Luka Rimanic, Ana Klimovic and Ce Zhang. "SHiFT: An Efficient, Flexible Search Engine for Transfer Learning". *Under Review*. 2022.
- **Cedric Renggli**, Bojan Karlaš, Bolin Ding, Feng Liu, Kevin Schawinski, Wentao Wu and Ce Zhang."Continuous Integration of Machine Learning Models with Ease.ML/CI: Towards a Rigorous Yet Practical Treatment". *Machine Learning and Systems*. 2019.

Further published works, which are outside the scope of this thesis in chronological order:

- John T. Halloran, ..., **Cedric Renggli**, ..., Lukas Käll and William Stafford Noble. "Speeding up Percolator". *Journal of Proteome Research*. 2019.
- Chen Yu, Hanlin Tang, **Cedric Renggli**, Simon Kassing, Ankit Singla, Dan Alistarh, Ji Liu, and Ce Zhang. "Distributed Learning over Unreliable Networks". *International Conference on Machine Learning*. 2019.
- Bojan Karlaš, Matteo Interlandi, **Cedric Renggli**, ..., Andy Xu and Markus Weimer. "Building Continuous Integration Services for Machine Learning". *Special Interest Group on Knowledge Discovery and Data Mining*. 2020.

- Maurice Weber, **Cedric Renggli**, Helmut Grabner and Ce Zhang. "Observer Dependent Lossy Image Compression". *German Conference on Pattern Recognition*. 2020.
- Nora Hollenstein, **Cedric Renggli**, Benjamin Glaus, Maria Barrett, Marius Troendle, Nicolas Langer and Ce Zhang. "Decoding EEG Brain Activity for Multi-Modal Natural Language Processing". *Frontiers in Human Neuroscience Vol. 15*. 2021.
- Aguilar Leonel, ..., **Cedric Renggli**, ..., Wentao Wu and Ce Zhang. "Ease.ML: A Lifecycle Management System for MLDev and MLOps". *Conference on Innovative Data Systems Research*. 2021.
- Joan Puigcerver, ..., **Cedric Renggli**, ..., Daniel Keysers and Neil Houlsby. "Scalable Transfer Learning with Expert Models". *International Conference on Learning Representations*. 2021.
- Lijie Xu, ..., **Cedric Renggli**, ..., Wentao Wu and Ce Zhang. "In-Database Machine Learning with CorgiPile: Stochastic Gradient Descent without Full Data Shuffle". *ACM Special Interest Group in Management Of Data*. 2022.
- Thórhildur Thorleiksdóttir, **Cedric Renggli**, Nora Hollenstein and Ce Zhang. "Dynamic Human Evaluation for Relative Model Comparisons". *Language Resources and Evaluation Conference*. 2022.

Chapter 2

Preliminaries

We start this thesis by outlining some machine learning and MLOps-related preliminaries used throughout all chapters.

2.1 Machine Learning Primer

The main goal of this primer on ML is to highlight the strong dependency between the data samples used to train or validate an ML model and its assumed underlying probability distribution. In this thesis we restrict ourselves to supervised learning in which, given a *feature space* $\mathcal{X}$ and a *label space* $\mathcal{Y}$, a user is given access to a dataset with n samples $\mathcal{D} := \{(x_i, y_i)\}_{i\in[n]}$, where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. Usually $\mathcal{X} \subset \mathbb{R}^d$, in which case a sample is simply a d-dimensional vector, whereas $\mathcal{Y}$ depends on the task at hand. For a regression task one usually takes $\mathcal{Y} = \mathbb{R}$, whilst for a classification task on C classes one usually assumes $\mathcal{Y} = \{1, 2, \ldots, C\}$. We restrict ourselves to classification problems.

Supervised ML aims at *learning* a map $h : \mathcal{X} \to \mathcal{Y}$ that generalizes to unseen samples based on the provided labeled dataset $\mathcal{D}$. A common assumption used to learn the mapping is that all data points in $\mathcal{D}$ are sampled identically and independently (i.i.d.) from an unknown distribution $p(X, Y)$, where X, Y are random variables taking values in $\mathcal{X}$ and $\mathcal{Y}$, respectively. For a single realization (x, y), we abbreviate $p(x, y) = p(X = x, Y = y)$.

The goal is to choose $h(\cdot) \in \mathcal{H}$, where $\mathcal{H}$ represents the hypothesis space that minimizes the expected risk with respect to the underlying probability distribution (Shalev-Shwartz and Ben-

David, 2014). In other words, one wants to construct h^* such that

$$h^* = \arg\min_{h \in \mathcal{H}} \mathbb{E}_{X,Y} \left(L(h(x), y) \right) = \arg\min_{h \in \mathcal{H}} \int_{\mathcal{X}} \int_{\mathcal{Y}} L(h(x), y) p(x, y) \, dy \, dx, \tag{2.1.1}$$

with $L(\hat{y}, y)$ being a loss function that penalizes wrongly predicted labels $\hat{y}$. For example, $L(\hat{y}, y) = \mathbf{1}(\hat{y} = y)$ represents the 0-1 loss, commonly chosen for classification problems, whereas the mean squared error (MSE) typically used for regression problems is given by $L(\hat{y}, y) = (\hat{y} - y)^2$. Finding the optimal mapping h^* is not feasible in practice: (1) the underlying probability $p(X, Y)$ is typically unknown and it can only be approximated using a finite number of samples, (2) even if the distribution were known, calculating the integral is intractable for many possible choices of $p(X, Y)$. Therefore, in practice one performs an empirical risk minimization (ERM) by solving $\hat{h} = \arg\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} L(h(x_i), y_i)$. Despite the fact that the model is learned using a finite number of data samples, the ultimate goal is to learn a model which generalizes to any sample originating from the underlying probability distribution, by approximating its posterior $p(Y|X)$. Using $\hat{h}$ to approximate h^* can run into what-is-known as "overfitting" to the training set $\mathcal{D}$, which reduces the generalization property of the mapping. However, advances in statistical learning theory managed to considerably lower the expected risk for many real-world applications while avoiding overfitting (Friedman et al., 2001; Shalev-Shwartz and Ben-David, 2014; Vapnik and Chervonenkis, 2015). Altogether, all data management aspects for ML applications should not only be treated with respect to the dataset $\mathcal{D}$ or individual data points therein, but also *with respect to the underlying probability distribution the dataset $\mathcal{D}$ is sampled from*.

2.1.1 Validation vs. Test

Standard ML cookbooks suggest that the data should be represented by three disjoint sets to *train*, *validate*, and *test*. The validation set accuracy is typically used to choose the best possible set of hyper-parameters used by the model trained on the training set. The final accuracy and generalization properties are then evaluated on the test set. Following this, we use the term *validation* for evaluating models in the pre-training phase, and the term *testing* for evaluating models in the post-training phase.

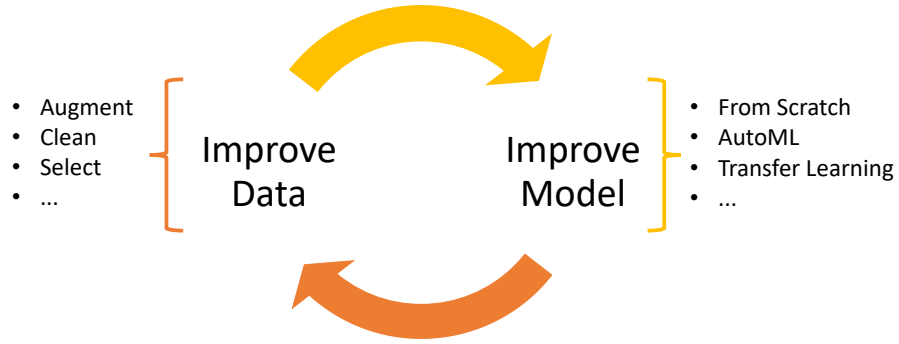


Figure 2.1: (Left) Data-centric approach, where the model architecture and training procedure is fixed and the data is improved. **(Right)** Model-centric approach where the data is fixed and higher accuracies are achieved using different training approaches.

2.2 Machine Learning Development

ML research over the last decade focused mainly on achieving better state-of-the-art accuracy on well-established, large, and noise-free (i.e., clean) benchmark datasets such as ImageNet (Deng et al., 2009). By following a *model-centric* approach, researcher fix a dataset and change properties of the model or the training procedure to achieve higher accuracies. There is a flurry of techniques on how to model the learned architecture or how to train it (LeCun et al., 2015). Figure 2.1 illustrates the interplay between this model-centric development and a data-centric one, which we describe later in this chapter. For model-centric approaches, we focus on three prominent methods illustrated on the right of Figure 2.1.

Training From Scratch This classical approach is mostly described in standard textbooks. The ML engineer hereby defines the architecture of the model (e.g., number of models and their type) as well as the optimization algorithm used to train (e.g., stochastic gradient descent) along with its hyper-parameters (e.g., mini-batch size, regularizer, or learning rate). The parameters of the model (e.g., weight of the network) are then randomly initialized and optimized from there.

AutoML AutoML systems aim at reducing the large space of design choices for both model architecture and optimization parameters by automatically searching through it. By defining a computational budget and some constraints on the search space, AutoML systems automatically explore various configurations. For every fixed configuration, training is typically started from randomly initialized weights.

Transfer Learning Transfer learning is a technique benefiting from previously trained models. The architecture and learned weights are initially copied and only slightly modified. The initialization is only random for new or changes parts of the architecture. The design space therefore restricted the changes of the model architecture, as well as the optimization algorithm with its hyper-parameters. In theory, many different pre-trained models could be fused into existing AutoML systems. This is rarely the case, given that AutoML systems are designed for small and relatively fast architectures, and they quickly become too slow when automatically searching through large neural network configurations. Transfer learning is the technique we focus on in Chapter 4, where we give a more detailed background.

2.2.1 Data-Centric AI

In reality, a wide audience trying to unleash the power of ML for their domain specific applications have much different data characteristics compared to the benchmark datasets mentioned earlier. Data is typically sparse, noisy, and possibly inconsistent. Iterating on and improving the data, instead of the model, is nowadays known as *data-centric* artificial intelligence (AI). The recipe for using ML in a most effective manner is probably neither purely model-, nor data-centric, but rather an interplay between the both, as illustrated in Figure 2.1. Data-centric AI does not only encompass how to *improve* the data artifacts for downstream ML, but also includes how to *use* the data in an efficient and theoretically understood manner (i.e., the focus of this thesis).

We next summarize some key concepts of data-centric AI:

Data Collection and Labeling At the core and beginning of every ML project, data from possibly multiple sources must be gathered. The different splits (train and test) need to be labeled, often a tedious human effort. There are some techniques to increase the amount of samples (e.g., augmentation (Li et al., 2021)) or automatically label samples (e.g., weak supervision (Ratner et al., 2017)), but given that they often add a substantial amount of noise, they should only be applied for the training set.

Data Transformations Deep learning aims at removing the manual effort of performing feature extraction. Nonetheless, a neural network still requires specifically encoded input types. Therefore, features like categorical ones need to be pre-processed, which lead to entire pipelines transforming raw data to machine learning ready formats (Xin et al., 2021).

Data Cleaning There are various types of error, for which solutions are needed to detect and fix (repair) them: (1) missing data, (2) outliers, (3) duplicates, (4) inconsistencies, and (5) mislabels (Ilyas and Chu, 2019). All these errors are sources of noise that change the distributional properties. Some of these errors are easily detectable by building and evaluating a data schema (Polyzotis et al., 2019) or "unit tests" for data (Schelter et al., 2018b), while others use more advanced concepts from database theory (Karlaš et al., 2020b).

Data Selection Restricting the dataset to be a subset of the available examples can help to speed up the training process (Coleman et al., 2019). Furthermore, finding hand-picked informative examples can help to explain or debug a trained model. Two prominent techniques enabling this are known as influence functions (Koh and Liang, 2017) and Shapley values (Ghorbani and Zou, 2019).

Data Validation Finally, for a fixed dataset, reliably estimating its quality before training or evaluating an ML model can prevent data scientists from jumping into the model development phase when doomed to fail given some inherent data quality limitations. This action is the focus of Chapter 3.

2.3 Machine Learning Operations

Machine learning operations, not to confuse with MLOps which consists of both the development and operational phase, is the final stage in an ML lifecycle after having successfully trained a new or better model. In operations, the goal is to put the model safely into production and ensure its robustness there. While many technical challenges are similar to classical software artifacts (e.g., fault tolerance or load balancing), there are ML specific aspects, mainly data related, which need to be dealt with. We next describe these differences for various stages of the ML operations phase.

Model Evaluation New models have to be thoroughly evaluated before being put into production or replacing an existing artifact in there. When it comes to testing ML models safely, ML theory suggests to use a fixed test set only once, which is rarely doable in practice. Furthermore, applying standard techniques such as A/B testing tends to require an extremely large number of labeled samples (Nandy et al., 2020). We address this issue in Chapter 5 of this thesis.

Model Deployment Typically, there is not only a single model being developed or active in production. Various online repositories such as Hugging Face, PyTorch Hub, or TensorFlow Hub facilitate sharing and reusing pre-trained models. We make use of these repositories in Chapters 3 and 4. Other systems such as ModelDB (Vartak et al., 2016) or MLFlow (Zaharia et al., 2018) extend the repository functionality by further enabling version of models, tracking of experiments, and efficient deployment.

Model Monitoring It is well-known that the accuracy of active models in production typically diminishes over time. The main reason for this lies in the distribution shift between the new real-time test data and the data used to train the model originally. The most prominent remedy to this problem still lies in periodically (sometimes on a daily or even hourly basis) re-training models using fresh training data. This is a very costly undertaking which can be prevented by having access to drift detectors (also referred to as anomaly or outlier detectors) (Ginart et al., 2022).

Model Adaptation Finally, a model in production might have to be slightly adapted without having to go through the entire ML development process. Small changes can be made via continual learning (e.g., to adapt to small variations in the distribution) (Chen and Liu, 2018), or, when required by regulations to forget a specific data point, techniques from model unlearning apply (Schelter, 2020). To ensure stability in production, modified models should go through the model evaluation stage before safely be put into production.

Chapter 3

Ease.ML/Snoopy: Feasibility Study for ML

Modern software development is typically guided by software engineering principles that have been developed and refined for decades (Van Vliet et al., 2008). Even though such principles are yet to come to full fruition regarding the development of ML applications, in recent years we have witnessed a surge of work focusing on ML usability through supporting efficient ML systems (Li et al., 2014; Meng et al., 2016; Zaharia et al., 2018), enhancing developer's productivity (Bergstra et al., 2013; Baylor et al., 2017; Li et al., 2017), and supporting the ML application development process itself (Vartak et al., 2016; Kraska, 2018; Polyzotis et al., 2019; Nakandala et al., 2020).

Calls for a Feasibility Study for ML In this chapter, we focus on one specific "failure mode" that we frequently witness whilst working with a range of domain experts, which we call "*unrealistic expectations*." Unlike classical software artifacts, the quality of ML models (e.g., its accuracy) is often a reflection of the *data quality* used to train or test the model. We regularly

The content of this chapter are largely based on the following two publications lying the theoretical foundation: (1) Rimanic et al. (2020). On Convergence of Nearest Neighbor Classifiers over Feature Transformations. *NeurIPS*, and (2) Renggli et al. (2021b). Evaluating Bayes Error Estimators on Real-World Datasets with FeeBee. *NeurIPS Datasets and Benchmarks*. A paper describing the system is currently under review and can be accessed via: (4) Renggli et al. (2022b) Ease.ML/Snoopy: Towards Automatic Feasibility Studies for ML via Quantitative Understanding of "Data Quality for ML". *Under Review*. A running system has been demonstrated in (5) Renggli et al. (2020). Ease.ML/Snoopy in Action: Towards Automatic Feasibility Analysis for Machine Learning Application Development. *VLDB Demo*.

see developers that work on challenging tasks with a dataset that is *too* noisy to meet the unrealistically high expectations on the accuracy that can be achieved with ML — such a project is predestined to fail. Ideally, problems of this type should be caught *before* a user commits significant amount of resources to train or tune ML models.

In practice, if this were done by a *human ML consultant*, she would first analyze the representative dataset for the defined task and assess the *feasibility* of the target accuracy — if the target is not achievable, one can then explore alternative options by refining the dataset, the acquisition process, or investigating different task definitions. Borrowing the term from classic software engineering, we believe that such a *feasibility study* step is crucial to the usability of future ML systems for application developers. In this chapter, we ask: *Can we provide some systematic and theoretically understood guidance for this feasibility study process?*

Quantitative Understanding of "Data Quality for ML" Data quality, along with its cleaning, integration, and acquisition, is a core data management problem that has been intensively studied in the last few decades (Cong et al., 2007; Fan, 2015; Schelter et al., 2018b; Ilyas and Chu, 2019). Agnostic to ML workloads, the data management community has been conducting a flurry of work aimed at understanding and quantifying data quality issues (Wang and Strong, 1996; Strong et al., 1997; Batini et al., 2009). In addition to these fundamental results, the presence of an ML training procedure as a downstream task over data provides both challenges and opportunities. On the one hand, systematically mapping these challenges to ML model quality issues is largely missing (with the prominent exceptions of (Krishnan et al., 2016; Ghorbani and Zou, 2019) together with some of our own previous efforts (Li et al., 2019; Jia et al., 2019a; Karlaš et al., 2020b)). On the other hand, the ML training procedure provides a quantitative metric to measure precisely the utility of data, or its quality. In this chapter, we take one of the early steps in this direction and ask: "*Can we quantitatively map the quality requirements of a downstream ML task to the requirements of the data quality of the upstream dataset?*"

The Scope and Targeted Use Case As one of the first attempts towards understanding these fundamental problems, this chapter by no means provides a complete solution. Instead, we have a very specific application scenario in mind for which we develop a deep understanding both theoretically and empirically. Specifically, we focus on the case in which a user has access to a dataset $\mathcal{D}$, large enough to be representative for the underlying task. The user is facing the following question: *Is my current data artefact $\mathcal{D}$ clean/good enough for some ML models to reach a target accuracy α_{target}?* If the answer to this question is "Yes", the user can start expensive

AutoML runs and hopefully can find a model that can reach α_{target}; otherwise, it would be better for the user to improve the *quality* (via data cleaning for example) of the data artefact $\mathcal{D}$ before starting an AutoML run which is "doomed to disappoint". We call the process of answering this question a "*feasibility study*". Our main goal is to derive a strategy for the feasibility study that is **(i)** *informative and theoretically justified*, **(ii)** *inexpensive*, and **(iii)** *scalable*.

Such a process can be useful in many scenarios. In this chapter we develop a fundamental building block of the feasibility study and evaluate it focusing on one specific use case as follows — the dataset $\mathcal{D}$ is noisy in its labels, probably caused by (1) the inherent noise of the data collection process such as *crowd sourcing* (Gadiraju et al., 2015; Sun et al., 2020), or (2) *bugs* in the data preparation pipeline (which we actually see quite often in practice). The user has a target accuracy α_{target} and can spend time and money on two possible operations: (1) manually clean up some labels in the dataset, or (2) find and engineer suitable ML models, manually or automatically.

Challenges of the Strawman There are multiple strawman strategies, each of which has its own challenge. A natural, rather trivial approach is to run a *cheap proxy model*, e.g., logistic regression, to get an accuracy α_{proxy}, and use it to produce an estimator $\alpha_{\text{est}} = c \cdot \alpha_{\text{proxy}}$, say, for some $c \in [1, 1/\alpha_{\text{proxy}}]$. The challenge of this approach is to pick a universally good constant c, which depends not only on the data but also on the cheap proxy model. It is important, but often challenging, to provide a principled and theoretically justified way of adjusting the gap between α_{proxy} and α_{est}. Another alternative approach would be to simply fire up an AutoML run that systematically looks at various configurations of feature extractors, ML models, and potentially neural architectures. Given enough time and resources, this could converge closely to the best possible accuracy that one can achieve on a given dataset; nevertheless, this can be very expensive and time consuming, thus might not be suitable for a quick feasibility study.

Feasibility Study: Theory vs. Practice The main challenges of the strawman approaches motivate us to look at this problem in a more principled way. From a theoretical perspective, our view on feasibility study is not new, rather it connects to a decades-old ML concept known as the Bayes error rate (BER) (Cover and Hart, 1967), *the "irreducible error" of a given task* corresponding to the error rate of the *Bayes optimal classifier*. In fact, all factors leading to an increase of the BER can be mapped to classical data quality dimensions, e.g., "label noise" to "accuracy", or "missing features " to "completeness" (Renggli et al., 2021a).

Despite decades of research, the problem of performing an efficient feasibility study if far more complex than simply employing an existing BER estimator and report the value to the user. The next three challenges, which we tackle chronologically in this chapter, motivate the complexity of the endeavour.

Challenge 1: BER Estimators on Real-World Datasets. Over the last 50 years, coming up with practical Bayes error estimators in the finite-data regime has been a never ending pursuit of the ML community: from Fukunaga's early effort back in 1975 (Fukunaga and Hostetler, 1975) to Sekeh et al.'s recent effort (Sekeh et al., 2020) and a diverse collection of other Bayes error estimators (Berisha et al., 2015; Buturovic and Markovic, 1992; Devijver, 1985; Fukunaga and Hummels, 1987; Pham-Gia et al., 2007).

Despite the large number of estimators, there exists a technical gap: *BER estimators have only been evaluated and compared on simple synthetic datasets for which the true BER can be calculated*. This results in a lack of understanding of the performance and practicality of BER estimators in real-world scenarios. First, all these synthetic datasets are constructed with simple data generative processes, which can be very different from real-world datasets that are diverse in their modalities (e.g., images, text, tabular data) and data distributions. Second, many BER estimators consist of a hyper-parameter tuning mechanism that relies on the true BERs of synthetic datasets (Cover and Hart, 1967; Fukunaga and Hummels, 1987; Snapp and Xu, 1996), which are in practice not available for real-world datasets. Third, real-world datasets can often take advantage of pre-trained feature transformations which can lead to improving convergence rates. Such a behavior is also hard to explore on synthetic datasets.

Challenge 2: Impact of (Pre-trained) Feature Transformations. At the same time, most, if not all, theoretical results for BER estimators are derived under assumptions that the algorithms are directly applied on the raw data, resulting in an *emerging gap between the theoretical understanding and practical application.* Identifying estimators based on the kNN classifier accuracy as the most suitable ones for our purpose, we are interested in the *convergence behavior* of kNN over transformations: *Given a transformation f and a set of n training examples, can we describe the (expected) error of kNN over f as a function of n and some properties of the transformation f?* In other words, given a fixed n, *what are the properties of f that strongly correlate with the kNN error over f?* First, as many existing results on the convergence have a factor of $(k/n)^{1/D}$, where D is the dimension of the space, a natural hypothesis could be: *given a fixed n, transformations resulting in lower D have lower kNN error.* While this is plausible, it leaves many empirical phenomenons unexplained. For example, Figure 3.1 (left) illustrates kNN errors

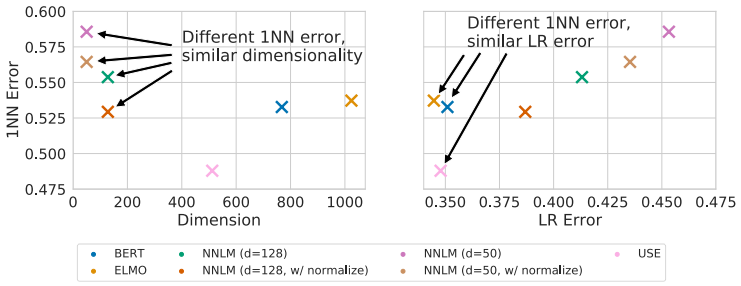


Figure 3.1: Challenges of examining the behavior of the 1NN classifier on top of feature transformations on YELP dataset. **(Left)** 1NN vs. dimension, **(Right)** 1NN vs. LR Error.

over feature transformations with different dimensions, showing a real-world dataset in which transformations of the same dimension have drastically different kNN accuracies. Another approach is to consider the *accuracy* of some other classifier. For example, training a logistic regression (LR) model on f and using it directly to form a hypothesis: *given a fixed n, transformations that lead to higher logistic regression accuracy have lower kNN error.* This is a much stronger hypothesis and can explain many scenarios that dimensionality alone cannot. However, Figure 3.1 (right) shows that this still leaves some important cases unexplained, providing examples in which multiple transformations achieve similar logistic regression accuracies, whilst having noticeably different kNN accuracies.

Challenge 3: Practical System Design. Finally, based on the insights of deploying BER estimators on real-world datasets along with a deeper theoretical understanding of their convergence behaviour, there are still two major non-trivial questions. Firstly, *how should estimations of the BER, and especially lower bound estimations, be used for the purpose of a systematic feasibility study for ML*? Secondly, *how can we build a scalable system using a BER estimator to remain practical for today's large-scale real-world datasets*?

3.1 Contributions

For each of the challenges outlined before we make multiple contributions.

(C1) To evaluate and compare BER estimators on real-world datasets, we make multiple key contributions. Firstly, we propose `FeeBee`, the first principled and practical framework for comparing different BER estimators by reasoning about the *evolution* of the true (unknown) BER rather than evaluating the BER as a single value. Secondly, we open-source and deploy `FeeBee`

on 6 well-established, real-world, computer vision and text classification benchmark datasets, on which we systematically evaluate a range of 7 existing BER estimators. The framework can easily be extended with new BER estimators.[1] Finally, we perform further studies in order to understand the behavior of certain BER estimators, as well as studying the potential of simple (scaled) classifiers to be used as BER estimators.

(C2) To understand the convergence behaviour of the kNN classifier on top of different (pre-trained) transformations we first define two key factors any transformation f relies on: (1) *safety – how well can we recover the posterior in the original space from the feature space*, and (2) *smoothness – how hard it is to recover the posterior in the original space from the feature space?* We then answer these two questions by defining and examining *safety* as the decrease in the best possible accuracy of any classifier (i.e., the Bayes error) in the feature space when compared to the original features, whilst we use the geometrical nature of the transformed features to examine *smoothness*. We show that with this formulation we can explain the relative performance of many transformations used in the kNN setting.

We highlight the usefulness and validate our novel theoretical understanding by conducting a thorough experimental evaluation ranging over 6 real-world datasets from two popular machine learning modalities, and 30 different feature transformations.

(C3) We present `Ease.ML/Snoopy`— a fast, practical and systematic feasibility study system for machine learning. We model the problem of feasibility study as estimating a *lower bound* of the BER. Users provide `Ease.ML/Snoopy` with a dataset representative for their ML task along with a target accuracy. The system then outputs a binary signal assessing whether the target accuracy is realistic or not, by taking the minimum over a collection of different BER estimators built on top of a collection of pre-trained feature transformations. Being aware of failures (false-positives and false-negatives) in the binary output of our system, which we carefully outline and explain in this chapter, we support the users in deciding on whether to "trust" the output of our system by providing additional numerical and visual aids. We provide a theoretical analysis on the regimes under which this aggregation function is justified. We then optimize `Ease.ML/Snoopy` with the *successive-halving* algorithm (Jamieson and Talwalkar, 2016), a part of the textbook Hyperband algorithm (Li et al., 2017), to balance the resources spent on different estimators. Finally, we perform a thorough experimental evaluation of `Ease.ML/Snoopy` against the baselines that use cheap and expensive proxy models. We show that `Ease.ML/Snoopy` consistently outperforms

[1] Accessible via: `https://github.com/DS3Lab/feebee`.

the cheap, and matches the expensive strategy in terms of predictive performance for synthetic and natural label noise, whilst being computational much more efficient than both approaches.

3.2 Background and Related Work

We start by summarizing the background and related work required to build `Ease.ML/Snoopy`. We give a short overview over the technical terms and the notation used throughout this chapter. Let $\mathcal{X}$ be the feature space and $\mathcal{Y}$ be the label space, with $C = |\mathcal{Y}|$. Let $X \in \mathcal{X}, Y \in \mathcal{Y}$ be random variables. Let $p(X, Y)$ be their joint distribution, often simplified by $p(x, y) = p(X = x, Y = y)$. We define $\eta_y(x) = p(y|x)$ when $C > 2$, and $\eta(x) = p(1|x)$ when $C = 2$, assuming $\mathcal{Y} = \{0, 1\}$.

3.2.1 kNN Classifier

For a training set $\mathcal{D}_n := \{(x_i, y_i)\}_{i\in[n]}$, and a new instance x, let $(x_{\pi(1)}, \ldots, x_{\pi(n)})$ be a reordering of the training instances with respect to their distance from x, through some appropriate metric (e.g., euclidean distance). In that setting, the *kNN classifier* $h_{n,k}$ and its *n-sample error rate* are defined by

$$h_{n,k}(x) = \arg\max_{y\in\mathcal{Y}} \sum_{i=1}^{k} \mathbf{1}_{\{y_{\pi(i)}=y\}}, \qquad (R_X)_{n,k} = \mathbb{E}_{X,Y}\mathbf{1}_{\{h_{n,k}(X)\neq Y\}},$$

respectively. The *infinite-sample error rate* of kNN is given by $(R_X)_{\infty,k} = \lim_{n\to\infty}(R_X)_{n,k}$.

3.2.2 Bayer Error Rate

The *Bayes optimal classifier* is the classifier that achieves the lowest error rate among all possible classifiers from $\mathcal{X}$ to $\mathcal{Y}$, with respect to $\mathcal{D}$. Its error rate is called the *Bayes error rate (BER)* and we denote it, depending on the context, by $R^*_{\mathcal{D}}$ or $R^*_{X,Y}$, often abbreviated to R^*_X when Y is clear from the context. It can be expressed as

$$R^*_X = \underbrace{\mathbb{E}_X[1 - \overbrace{\max_{y\in\mathcal{Y}} \underbrace{\eta_y(x)}_{(1)}}^{(2)}]}_{(3)}. \tag{3.2.1}$$

The BER often plays a central role in the analysis of kNN classifiers (Chaudhuri and Dasgupta, 2014; Döring et al., 2017; Györfi et al., 2002).

Table 3.1: Overview of BER estimators and their hyper-parameters.

Estimator	Parameter	Values
GHP	*None*	
DE-kNN,1NN-kNN	k	$[2, 100]$
Gaussian KDE	B	$\{0.0025, 0.05, 0.1, 0.25, 0.5\}$
1NN / kNN / kNN-LOO/ kNN-Extrapolate	k; dist	$[1, 10]$; $\{\text{cosine}, L_2\}$

3.2.3 BER Estimators

When examining BER estimators, we omit approaches which only work for binary classification tasks (e.g., Ishida et al. (2022)) given their limited application. We thus only consider methods which are capable of estimating the BER for multi-class classification problems ($C \geq 2$), and divide them into three categories, based on different parts of Equation 3.2.1: *density estimators* that estimate (1), *divergence estimators* that address (2), and *estimators* built around the *k-nearest neighbors algorithm*, which focus on (3). We distill each of these groups in detail.

(1) Density estimators Both methods in this group use the full test set with labels to estimate the class posterior, repeated once again for the same test set, but this time without the labels. This allows one to sample the feature space accordingly and get an estimate of the expectation over $\mathcal{X}$. **(DE-kNN)** This method estimates the per-class posterior η_y for all $y \in \mathcal{Y}$, by counting the fraction k_y/k of samples with that specific label amongst the k nearest neighbors, in expectation over the feature space (Fukunaga and Kessell, 1973). One uses the full test set only to estimate an upper bound by the leave-one-out (LOO) technique, and an optimistic lower bound by the re-substitution technique (Fukunaga and Hummels, 1987). **(KDE)** This method estimates the class prior by first taking a fraction of per-class samples in the full test set. Using a kernel density approach, the class likelihood is then estimated using all the samples per class separately. Finally, by using the Bayes formula, one can derive the posterior density per class, which is used as the lower and upper bound.

(2) Divergence estimator **(GHP)** This estimator uses the generalized Henze-Penrose divergence (Sekeh et al., 2020) between every pair of η_i and η_j, to get a provably valid estimator of the BER in the asymptotic regime. The estimation of the divergence can be further utilized to

get an upper/lower-bound estimate of the BER. Implementation-wise, using only a single set of samples, one first constructs the minimum spanning tree (MST) over the fully connected graph over all the samples, with edges being defined through Euclidean distances, and then uses the number of dichotomous edges to estimate the BER, noting that GHP and kNN-LOO (introduced below) have similar computational complexity.

(3) kNN classifier accuracy **(1NN-kNN)** The approach by Devijver (1985) aims at estimating the 1NN classifier accuracy by using the k-nearest-neighbor information. In order to get an unbiased estimator of the 1NN classifier accuracy, Devijver (1985) proposed using the formula $\frac{1}{k(k-1)} \sum_{y \in \mathcal{Y}} k_y(k - k_y)$ as the estimator, where k is the hyper-parameter of the method. This approach is very similar to *DE-kNN*, with the difference that we are not estimating η_y based on a test set, but directly the 1NN classifier accuracy. Samples are usually used twice through the resubstitution technique in order to get the 1NN classifier accuracy. **(kNN-Extrapolate)** One major caveat of bounding the BER by any kNN accuracy method lies in the fact that the bounds hold only in the asymptotic regime. As an attempt to surpass this limitation, Snapp and Xu (1996) extrapolate the convergence values of kNN for different number of training samples by assuming probability densities with uniformly bounded partial derivatives up through order N. However, this requires the number of samples to be exponential in the input dimension and, hence, is challenging to generalize it to representations of higher dimension on real-world datasets. **(1NN)** Inspired by Cover and Hart (1967), we define

$$\widehat{R}_{X,1\mathrm{NN}} := (R_X)_{n,1} \Big/ \left(1 + \sqrt{1 - \frac{C(R_X)_{n,1}}{C-1}}\right), \tag{3.2.2}$$

where $(R_X)_{n,k}$ is the validation error of a kNN classifier (with n training samples). The main motivation comes from the fact that in the asymptotic regime, i.e. for $n = \infty$, Cover and Hart (1967) proved that the RHS of Equation 3.2.2 serves as a lower bound of the BER. More precisely, Cover and Hart (1967) derived the following fundamental, and now well-known, relationship between the nearest neighbor algorithm and the BER (under very mild assumptions on the underlying probability distribution):

$$(R_X)_{\infty,1} \geq R_X^* \geq \frac{(R_X)_{\infty,1}}{1 + \sqrt{1 - \frac{C(R_X)_{\infty,1}}{C-1}}}. \tag{3.2.3}$$

(kNN) For $k > 1$ and $C > 2$ there is no known bound as strong as for $k = 1$. However, we can still use the same bound even for $k > 1$, even though it is less tight, with a further improvement by Devroye (1981) when $C = 2$, yielding

$$\widehat{R}_{X,\mathrm{kNN}} = \begin{cases} (R_X)_{n,k} / \left(1 + \sqrt{1 - \frac{C(R_X)_{n,k}}{C-1}}\right), & C > 2, k > 1, \\ (R_X)_{n,k} / (1 + \sqrt{2/k}), & C = 2, k = 2, \\ (R_X)_{n,k} / (1 + \sqrt{1/k}), & C = 2, k > 2. \end{cases} \tag{3.2.4}$$

(kNN-LOO) When one wants to omit splitting the dataset into test and train sets, the kNN classifier accuracy can be reported using a leave-one-out approach. In this chapter we simply use this estimator for a fair comparison with certain non-scalable methods, noting that this approach is typically not computationally feasible in practice.

3.2.4 Synthetic vs Non-Synthetic Regimes

In previous work, even though BER estimators were sometimes applied on real-world datasets (e.g., as a utility for feature reduction strategies, or for quantifying layer-to-layer change in convolutional neural networks, both by Sekeh et al. (2020)), their theoretical properties were tested only on synthetic datasets (Cover and Hart, 1967; Devijver, 1985; Fukunaga and Hummels, 1987; Fukunaga and Kessell, 1973; Sekeh et al., 2020; Snapp and Xu, 1996). Upon knowing the underlying probability distribution, the true BER can then either be computed directly (e.g., (Cover and Hart, 1967; Snapp and Xu, 1996)), or through a simulation, such as Monte Carlo method (Sekeh et al., 2020). These synthetic datasets often assume Gaussian distributions, over small dimensions (Fukunaga and Hummels, 1987; Fukunaga and Kessell, 1973; Snapp and Xu, 1996; Sekeh et al., 2020)). Since a BER estimator is usually constructed based on strong theoretical guarantees in the asymptotic regimes, for each such synthetic dataset one can usually find a set of hyper-parameters which make the estimator predict the true BER reasonably well. However, in order to be able to compare BER estimators, a useful framework needs to give them a chance to be wrong. Therefore, comparing them on synthetic datasets would not yield transferable insights towards real-world, non-synthetic datasets, since they perform well on synthetic datasets, whereas in practice, predicting the true BER is notoriously hard.

3.2.5 kNN Convergence

As one of the fundamental machine learning models, the kNN classifier enjoys a long history of theoretical understanding and analysis. The first convergence rates of a kNN classifier were established by Cover (1968), where convergence rates of 1NN in $\mathbb{R}$ under the assumption of uniformly bounded third derivatives of conditional class probabilities were given, further extended

to $\mathbb{R}^d$ (Fritz, 1975; Wagner, 1971). Distribution-dependant rates of convergence have been examined by Györfi (1981); Kulkarni and Posner (1995), where certain smoothness assumptions were imposed. These were further developed by Chaudhuri and Dasgupta (2014) through the notion of effective boundary, similar to the margin and strong density conditions (Audibert and Tsybakov, 2007; Döring et al., 2017; Gadat et al., 2016; Kohler and Krzyzak, 2007; Tsybakov, 2004). Finally, Györfi et al. (2002) provides convergence rates under the simplest assumptions, which perfectly suit our purposes. To the best of our knowledge, all previous works on the rates of convergence of kNN classifiers are derived on raw features.

It is no secret that kNN classifiers over raw features are "cursed", suffering from what is known as the *curse of dimensionality* (Cover and Hart, 1967; Snapp et al., 1991; Snapp and Xu, 1996). In practice, there have been many results that apply kNN classifiers over feature transformations. In the easiest form, such transformations can be as simple as a PCA transformation (Kacur et al., 2011; Panahi et al., 2011; Trstenjak et al., 2014). Recently, a popular choice is to apply kNN over pre-trained deep neural networks (Bahri et al., 2020; Jia et al., 2019b; Papernot and McDaniel, 2018; Wang et al., 2017). Other related works propose to optimize a neural feature extractor explicitly for a kNN classifier (Goldberger et al., 2005; Wu et al., 2018). Most of these results show significant improvements on accuracy, empirically, bringing the performance of a simple kNN classifier on a par with state-of-the-art models. However, there is an obvious lack of rigorous understanding of properties that a feature transformation needs to satisfy in order to achieve a good kNN performance.

3.3 Experimental Setup

We outline the experimental setup shared across all three parts (i.e., for the challenges C1 - C3 outlined earlier) of this chapter next.

Datasets We perform the evaluation on two data modalities which are ubiquotous in modern machine learning and are accompanied by strong state-of-the-art (SOTA) performances summarized in Table B.1. The first group consists of *visual classification tasks*, including MNIST, CIFAR10 and CIFAR100. The second group consists of standard *text classification tasks*, where we focus on IMDB, SST2 and YELP. The details are presented in Appendix B.1.1. We remark that for the visual classification tasks, the raw features are the pixel intensities, whereas for the text classification we apply the standard bag-of-words preprocessing, with and without term-

frequency/inverse-document-frequency weighting (Jones, 1972). The SOTA values for SST2 and YELP are provided on slightly different sizes of training sets.

Feature Transformations We run the experiments on a diverse set of feature transformations used in practice. The list of 30 transformations includes standard dimension reduction methods like the Principal Component Analysis (PCA) and Neighborhood Component Analysis (NCA), as well as using pre-trained feature extraction networks available in TensorFlow Hub and PyTorch Hub. A complete list of the used transformations per dataset is given in Appendix B.1.2.

3.4 Comparing Bayes Error Estimators on Real-World Data with FeeBee

In this section, we present FeeBee, which, to our best knowledge, is the first evaluation framework of BER estimators on realistic, non-synthetic datasets. The goal of FeeBee is to enable a systematic study of the performance and practicality of BER estimators in more realistic scenarios. Designing FeeBee is challenging — after all, *how can we compare BER estimators without knowing the true BER?* FeeBee's idea is to go beyond simply evaluating an estimator at a *single point* — instead, evaluate it on a series of points, for which we know the *relative relationship* among their BERs. At its core, FeeBee injects label noise to existing datasets, and using a simple but novel technical result, it estimates the corresponding BER on different noise levels to measure to what degree a BER estimator under/overestimates. The FeeBee framework allows us to systematically evaluate BER estimators. Especially, we focus on two main reasons that hamper the applications of BER estimators in real-world scenarios: **(i)** difficulty of choosing correct hyper-parameters and a feature transformation, and **(ii)** computational and data efficiency.

3.4.1 Evaluation Framework

Given a dataset $\mathcal{D}$ containing n i.i.d. samples from $p(X, Y)$, a BER estimator m provides us an estimation of the lower bound $\ell_{\mathcal{D},m}$ and the upper bound $u_{\mathcal{D},m}$ of the *unknown* BER $R^*_{\mathcal{D}}$. In the following, we also assume that we are aware of the state-of-the-art performance (SOTA) of applying machine learning on this dataset: $s_{\mathcal{D}} \geq R^*_{\mathcal{D}} \geq 0$. The goal of our evaluation framework is to come up with some metrics to measure the quality of the lower bound $\ell_{\mathcal{D},m}$ and the upper bound $u_{\mathcal{D},m}$:

$$L_{\mathcal{D}}(m) \in [0,1] \triangleq \texttt{Sub-optimality of LB estimator } \ell_{\mathcal{D},m}\texttt{; lower the better}$$
$$U_{\mathcal{D}}(m) \in [0,1] \triangleq \texttt{Sub-optimality of UB estimator } u_{\mathcal{D},m}\texttt{; lower the better}$$

What is a good evaluation framework? We consider two natural key requirements that we believe need to be satisfied when evaluating BER estimators. First, if the estimator predicts exactly the BER $R^*_{\mathcal{D}}$ for both the lower and the upper bound, then it should hold that $L_{\mathcal{D}}(m) = U_{\mathcal{D}}(m) = 0$. Second, any lower-bound estimate that satisfies $\ell_{\mathcal{D},m} > s_{\mathcal{D}}$ is clearly wrong, whereas any upper-bound estimate that satisfies $u_{\mathcal{D},m} > s_{\mathcal{D}}$ is clearly outperformed by $u = s_{\mathcal{D}}$, SOTA itself. In particular, a good framework should penalize these two cases.

Challenges of evaluating BER estimators at a single point The key challenge of evaluating BER estimators on real-world datasets is that the true BER $R^*_{\mathcal{D}}$ is unknown and we only have access to the SOTA $s_{\mathcal{D}}$. We first show that simply evaluating BER estimators using $s_{\mathcal{D}}$ is challenging in meeting these two natural requirements. The first baseline would use the SOTA as the proxy of $R^*_{\mathcal{D}}$:

$$L_{\mathcal{D}}(m) = |s_{\mathcal{D}} - \ell_{\mathcal{D},m}|, \quad U_{\mathcal{D}}(m) = |u_{\mathcal{D},m} - s_{\mathcal{D}}|.$$

This does not satisfy our first requirement — an "ideal" estimator that always outputs $R^*_{\mathcal{D}}$ would have non-zero sub-optimality as long as $s_{\mathcal{D}} \neq R^*_{\mathcal{D}}$.

We could construct another baseline inspired by the fact that $\ell_{\mathcal{D},m} = 0$ and $u_{\mathcal{D},m} = s_{\mathcal{D}}$ are true bounds and assign them zero sub-optimality:

$$L_{\mathcal{D}}(m) = |\ell_{\mathcal{D},m}|, \quad U_{\mathcal{D}}(m) = |u_{\mathcal{D},m} - s_{\mathcal{D}}|.$$

It satisfies that predicting $\ell_{\mathcal{D},m} = 0$ and $u_{\mathcal{D},m} = s_{\mathcal{D}}$ gives a perfect score. However, the true BER $R^*_{\mathcal{D}}$ has a non-zero score as soon as it differs from $s_{\mathcal{D}}$, again contradicting the first requirement.

(ii) One could attempt at penalizing intervals that are *certainly* sub-optimal, for example by

$$L_{\mathcal{D}}(m) = \mathbf{1}_{\{\ell_{\mathcal{D},m} > s_{\mathcal{D}}\}}, \quad U_{\mathcal{D}}(m) = \mathbf{1}_{\{u_{\mathcal{D},m} > s_{\mathcal{D}}\}}.$$

However, this is not distinguishable enough as both the estimator that predicts $\ell_{\mathcal{D},m} = u_{\mathcal{D},m} = 0$ and $\ell_{\mathcal{D},m} = u_{\mathcal{D},m} = s_{\mathcal{D}}$ have perfect score, whereas it is obvious that even though an estimator that always predicts 0 gives a valid lower bound, it is completely non-informative.

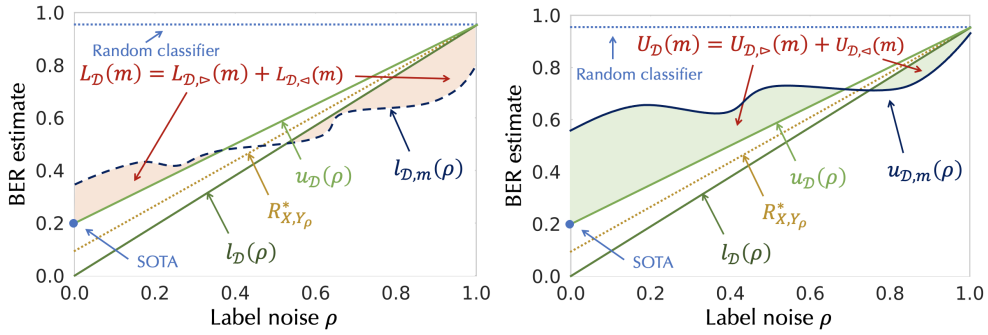


Figure 3.2: Evaluation Methodology for estimating the lower bound **(left)** and the upper bound **(right)**.

Evaluating through a series of points In order to overcome the above limitation, `FeeBee` injects different levels of label noise and measures how an estimator follows the evolution of the BER, as illustrated in Figure 3.2. For each sample, with probability $\rho \in [0, 1]$ we flip the label, in which case we choose a label uniformly at random. The following result quantifies the increase in the BER.

Lemma 3.4.1. *Let Y_ρ be a random variable defined on $\mathcal{Y}$ by setting $Y_\rho = Z \cdot U(\mathcal{Y}) + (1 - Z) \cdot Y$, where U is a uniform variable taking values in $\mathcal{Y}$, and Z is a Bernoulli variable with probability $0 \leq \rho \leq 1$, both independent of X and Y. Then $R^*_{X,Y_\rho} = R^*_{X,Y} + \rho(1 - 1/C - R^*_{X,Y})$.*

Proof: Let $p(X, Y_\rho)$ be the corresponding joint distribution on $\mathcal{X} \times \mathcal{Y}$ for the random variables X and Y_ρ, simplified as $p_\rho(x, y)$. Note that

$$p_\rho(y|x) = \underbrace{p_\rho(y|x, Z = 0)}_{p(y|x)} p(Z = 0) + \underbrace{p_\rho(y|x, Z = 1)}_{p(U=y)} p(Z = 1) = (1 - \rho)p(y|x) + \frac{\rho}{C}.$$

Thus,

$$\begin{aligned} R^*_{X,Y_\rho} &= \mathbb{E}_X[1 - \max_{y \in \mathcal{Y}} p_\rho(y|x)] = 1 - \mathbb{E}_X \max_{y \in \mathcal{Y}} [(1 - \rho)p(y|x) + \rho/C] \\ &= 1 - \rho/C - (1 - \rho)\mathbb{E}_X \max_{y \in \mathcal{Y}} p(y|x) = R^*_{X,Y} + \rho(1 - 1/C - R^*_{X,Y}). \qquad \square \end{aligned}$$

We remark that $Y_\rho \in \mathcal{Y}$, where ρ corresponds to the probability of randomly changing the original label to a random value in $\mathcal{Y}$ and note that Lemma 3.4.1 implies that $1 - 1/C \geq R^*_{X,Y_\rho} \geq R^*_{X,Y}$. As a direct consequence of Lemma 3.4.1, using the SOTA as an upper bound for R^*_X, and $R^*_X \geq 0$ as the lower bound, we can define the valid bounds on R^*_{X,Y_ρ}:

$$\ell_{\mathcal{D}}(\rho) = \rho(1 - 1/C), \quad u_{\mathcal{D}}(\rho) = s_{\mathcal{D}} + \rho(1 - 1/C - s_{\mathcal{D}}),$$

yielding $R^*_{X,Y_\rho} \in [\ell_{\mathcal{D}}(\rho), u_{\mathcal{D}}(\rho)]$. For a fixed method m, we can estimate the *lower bound* $\ell_{\mathcal{D},m}(\rho)$ and the *upper bound* $u_{\mathcal{D},m}(\rho)$ using any BER estimation method on a manipulated dataset $\mathcal{D}_\rho$ obtained by taking $\rho \cdot n$ samples out of $\mathcal{D}$, and randomly changing their labels, whilst keeping the other $(1-\rho) \cdot n$ samples intact. Notice that our evaluation framework needs to change the label on all the data points (i.e. in both the training and test sets). For those estimators that predict only a single value, i.e., try to estimate the exact BER, we set both $\ell_{\mathcal{D},m}(\rho)$ and $u_{\mathcal{D},m}(\rho)$ to that estimate.

As illustrated in Figure 3.2, in order to define the error of a given method m on the modified dataset $\mathcal{D}_\rho$, we can estimate four areas with respect to the curves: *the area where m clearly under/overestimates the Bayes error lower/upper bound*. More formally, for a given method m we define the *lower BER estimator score* $L_{\mathcal{D}}(m)$ and the *upper BER estimator score* $U_{\mathcal{D}}(m)$ by

$$
\begin{aligned}
L_{\mathcal{D}}(m) =& L_{\mathcal{D},\triangleright}(m) + L_{\mathcal{D},\triangleleft}(m), \quad U_{\mathcal{D}}(m) = U_{\mathcal{D},\triangleright}(m) + U_{\mathcal{D},\triangleleft}(m), \\
L_{\mathcal{D},\triangleleft}(m) &= \frac{2C}{C-1} \int_{\rho=0}^{1} \mathbf{1}_{\{\ell_{\mathcal{D},m}(\rho) < \ell_{\mathcal{D}}(\rho)\}} (\ell_{\mathcal{D}}(\rho) - \ell_{\mathcal{D},m}(\rho)) d\rho, \\
L_{\mathcal{D},\triangleright}(m) &= \frac{2C}{C-1} \int_{\rho=0}^{1} \mathbf{1}_{\{\ell_{\mathcal{D},m}(\rho) > u_{\mathcal{D}}(\rho)\}} (\ell_{\mathcal{D},m}(\rho) - u_{\mathcal{D}}(\rho)) d\rho, \\
U_{\mathcal{D},\triangleleft}(m) &= \frac{2C}{C-1} \int_{\rho=0}^{1} \mathbf{1}_{\{u_{\mathcal{D},m}(\rho) < \ell_{\mathcal{D}}(\rho)\}} (\ell_{\mathcal{D}}(\rho) - u_{\mathcal{D},m}(\rho)) d\rho, \\
U_{\mathcal{D},\triangleright}(m) &= \frac{2C}{C-1} \int_{\rho=0}^{1} \mathbf{1}_{\{u_{\mathcal{D},m}(\rho) > u_{\mathcal{D}}(\rho)\}} (u_{\mathcal{D},m}(\rho) - u_{\mathcal{D}}(\rho)) d\rho.
\end{aligned}
$$

Notation $\triangleright$ and $\triangleleft$ reflects the corresponding upper-left and bottom-right triangles in Figure 3.2. The scaling constant is chosen in the way that the random classifier, the one that chooses a label uniformly at random, satisfies $L_{\mathcal{D}}(m) = U_{\mathcal{D}}(m) = 1$. We will use BER estimator scores $L_{\mathcal{D}}(m), U_{\mathcal{D}}(m)$ to assess the performance of existing methods on real-world datasets. We estimate the expectation and standard deviation of the score $S_{\mathcal{D}}(m)$ by sampling linearly 10 values for ρ and constructing at least 5 random datasets $\mathcal{D}_\rho$ for every sampled ρ.

Discussion Going back to the two requirements laid down at the beginning of this section, both are clearly satisfied: **(i)** Lemma 3.4.1, and the construction of the areas (see Figure 3.2), prove that predicting $\ell_{\mathcal{D},m}(\rho) = u_{\mathcal{D},m}(\rho) = R^*_{\mathcal{D}_\rho}$ always yields $L_{\mathcal{D}}(m) = U_{\mathcal{D}}(m) = 0$. In that case the method m is an *optimal* lower/upper-bound estimate. **(ii)** Estimators that consistently predict $\ell_{\mathcal{D},m}(\rho) = 0$ and $u_{\mathcal{D},m}(\rho) = s_{\mathcal{D}}$ clearly have $L_{\mathcal{D}}(m) > 0$ and $U_{\mathcal{D}}(m) > 0$. Furthermore, for two methods m and m', $\mathbb{E}[L_{\mathcal{D}}(m)] < \mathbb{E}[L_{\mathcal{D}}(m')]$ implies that m yields a *better* lower-bound estimate

compared to m', whereas $\mathbb{E}[U_{\mathcal{D}}(m')] > \mathbb{E}[U_{\mathcal{D}}(m)]$ implies that m yields a *better* upper-bound estimate compared to m', in expectation. This allows one to compare two estimators even when they produce valid bounds.

Limitations The main limitation we see in `FeeBee` is the need of having a relatively strong SOTA value. Note that having no SOTA value would allow even random classifier to satisfy $L_{\mathcal{D},\rho}(m) = U_{\mathcal{D},\rho}(m) = 0$. In that sense, for `FeeBee` to give valuable insights, we need SOTA to be close to BER to have a wide range of possible values for the areas.

3.4.2 Analysis of Existing Estimators

Computational feasibility Some BER estimators are not suitable for very large datasets for two reasons: (1) algorithmic or (2) space (i.e., memory) complexity. We restrict the memory and compute time available to run the described evaluation for a single combination of dataset, estimator, transformation and set of hyper-parameters to 45GB and 24h respectively on 4 CPU nodes and access to a single NVIDIA GeForce RTX 2080 Ti GPU. Out of 2376 combinations, 103 ran out of memory and 60 ran out of time (e.g., for high dimensional representations such as BoW, or large datasets such as the test sets of IMDB or YELP).

Hyper-parameters We define a meaningful range of hyper-parameters per method in Table 3.1. The list was manually curated during the evaluation process by shrinking the range of possible values to get the best possible set of hyper-parameters per dataset and per method. For every combination of dataset and feature transformation, we evaluate each method for every value of the hyper-parameter.

Experiment protocol In order to estimate the quantities described in Section 3.4.1, we run every combination of **(1)** dataset, **(2)** pre-trained transformations available for the data modality, **(3)** estimator, and **(4)** the values of their hyper-parameter, multiple times with different random seeds. The code to reproduce all the results along with a public colab that was used to analyze the results are available in the public repository under `https://github.com/DS3Lab/feebee`. We perform 5 independent runs for YELP and 10 independent runs for all other datasets. We sample 11 values linearly between 0.0 and 1.0 for the fraction of label noise ρ. We omit reporting derivations of the estimated mean quantities in all tables and plots with more than two lines, as we observe that the variance mostly lies near zero (except for the kNN-Extrapolate method).

Table 3.2: $L_{\mathcal{D}}(m)$ **(upper part)** and $U_{\mathcal{D}}(m)$ **(lower part)**: The optimal values per method.

Dataset	DE-kNN	KDE	GHP	1NN-kNN	1NN	kNN	kNN-LOO	kNN_Ext
MNIST	0.11	0.41	0.03	0.07	**0.02**	**0.02**	0.03	0.28
CIFAR10	0.14	0.36	0.07	0.10	0.05	**0.03**	**0.03**	0.34
CIFAR100	0.27	0.31	0.20	0.29	0.14	**0.06**	0.07	0.22
IMDB	-	0.49	0.16	0.31	0.25	0.25	**0.15**	0.25
SST2	0.42	0.49	0.44	0.38	0.32	**0.29**	0.34	0.47
YELP	-	0.20	-	-	0.03	**0.00**	-	-
MNIST	0.11	0.41	0.33	0.22	0.32	**0.09**	0.11	0.37
CIFAR10	**0.15**	0.36	0.39	0.26	0.37	**0.15**	0.17	0.20
CIFAR100	0.28	0.31	0.55	0.36	0.49	0.31	0.37	**0.21**
IMDB	-	0.49	0.51	0.39	0.60	0.43	0.34	**0.28**
SST2	0.43	0.49	0.73	**0.42**	0.63	0.48	0.57	0.61
YELP	-	**0.20**	-	-	0.38	0.25	-	-

We now dive into the analysis by examining each presented BER estimator using FeeBee, noting that in each plot a solid line represents an upper bound, whereas a dashed line represents a lower bound.

Analysis of $L_{\mathcal{D}}(m)$ and $U_{\mathcal{D}}(m)$ The main quantities that FeeBee reports are $L_{\mathcal{D}}(m)$ and $U_{\mathcal{D}}(m)$. In Table 3.2 we list the optimal scores $L_{\mathcal{D}}(m)$ and $U_{\mathcal{D}}(m)$ per method and per dataset, meaning that for each m, each dataset and each score, we choose a transformations and hyper-parameters that minimize that score. Further details can be found in Tables B.9–B.20 in Appendix B.2.2, where we report all the corresponding individual areas $L_{\mathcal{D},\triangleleft}$, $L_{\mathcal{D},\triangleright}$, $U_{\mathcal{D},\triangleleft}$ and $U_{\mathcal{D},\triangleright}$, with further example plots presented in Figure B.4 in Appendix B.2.3.

For $L_{\mathcal{D}}(m)$ we observe that 1NN, kNN, kNN-LOO and GHP are consistently outperforming all the other methods, with either kNN or kNN-LOO being the best choice on each dataset. It is important to note that the main contribution to $L_{\mathcal{D}}(m)$ in the case of 1NN, kNN, kNN-LOO and GHP comes from $L_{\mathcal{D},\triangleright}(m)$, whereas for DE-kNN, KDE and kNN-Extrapolate there is significant contribution from $L_{\mathcal{D},\triangleleft}(m)$. This yields that the first group will only get better as more transformations become available (by reducing the bias that a transformation introduces), whereas the second group provides less informative lower-bound estimators. For $U_{\mathcal{D}}(m)$, we see that there is no method that consistently outperforms the others. Out of the well-performing

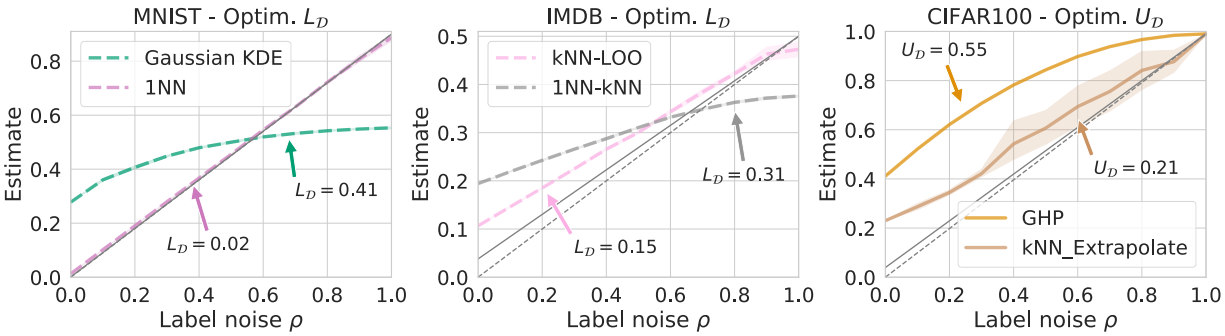


Figure 3.3: Plotting two methods where one of them has low and the other has high **(left, middle)** $L_{\mathcal{D}}(m)$, or **(right)** $U_{\mathcal{D}}(m)$, confirming that numerical values correspond to estimator's quality. The shaded area represents the 95% quantiles.

methods on $L_{\mathcal{D}}(m)$, 1NN and GHP are tangibly inferior to kNN and kNN-LOO, noting similarly as above that the main contribution for these methods comes from $U_{\mathcal{D},\rho}(m)$ which will decrease with better feature transformations.

We also note that BER estimators are often performing better on certain type of feature transformations, as seen in Tables B.9–B.20 in the supplementary materials. E.g., 1NN and kNN perform better on pre-trained embeddings, whereas kNN-Extrapolate performs best under transformations that reduce the dimension, such as a low-dimensional PCA (supporting Snapp et al. (1991)).

Numerical values and estimator's quality A key question is whether the numerical quantities correspond to the performance: *does a lower $L_{\mathcal{D}}(m)$ imply a better lower-bound estimator?* For simplicity, in Figure 3.3 we plot over 3 different dataset, 2 methods each such that one has a high score and the other has a low score and see that their ability to follow the evolution of the BER differs. We provide full graphs of each methods over all datasets in Appendix B.2.2, positively answering the above question.

Minimizing $\ell_{\mathcal{D},m}(0)$ vs minimizing $L_{\mathcal{D}}(m)$ As described in Section 3.4.1, whilst an estimator that always predicts zero gives a valid lower bound, its behavior is non-informative. However, for some methods tuning for $\ell_{\mathcal{D},m}(0)$ might be (close to) optimal for $L_{\mathcal{D}}(m)$. Thus, in Table 3.3, for each method and over each dataset we report the difference in the optimal $L_{\mathcal{D}}(m)$, i.e. the best hyper-parameters and the best transformation, and the one chosen by hyper-parameters and transformation that minimize $\ell_{\mathcal{D},m}(0)$. We observe that 1NN and GHP are the only ones that are fully robust in the sense that choosing the best hyper-parameters and transformation for $\ell_{\mathcal{D},m}(0)$

Table 3.3: $\ell_{\mathcal{D},m}(0)$ vs $L_{\mathcal{D}}(m)$: The difference between the *optimal* $L_{\mathcal{D}}(m)$ and the $L_{\mathcal{D}}(m)$ that is calculated for hyper-parameters and transformations that minimize $\ell_{\mathcal{D},m}(0)$.

Dataset	DE-kNN	KDE	GHP	1NN-kNN	1NN	$k_{>1}$NN	kNN-LOO	kNN_Ext
MNIST	0.24	0.59	**0.00**	0.56	**0.00**	0.06	0.04	0.37
CIFAR10	0.21	0.64	**0.00**	0.52	**0.00**	0.11	0.10	0.42
CIFAR100	0.10	0.69	**0.00**	0.30	**0.00**	0.05	0.02	0.59
IMDB	-	0.01	**0.00**	0.26	0.01	0.14	0.20	0.33
SST2	0.00	0.50	0.01	0.20	**0.00**	0.11	0.09	0.30
YELP	-	0.09	-	-	**0.00**	**0.00**	-	-

yields close to optimal $L_{\mathcal{D}}(m)$, whereas for every other method we can find examples in which the choice based on $\ell_{\mathcal{D},m}(0)$ is significantly suboptimal for $L_{\mathcal{D}}(m)$.

Influence of SOTA In general, we observe that our framework works when the SOTA value is not too weak. This can be seen by comparing the results on YELP (SOTA error of 27.80%) vs other datasets (SOTA errors of at most 3.92%). For example, on YELP kNN has $L_{\mathcal{D}}(m) = 0.0$ (see Table 3.2), whilst clearly having difficulties in following the BER evolution (see Figure B.3a in Appendix B.2.2, top-right). A stronger SOTA would detect such difficulties for kNN on YELP through a non-zero $L_{\mathcal{D}}(m)$. Other datasets involved in this framework satisfy that these have been intensively studied by the research community, particularly in the last few years, resulting in strong SOTA values.

Sensitivity to changes in SOTA In order to study FeeBee's robustness with respect to changes in SOTA values, we perform additional experiments by altering SOTA. In Figure 3.4 we show three examples in which SOTA values inserted into the framework are SOTA $\pm\ \delta\cdot$ SOTA, for $\delta \in \{0, 0.05, 0.1, 0.25\}$, which, we believe, represent realistic improvements in the near future. On datasets which have strong SOTAs (all but YELP), we see that modifying the values has almost no impact on FeeBee's insights (kNN and kNN-LOO perform best on CIFAR10 and CIFAR100, followed by 1NN and GHP, cf. Table 3.2). By comparing CIFAR10 and CIFAR100 we see that the robustness is further improved with stronger SOTA values. Therefore, we conclude that FeeBee is rather robust to small changes in the values when SOTA is relatively strong. On the other hand, when the SOTA value is weak, as in the case of YELP, Figure 3.4 implies that the sensitive of SOTA value starts to increase, introducing significantly more uncertainty than in the

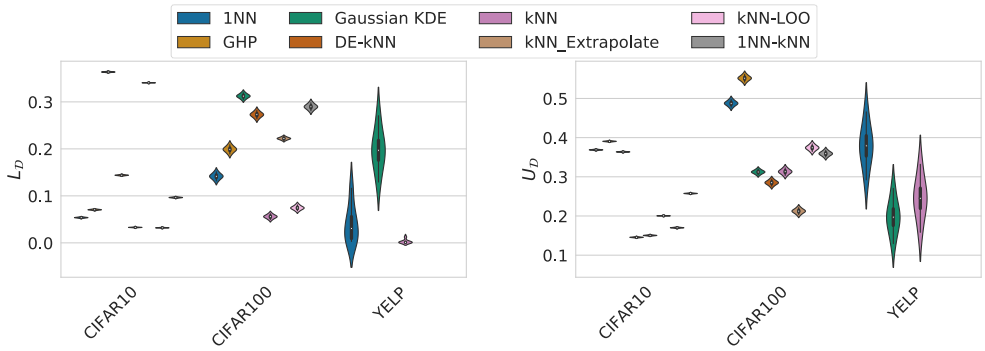
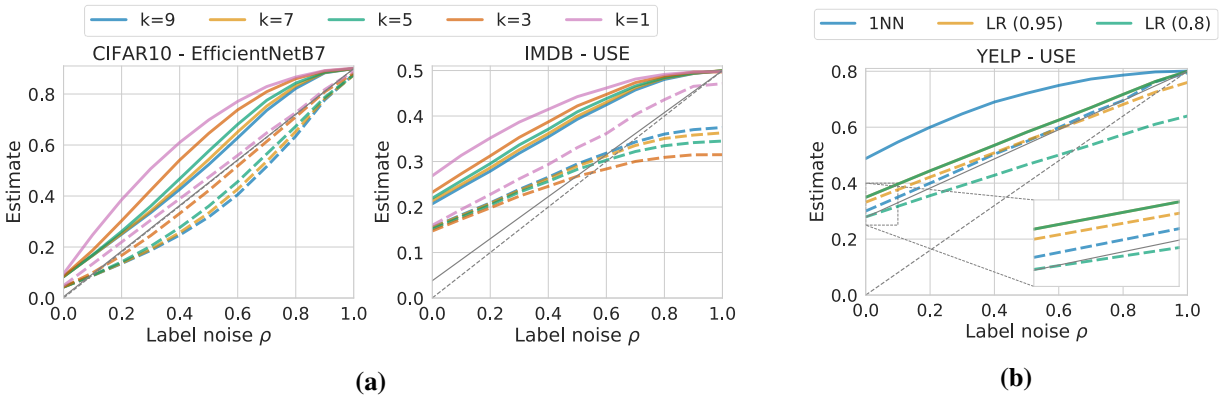


Figure 3.4: Sensitivity of `FeeBee` with respect to changes in the SOTA values for **(left)** $L_{\mathcal{D}}(m)$, and **(right)** $U_{\mathcal{D}}(m)$. Strong SOTA values are robust to changes (CIFAR10, CIFAR100), whereas weak SOTA values suffer from greater uncertainty (YELP).

other datasets. However, we remark that the insights are still useful and preserved even in this difficult case.

Alternative flipping strategies One could create noisy labels in many different ways. For example, by assigning different flipping probabilities to different classes, or by conditioning on certain features and flipping differently on these features (e.g., different probabilities for day vs night images). In most of these cases one could produce an analogue of Lemma 3.4.1 which would provide foundations for an alternative framework. However, each such method would put certain constraints on the applicable datasets which makes the task of constructing a framework inherently more difficult. Having in mind that `FeeBee` in this simplest form is already successful in distinguishing existing estimators, and our belief that it will stay successful in the future, we opt for using the simplest such framework.

GHP vs 1NN When looking at the derivation of both the upper and lower bounds of GHP (Theorem 1 in Sekeh et al. (2020)), we see that, asymptotically, this method averages the number of dichotomous edges connecting a sample from two different classes, on the minimum spanning tree (MST) that is connecting all the samples in Euclidean space. The 1NN-LOO estimator implicitly performs the same task over the 1-nearest-neighbor graph, instead of the MST. Intuitively, the 1-nearest-neighbor graph should lead to a slightly lower error rate when compared to the MST. In particular, 1NN-LOO should outperform GHP if the feature transformation forms well-separable clusters of samples from the same class. Nevertheless, showing this is beyond the scope of this thesis and left for future work.



(a) **(b)**

Figure 3.5: **(a)** kNN estimator for $k > 1$: Increasing k increases $L_{\mathcal{D},\blacktriangleleft}(kNN)$. **(b)** Logistic regression (LR) as BER estimator: Larger scaling decreases $U_{\mathcal{D}}(m)$ further, but increases $L_{\mathcal{D},\blacktriangleleft}(m)$.

kNN estimator for $k > 1$ Due to its success in minimizing $L_{\mathcal{D}}(m)$, whilst being inferior to 1NN and GHP in choosing hyper-parameters and transformations based on $\ell_{\mathcal{D},m}(0)$, we further examine the kNN estimator. In Figure 3.5a we plot 1NN and kNN for $k \in \{3, 5, 7, 9\}$ on two representing datasets and over transformations that are minimizing $L_{\mathcal{D}}(m)$, with corresponding values in Table 3.4. We see that increasing k increases $L_{\mathcal{D},\blacktriangleleft}(m)$, which makes kNN a worse lower-bound BER estimator since it provides a less informative lower bound. We believe that the reason for such a behavior lies in the fact that for $k > 1$ the only relevant bounds are given in the case when $C = 2$, by Devroye Devroye (1981), whereas for $C > 2$ and $k > 1$ there are no known bounds as strong as the one of Cover and Hart Cover and Hart (1967) for $k = 1$. This results in a larger gap between the BER and the lower bound in the asymptotic regime. In the finite regime, at the moment this is mitigated by the bias that feature transformations introduce, however, in the future we expect this bias to decrease and, thus, kNN might further suffer from these loose bounds.

Improved upper bounds In Table 3.2, we see that BER estimators are better at minimizing $L_{\mathcal{D}}(m)$ than $U_{\mathcal{D}}(m)$. Even though any classifier can be used as an upper-bound BER estimator, to the best of our knowledge, the estimators 1NN and kNN from Equations 3.2.2 and 3.2.4 are the only classifier-based estimators for which one has theoretical guarantees (Cover and Hart (1967) for 1NN and Devroye (1981) for kNN) for the lower bound on the BER. However, one can construct a lower-bound estimator by scaling the accuracy of the estimator by some constant $c \in (0, 1)$. We test the simplest such estimator – logistic regression (LR) on top of the frozen representations, and several scaling constants. In Figure 3.5b we observe that even though LR

Table 3.4: Impact of $k > 1$ and LR Model with different constants vs 1NN.

Dataset	Method	Transformation	$U_{\mathcal{D},\triangleright}(m)$	$L_{\mathcal{D}}(m)$	$L_{\mathcal{D},\triangleright}(m)$	$L_{\mathcal{D},\triangleleft}(m)$
CIFAR10	1NN	EfficientNet-B7	0.37	0.05	0.05	0.00
	3NN	EfficientNet-B7	0.28	0.04	0.01	0.03
	5NN	EfficientNet-B7	0.21	0.11	0.01	0.11
	7NN	EfficientNet-B7	0.18	0.14	0.01	0.14
	9NN	EfficientNet-B7	0.16	0.16	0.01	0.15
IMDB	1NN	USE	0.60	0.26	0.25	0.01
	3NN	USE	0.52	0.30	0.12	0.17
	5NN	USE	0.48	0.27	0.14	0.13
	7NN	USE	0.45	0.26	0.15	0.11
	9NN	USE	0.43	0.25	0.16	0.09
YELP	1NN	USE	0.38	0.03	0.03	0.00
	LR Model (0.8)	USE	0.10	0.08	0.00	0.08
	LR Model (0.95)	USE	0.10	0.06	0.05	0.01

yields a significantly better estimator of the upper bound than 1NN, it has worse $L_{\mathcal{D}}(m)$ even for the best scaling constant. Furthermore, we note that increasing the scaling, in order to reduce $L_{\mathcal{D},\triangleright}(m)$, further increases $L_{\mathcal{D},\triangleleft}(m)$, yielding a less informative lower-bound estimator, also visible in Table 3.4.

3.5 On the Impact Of Feature Transformations

As a major finding of `FeeBee`, we established that the *1NN*-based estimator over publicly available feature transformations is a powerful one — on par or better than all other estimators when it comes to performance, whilst being highly scalable and insensitive to hyper-parameters

The main goal of this section is to provide a novel theoretical guarantee on the behavior of convergence rates of a kNN classifier over feature transformations. We state the main ingredients and show how they are used to prove the main theorem, whilst the proofs for these ingredients can be found in Appendix A.1 of the supplementary material, in the same order as presented in this section.

Overview of Results We begin by providing an overview of our results before expanding them in details in the next two sections.

It is well known that a kNN classifier can converge arbitrarily slowly if one does not assume any regularity (Antos et al., 1999). A common starting assumption for determining convergence rates of a kNN classifier is that of the a-posteriori probability $\eta(x)$ being an *L-Lipschitz* function, that is for all $x, x' \in \mathcal{X}$,

$$|\eta(x) - \eta(x')| \leq L\|x - x'\|. \tag{3.5.1}$$

In order to stay exposition friendly, we will stay close to the Lipschitz condition by imposing the mildest assumptions, as found in Györfi et al. (2002). We assume that we are given the *raw* data in $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X} \subset \mathbb{R}^D$. Our reference point will be Theorem 6.2 from Györfi et al. (2002) which states that if $\mathcal{X}$ is bounded and $\eta(x)$ is L-Lipschitz, then

$$\mathbb{E}_n\left[(R_X)_{n,k}\right] - R_X^* = O\left(\frac{1}{\sqrt{k}}\right) + O\left(L\left(\frac{k}{n}\right)^{1/D}\right). \tag{3.5.2}$$

The main result of this section is an extension of the above to *transformed* data.

Theorem 3.5.1. *Let $\mathcal{X} \subseteq \mathbb{R}^D$ and $\tilde{\mathcal{X}} \subseteq \mathbb{R}^d$ be bounded sets, and let (X, Y) be a random vector taking values in $\mathcal{X} \times \{0, 1\}$. Let $g\colon \tilde{\mathcal{X}} \to \mathbb{R}$ be an L_g-Lipschitz function. Then for all transformations $f\colon \mathcal{X} \to \tilde{\mathcal{X}}$, one has*

$$\mathbb{E}_n\left[(R_{f(X)})_{n,k}\right] - R_X^* = O\left(\frac{1}{\sqrt{k}}\right) + O\left(L_g\left(\frac{k}{n}\right)^{1/d}\right) + O\left(\sqrt[4]{\mathcal{L}_{g,X}(f)}\right). \tag{3.5.3}$$

Implications An obvious comparison of Equations 3.5.2 and 3.5.3 shows that applying a transformation can introduce a non-negligible bias, and thus a feature transformation should not be used if one has an infinite pool of data points. However, as we will confirm in the experimental section, in the finite-sample regime the benefit of reducing the dimension and, often more importantly, changing the geometry of the space using (pre-trained) transformations significantly outweighs the loss introduced through this bias. In practice, g is typically chosen to be a linear layer with the sigmoid output function, that is $g_w(x) = \sigma(\langle w, x\rangle)$, where $\sigma(x) := (1 + e^{-x})^{-1}$. It is easy to see that σ is $1/4$-Lipschitz since $d\sigma/dx = \sigma(1 - \sigma) \leq 1/4$, whereas $x \mapsto \langle w, x\rangle$ is $\|w\|_2$-Lipschitz by the Cauchy-Schwarz inequality, implying that g_w is $\|w\|_2/4$-Lipschitz. Therefore, for any w we can simply insert $\mathcal{L}_{g_w,X}$ and $\|w\|_2$, since they comprise a valid bound in

Equation 3.5.3. In particular, this aligns with a common heuristic of minimizing a loss, while introducing the additional task of capturing $\|w\|_2$ in the process.

The proof of Theorem 3.5.1 is divided into two parts, motivated by

$$\mathbb{E}_n\left[(R_{f(X)})_{n,k}\right] - R^*_X = \underbrace{\mathbb{E}_n\left[(R_{f(X)})_{n,k}\right] - R^*_{f(X)}}_{\text{convergence rates of vanilla kNN on } f(X)} + \underbrace{R^*_{f(X)} - R^*_X}_{\text{safety of } f\text{: } \Delta^*_{f,X}}. \tag{3.5.4}$$

We examine these parts in the next two sections.

3.5.1 Safe Transformations

We start by bounding the increase in the Bayes error that a transformation can introduce.

In order to motivate what follows, note that one can rewrite $\Delta^*_{f,X}$ as[2]

$$\Delta^*_{f,X} = \mathbb{E}_{x\sim X}\left[p(y_x \mid x) - p_{f^{-1}}(y_{f(x)} \mid f(x))\right], \tag{3.5.5}$$

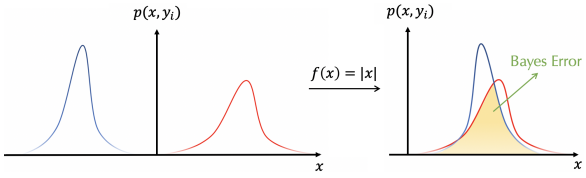
where $y_x = \arg\max_{y\in\mathcal{Y}} p(y|x)$ and $y_{f(x)} = \arg\max_{y\in\mathcal{Y}} p_{f^{-1}}(y|f(x))$.

It is clear that any feature transformation can only increase the Bayes error (as seen in Figure 3.6a), due to the fact that max(.) is a convex function. Hence, in order to understand and control this increase, we define and analyze *safe* transformations – those which increase the Bayes error only for a small value.

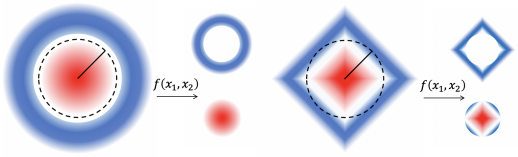
Definition 3.5.2. *We say that a transformation $f\colon \mathcal{X} \to \widetilde{\mathcal{X}}$ is* δ-safe *(with respect to p) if $\Delta^*_{f,X} \le \delta$, and* δ-unsafe *(with respect to p) if there is no $\delta' < \delta$ such that $\Delta^*_{f,X} \le \delta'$.*
When $\delta = 0$ we simply say that a function is safe *or* unsafe.

Motivated by Equation 3.5.5, we see that any injective function f is safe, since for all $x \in \mathcal{X}$ one has $f^{-1}(\{f(x)\}) = \{x\}$. For example, this implies that $x \mapsto (x, f(x))$ is a safe transformation, for any map f. In the supplementary material we weaken the notion of injectivity, allowing us to deduce that the CReLU (Concatenated ReLU) activation function (Shang et al., 2016), defined as $x \mapsto (x^+, x^-)$, where $x^+ = \max\{x, 0\}$ and $x^- = \max\{-x, 0\}$, is a safe function. Interestingly, both x^+ and x^- are $1/2$-unsafe, showing that unsafe functions can be concatenated into a safe one.

[2]When a deterministic feature transformation $f\colon \mathcal{X} \to \widetilde{\mathcal{X}}$ is applied, we define the induced joint probability by $p_{f^{-1}}(\widetilde{x}, y) = p(X \in f^{-1}(\{\widetilde{x}\}), Y = y)$, where $f^{-1}(\{\widetilde{x}\}) = \{x \in \mathcal{X}\colon f(x) = \widetilde{x}\}$ is the preimage of $\widetilde{x}$.



(a) An example in which the Bayes Error increases after applying $f(x) = |x|$, a function that is not injective and in this case collapses one class onto another.



(b) A transformation trained on the *source* distribution can increase the BER when applied to the *target* distribution. Here $f(x_1, x_2) = \mathbf{1}_{\{x_1^2 + x_2^2 \geq 1\}}$ leaves the BER at 0 after applying f on the left, whereas it increases the BER on the right, where the inner circle collects parts of both distributions.

Figure 3.6: Transformations introduce bias by increasing the BER.

When discussing safety using injective functions, one is restricted to only considering how f behaves on $\mathcal{X}$, without using the information that $\mathcal{Y}$ might facilitate. For example, a function that reduces the dimension is not injective, whilst in reality we might be able to avoid the loss of information that $\mathcal{Y}$ carries about $\mathcal{X}$, when it comes to the Bayes error. A perfect such example is the map $f_{\mathcal{Y}} \colon \mathcal{X} \to \mathcal{Y}$ defined by $f_{\mathcal{Y}}(x) := y_x$, as $y_{f_{\mathcal{Y}}(x)} = \arg\max_{y \in \mathcal{Y}} p_{f_{\mathcal{Y}}^{-1}}(y | f_{\mathcal{Y}}(x)) = \arg\max_{y \in \mathcal{Y}} p_{f_{\mathcal{Y}}^{-1}}(y | y_x) = y_x$. Through Equation 3.5.5 we see that $f_{\mathcal{Y}}$ does not change the Bayes error even though it reduces the size (of $\widetilde{\mathcal{X}}$) to the number of classes. Thus, we now provide an alternative sufficient condition for δ-safe functions, which takes into account both $\mathcal{X}$ and $\mathcal{Y}$ through the *Kullback-Leibler divergence*[3], given by

$$D_{KL}(q_1(x) \,\|\, q_2(x)) := \sum_{x \in \mathcal{X}} q_1(x) \log \frac{q_1(x)}{q_2(x)}.$$

We further denote $D_{KL}(p(y|x) \,\|\, p_{f^{-1}}(y|f(x))) := D_{KL}\left(p(x,y) \,\|\, \frac{p(x)}{p_{f^{-1}}(f(x))} p_{f^{-1}}(f(x), y)\right)$, which one can think of as the loss in the *mutual information* after applying f.

Lemma 3.5.3. *Let $\mathcal{X}$ and $\mathcal{Y}$ be finite sets, and let $f \colon \mathcal{X} \to \widetilde{\mathcal{X}}$ be a transformation such that $D_{KL}(p(y|x) \,\|\, p_{f^{-1}}(y|f(x))) \leq (2/\ln 2)\delta^2$. Then f is δ-safe.*

We remark that the converse of Lemma 3.5.3 does not hold, meaning that one can have a δ-safe function for which the corresponding KL divergence is larger than $(2/\ln 2)\delta^2$. For example, the above mentioned $f_{\mathcal{Y}}$ is a safe transformation, whereas the KL-divergence is strictly positive as

[3] Without loss of generality, we assume that $\mathcal{X}$ is finite and logarithm to the base 2.

soon as $p(y|x) \neq p_{f^{-1}}(y|f(x))$, for some x and y, which is easy to construct. On the other hand, we can prove that the bounds are tight in the following sense.

Lemma 3.5.4. *Let $\mathcal{X}$, $\widetilde{\mathcal{X}}$ be finite sets satisfying $|\widetilde{\mathcal{X}}| < |\mathcal{X}|$, $\mathcal{Y} = \{0, 1\}$, and let $f : \mathcal{X} \to \widetilde{\mathcal{X}}$ be a transformation. For any $\delta \in [0, 1/2)$ there exist random variables X, Y, taking values in $\mathcal{X}, \mathcal{Y}$, such that $R^*_{f(X)} - R^*_X = \delta$ and $D_{KL}(p(y|x) \,||\, p_{f^{-1}}(y|f(x))) = (2/\ln 2)\delta^2 + O(\delta^4)$.*

A key challenge with the proposed approach is that the feature transformation might have been trained on a significantly different joint probability distribution, and, as such, might change the Bayes error in an unfavourable way when applied to our distribution of interest (Figure 3.6b). It is clear that in the worst case this can induce an unbounded error in the estimation. In practice, however, these feature transformations are learned for each domain specifically (e.g. deep convolutional neural networks for the vision domain, transformers for NLP tasks), and recent results in transfer learning show that such transformations do indeed *transfer* to a variety of downstream tasks (Zhai et al., 2019b). Guided by this intuition we derive a sufficient condition under which our notion of safety is preserved on similar distributions.

Theorem 3.5.5. *Let p_S and p_T be two probability distributions defined on $\mathcal{X} \times \mathcal{Y}$ that satisfy $D_{KL}(p_S \,||\, p_T) \leq \varepsilon^2/(8 \ln 2)$. If a transformation f is δ-safe with respect to p_S, then f is $(\delta + \varepsilon)$-safe with respect to p_T.*

From Equation 3.5.2 we know that a kNN classifier on $f(X)$ converges to $R^*_{f(X)}$ under some mild assumptions. Since one aims at solving the original task, in other words to examine the convergence of $(R_{f(X)})_{n,k}$ to R^*_X, we need to make the transition to R^*_X by using some form of $\mathcal{L}_{g,X}(f)$. We conclude this section by stating the theorem that allows this transition.

Theorem 3.5.6. *For every transformation $f : \mathcal{X} \to \tilde{\mathcal{X}}$, one has $\Delta^*_{f,X} \leq 2\sqrt{\mathcal{L}_{g,X}(f)}$.*

3.5.2 Convergence Rates of kNN Classifiers over Transformations

In this section we derive convergence rates over transformed features, which together with Theorem 3.5.6 yield the proof of Theorem 3.5.1, via Equation 3.5.4. We provide statements for X, i.e. the raw data, keeping in mind that we apply the main theorem of this section on $f(X)$ later on. In order to get a convergence rate that works on $f(X)$ without imposing any particular structure on f, one needs a weaker notion than the Lipschitz condition. It suffices to define the *probabilistic Lipschitz condition*.

Definition 3.5.7. *Let $\varepsilon, \delta, L > 0$, and let $X, X' \in \mathcal{X}$ be random variables sampled using the same joint probability distribution p on $\mathcal{X} \times \{0, 1\}$. We say that $\eta(x) = p(1|x)$ is* (ε, δ, L)-probably Lipschitz *if*

$$\mathbb{P}\Big(|\eta(X) - \eta(X')| \leq \varepsilon + L\|X - X'\|\Big) \geq 1 - \delta.$$

With such a condition, similar to the one of Theorem 6.2 by Györfi et al. (2002), we can prove the following result.

Theorem 3.5.8. *Let $\mathcal{X} \subseteq \mathbb{R}^d$ be a bounded set, $X \in \mathcal{X}$, $Y \in \{0, 1\}$. If $\eta(x)$ is (ε, δ, L)-probably Lipschitz, then*

$$\mathbb{E}_n\left[(R_X)_{n,k}\right] - R^*_X = O\left(\frac{1}{\sqrt{k}}\right) + O\left(L\left(\frac{k}{n}\right)^{1/d}\right) + O\left(\sqrt{\delta} + \varepsilon\right). \tag{3.5.6}$$

The first term represents the variance in the training set, the second term comes from the Lipschitz-like structure of X, whilst the last term comes from the need of the probabilistic Lipschitz condition, which will later allow us to avoid additional constraints on f. We use Theorem 3.5.8 on $f(X)$, which needs an evidence of probably Lipschitz condition with respect to the g-squared loss. The following consequence of Markov's inequality yields the desired evidence. Since it is applied on $f(X)$, it suffices to have the identity function in place of f.

Lemma 3.5.9. *Let g be an L-Lipschitz function. Then for any real number $\varepsilon > 0$, the function $\eta(x)$ is $(\varepsilon, 8\mathcal{L}_{g,X}(id)/\varepsilon^2, L)$-probably Lipschitz.*

Proof sketch of the main result Deducing Theorem 3.5.1 is now straightforward – Lemma 3.5.9 says that we can apply Theorem 3.5.8 with X substituted by $f(X)$, meaning that for any f and any $\varepsilon > 0$,

$$\mathbb{E}_n\left[(R_{f(X)})_{n,k}\right] - R^*_{f(X)} = O\left(\frac{1}{\sqrt{k}}\right) + O\left(L\left(\frac{k}{n}\right)^{1/d}\right) + O\left(\frac{\sqrt{\mathcal{L}_{g,f(X)}(id)}}{\varepsilon} + \varepsilon\right). \tag{3.5.7}$$

We optimize this by setting $\varepsilon = \sqrt[4]{\mathcal{L}_{g,f(X)}(id)}$. In the supplementary material, we prove an easily attainable upper bound $\mathcal{L}_{g,f(X)}(id) \leq \mathcal{L}_{g,X}(f)$, which, together with the final transition from $R^*_{f(X)}$ to R^*_X that is available via Theorem 3.5.6, proves the main result.

Discussions We could further optimize the exponent of $\mathcal{L}_{g,X}$, however, we leave this for future work, as it already serves our purpose of providing a connection between the g-squared loss of f

and the Lipschitz constant of g. Furthermore, one might argue that Equation 3.5.2 could be used to yield a better rate of convergence than the one in Equation 3.5.3, provided that the Lipschitz constant of $\eta_{f^{-1}}$ is known for every f. However, checking whether $\eta_{f^{-1}}$ is L-Lipschitz is not a feasible task. For example, computing the best Lipschitz constant of a deep neural network is an NP-hard problem (Scaman and Virmaux, 2018), whilst upper bounding the Lipschitz constant over high-dimensional real-world data and many transformations becomes impractical. Our result provides a practical framework that can easily be deployed for studying the influence of feature transformations on a kNN classifier, as illustrated in the next section.

3.5.3 Experimental Results

The goal of this section is to show that one can use the upper bounds on the convergence rate of kNN, derived in the previous sections, to explain the impact of different feature transformations on the kNN performance. This works particularly well when compared with simpler metrics mentioned in the introduction, such as the dimension or the accuracy of another trained classifier on the same space.

In order to empirically verify our theoretical results, as discussed in the implications of Theorem 3.5.1, we focus on the logistic regression model, since it is most commonly used in practice, and for which one can easily calculate the Lipschitz constant. More precisely, we examine $g(x) = \sigma(\langle w, x \rangle)$, with $L_g = \|w\|_2$. When dealing with multi-class tasks, we use

$$g(x) = \text{softmax}\left(W^T x + b\right),$$

whilst reporting $\|W\|_F$, the Frobenius norm of the weights, in place of $\|w\|_2$. The Frobenius norm for a matrix $W \in \mathbb{R}^{m \times n}$ is defined as

$$\|W\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} \left(w_{i,j}\right)^2}$$

Calculating the g-squared loss of f is not feasible in practice, since one usually does not know the true distribution $\eta(X)$. However, using the fact that $\mathbb{E}_{Y|X=x}(\eta(x) - Y) = 0$, one can decompose the loss as

$$\mathcal{L}_{g,X}(f) = \mathbb{E}_{X,Y}\left((g \circ f)(X) - Y\right)^2 + \mathbb{E}_{X,Y}\left(\eta(X) - Y\right)^2.$$

As the second term does not depend on g or f, we can rank feature transformations by estimating the first term for a fixed g. With that in mind, when we compare different transformations, we

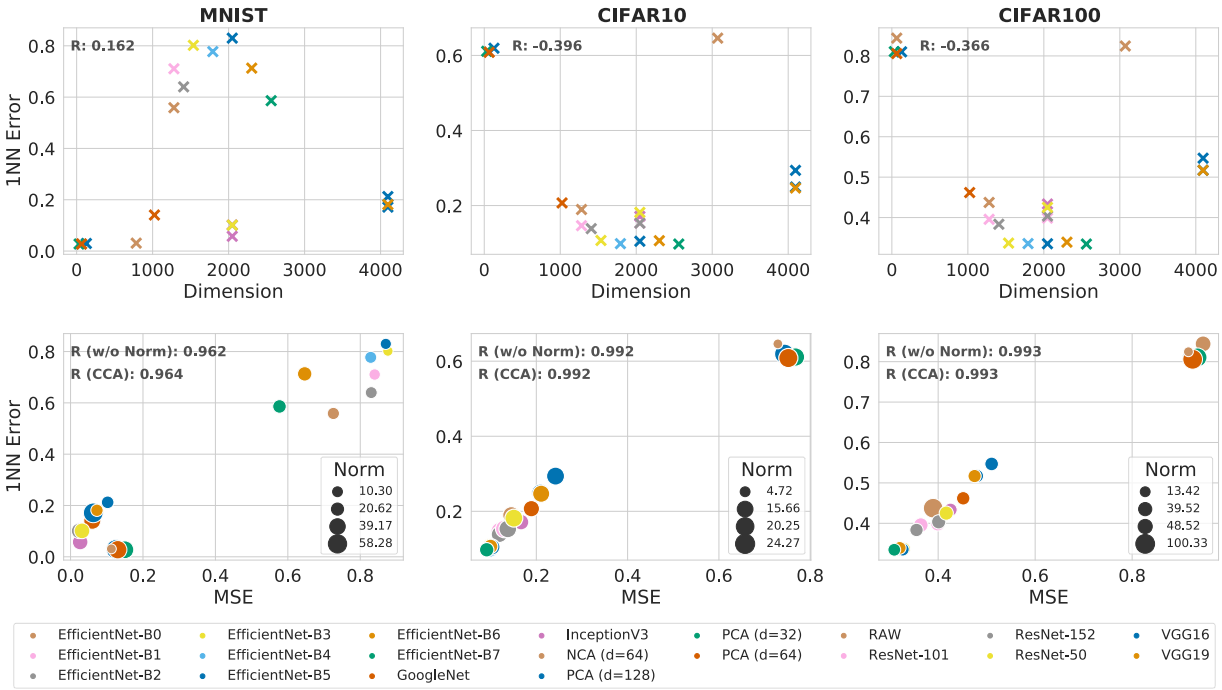


Figure 3.7: Results on computer vision datasets. **(Top row)** There is a poor linear correlation between the feature dimension and the 1NN error. **(Bottom row)** Comparing the 1NN error with the MSE shows a very high linear correlation. Combining MSE with the norm marginally improves the correlation coefficients.

report the *mean squared error*[4] of a test set consisting of M elements, defined by

$$MSE_g(f, W, b) = \frac{1}{M} \sum_{i \in [M]} (\text{softmax}(W^T f(x_i) + b) - y_i)^2.$$

For kNN, we restrict ourselves to the Euclidean distance, the most commonly used distance function. Furthermore, we set $k = 1$ for most of the analysis, conducting an empirical analysis of the influence of $k > 1$ at the end of this section.

kNN Classifier In order to illustrate the convergence rates, we subsample the training samples 10 times linearly (decreasingly), and perform 30 independent runs in order to report the variance. We plot the 95% confidence intervals on all the convergence graphs.

[4]In these terms, when one wants to rank different classifiers, the MSE is often referred to as the *Brier score* (we refer an interested reader to Brier et al. (1950); Cohen and Goldszmidt (2004)).

Logistic Regression Classifier We train all the logistic regression models (on all the datasets and transformations mentioned earlier) using SGD with a momentum value of 0.9 and a batch size of 64 on the entire training set for 200 epochs, minimizing the cross entropy loss. We report the best achieved test set error (misclassification error) and mean squared error (MSE) using different values of L_2 regularizer $(0.0, 0.0001, 0.001, 0.01, 0.1)$ and initial learning rates $(0.0001, 0.001, 0.01, 0.1)$. We pre-process the input before training by normalizing the features to range between -1 and 1.

Training infrastructure Training of the logistic regression models and evaluating kNN was executed on a single NVIDIA Titan Xp GPU.

Evaluation Metric We use *Pearson's r correlation* (Benesty et al., 2009) to analyze different approaches for comparing feature transformations with respect to the kNN error. Pearson's r ranges from -1 to 1, with 1 implying that two random variables have a perfect positive (linear) correlation, where -1 implies a perfect negative correlation.

Feature Dimension We start by inspecting the dimension of the resulting feature space and its achieved kNN error. We exhibit a very poor (or even negative) correlation on all text datasets, as illustrated in the top rows of Figures 3.7 and 3.8.

MSE vs 1NN Error On computer vision tasks we observe high correlation when comparing the 1NN error with the MSE, with marginal improvements when the norm is included, as reported in the bottom row of Figure 3.7. This is in line with Equation 3.5.3 by simply concluding that the last term dominates.

On text classification datasets the MSE alone struggles to explain the performance of 1NN on top of feature transformations, as visible in the bottom row of Figure 3.8. Without considering the norm (i.e., ignoring the size of the circles on Figure 3.8), we see a number of misjudged (important) examples, despite having a positive correlation.

Validation Based on Our Theoretical Results – MSE and Norm In order to report the correlation between multiple variables (notably the 1NN error vs. the combination of MSE and the norm), we first reduce the dimension of the latter by performing a canonical-correlation analysis (CCA) (Thompson, 2005) which enables us to evaluate Pearson's r afterwards.

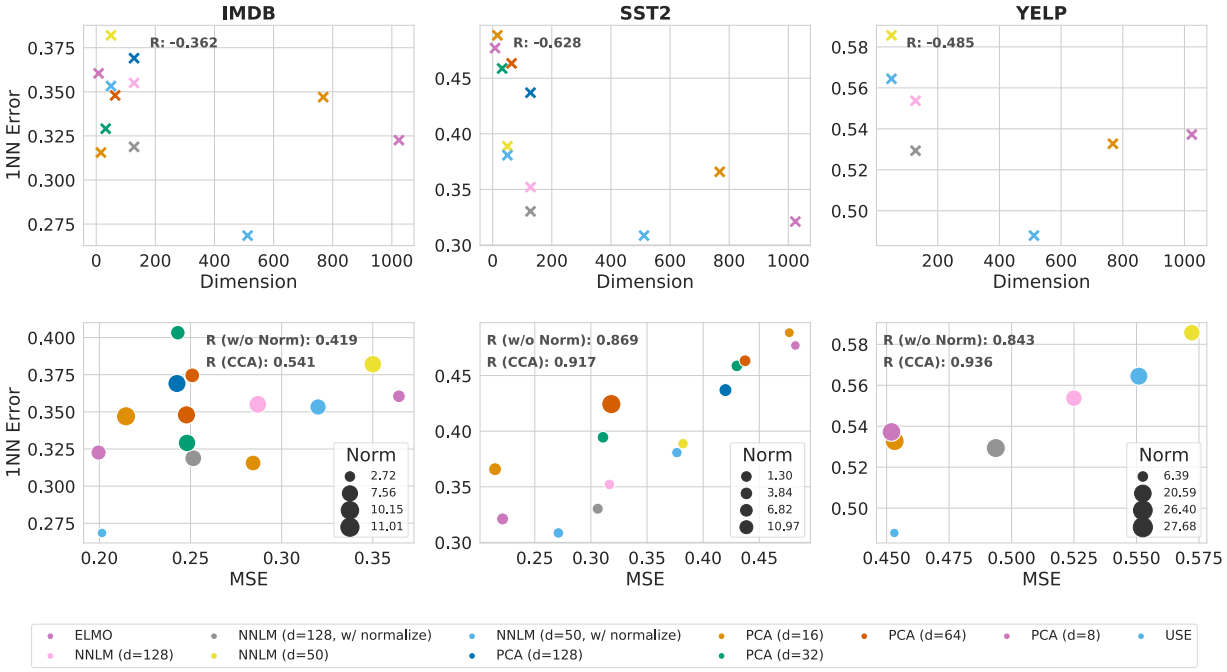


Figure 3.8: Results on text classification datasets. **(Top row)** There is a poor linear correlation between the feature dimension and the 1NN error. **(Bottom row)** Comparing the 1NN error with MSE without the norm (i.e., by ignoring the sizes of the circles) leads a strong positive correlation. Including the norm improves this further, particularly on important cases (the bottom left corner) which makes the inclusion of norm significant.

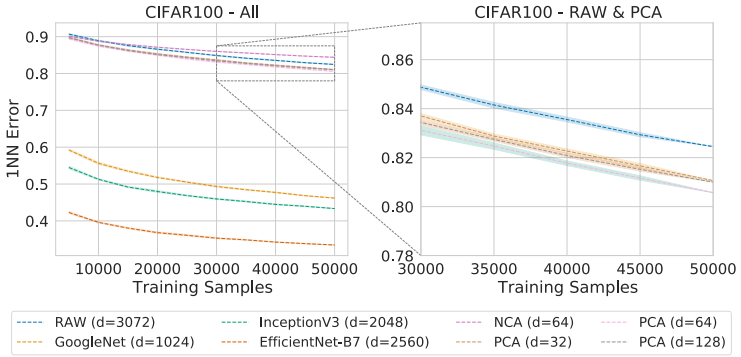


Figure 3.9: Impact of the dimension on CIFAR100 using all involved transformations **(Left)**, and PCA-based transformation only **(Right)**.

As visible in the second row of Figure 3.8, there is a significant improvement in correlation on all the text classification datasets. Furthermore, this method enables us to separate transformations that achieve similar MSE, but with different norms, in particular on those transformations that perform well. We can see that in those cases, the smaller norm implies smaller 1NN error.

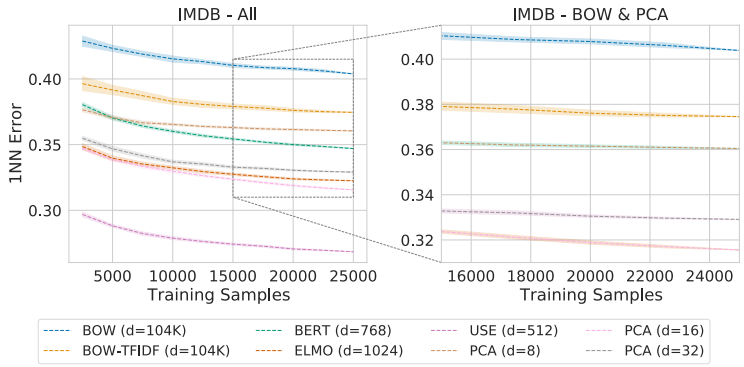


Figure 3.10: Impact of the dimension on IMDB using all involved transformations **(Left)**, and PCA-based transformation only **(Right)**.

Including the norm into the analysis of computer vision datasets, we see some marginal improvements, although on already high correlation values, which is also aligned with our theoretical analysis.

Convergence Plots We provide convergence plots for an interesting subset of the datasets (CIFAR100 and IMDB) and transformations in Figures 3.9 and 3.10. We remark that on both plots the transformations that achieve the best possible convergence in the finite sample regime do not have the lowest dimension. Furthermore, the starting point of the convergence lines for such transformations is typically much lower than the starting point of standard dimension-reduction techniques such as PCA. Having access to much more (ideally infinitely many) training samples would result in every line converging to the final, irreducible-bias term per transformation.

Empirical Results for $k > 1$ In addition to the experiments presented in the previous section, we performed experiments for $k > 1$ over all datasets and all embeddings. All the results for $k > 1$ confirm the theoretical findings, whilst for some k they offer an improvement in the linear correlation (e.g., on SST2 one can get CCA score up to 0.97 when $k = 8$, whereas in Figure 3.8 we report 0.917, for $k = 1$). In order to simplify the exposition, we opted for $k = 1$ and MSE instead of $MSE^{1/4}$ since it already serves the purpose of showing that MSE is important on its own, whereas including the smoothness assumption through the norm further improves the correlation.

It is well known that one can choose the hyper-parameter k to reach the best possible convergence in the finite data regime depending on the dataset. We investigate this with respect to transformations by showing that different transformations on the same dataset might have dif-

ferent optimal choices for k. This tradeoff for a fixed dataset is not clearly visible in the main Theorem 3.5.1 due to the usage of $O(\cdot)$ notation, hiding the constants. However, by exploring the proof outline and analyzing Equation A.1.5, one realizes that the upper bound of J_1 is dependent on the posterior in the transformed feature space, which might change for a fixed input dataset.

3.5.4 Discussion

One could examine the tightness of our theoretical bounds by constructing a toy dataset with a known true posterior probability and a set of transformations. However, any such transformation should either be an identity (if the toy dataset already has the desired property), or carefully constructed for this purpose only. In this section we opted for not constructing a single transformation ourselves, as our main goal is to bridge the gap between the real-world applications (e.g. pre-trained embeddings) with the theory of nearest neighbors. For example, even Lemma 3.5.4, which establishes the sharpness of the safety bound, works for any transformation. For that purpose, our probabilistic Lipschitz condition is as weak as possible. To this end, studying CCA of $k^{-1/2}$, $\|w\|(k/n)^{1/d}$ and $MSE^{1/4}$ (or some other exponent under stronger assumptions) for $k > 1$, could provide another angle for understanding the tightness of the bound, which is out of the scope of this thesis.

3.6 Building Ease.ML/Snoopy

As stated so far in this chapter, we set the *1NN*-based estimator on top of a feature transformation as our default choice for `Ease.ML/Snoopy`. For a fixed transformation f, and n-samples, it is defined by

$$\widehat{R}_{f(X),n} = \frac{(R_{f(X)})_{n,1}}{1 + \sqrt{1 - \frac{C(R_{f(X)})_{n,1}}{C-1}}}. \tag{3.6.1}$$

3.6.1 System Design

We next present the design of `Ease.ML/Snoopy`. A high-level overview of the workflow of our system is given in Figure 3.11.

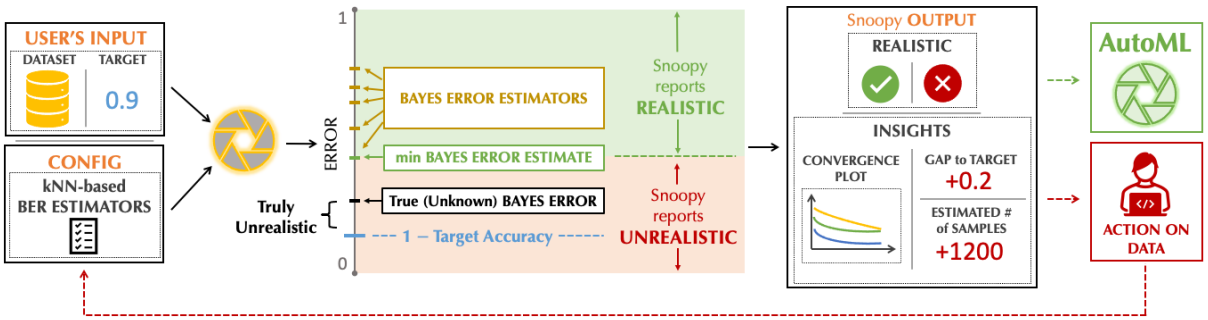


Figure 3.11: Overview of `Ease.ML/Snoopy`: Given user's input in terms of a dataset and a target, the system consults various Bayes error estimators (yellow ticks), aggregates them by taking the minimum (green tick) and outputs its belief whether the target is realistic, together with the insights in terms of convergence plots, gap towards target accuracy, and estimated number of additional samples needed to reach the target accuracy.

3.6.1.1 Functionality

`Ease.ML/Snoopy` interacts with users in a simple way. The user provides an input dataset that is representative for the classification task at hand, along with a target accuracy α_{target}. The system then estimates the "highest possible accuracy" that an ML model can achieve, and outputs a binary signal — `REALISTIC`, if the system deducts that this target accuracy is achievable; `UNREALISTIC`, otherwise. We note that `Ease.ML/Snoopy` does not provide a model that can achieve that target, only its *belief* on whether the target is achievable, using an *inexpensive* process. Furthermore, the goal of `Ease.ML/Snoopy` is *not* to provide a perfect answer on feasibility, but to give information that can guide and help with the decision-making process — the signal provided by the system may as well be wrong, as we will discuss later.

3.6.1.2 Interaction Model

The binary signal of `Ease.ML/Snoopy` given to a user is often correct, but not always. We now dive into the user's and `Ease.ML/Snoopy`'s interaction upon receiving the signal.

The Case When `Ease.ML/Snoopy` Reports `REALISTIC`: In general, one should trust the system's output when it reports the target to be realistic, and proceed with running AutoML. We note that wrongly reporting realistic can be a very costly mistake which any feasibility system should try to avoid. In theory, our system could also be wrong in that fashion, due to (1) a lower bound estimate based on the 1NN estimator by Cover and Hart (1967) that is known to be not

always tight, or (2) the fact that the estimators are predicting asymptotic values. However, as presented in the next section, we construct our estimator in a theoretically justified way that aims at reducing such mistakes and in our experiments we do not observe this behavior. Even if this were the case, we expect (2) to be the dominating reason, in which case gathering more data for the task at hand and running AutoML on this larger dataset might very well confirm the system's prediction.

The Case When Ease.ML/Snoopy Reports UNREALISTIC: In this case, our experiments showed that the system's output is also trustworthy, but with more caution. Under reasonable computational resources[5], Ease.ML/Snoopy is often correct in preventing unrealistic expectations for a varying amount of both synthetic and natural label noise. Nonetheless, there are two possible reasons for making wrong predictions in this manner: (1) either the data is not representative enough for the task (i.e., users might need to acquire more data), or (2) the transformations applied in order to reduce the feature dimension, or to bring raw features into a numerical format in the first place (e.g., from text), increased the BER. We note that (1) and (2) are complementary to each other. Even though estimating the BER on raw features (if applicable) prevents (2) from happening, having "better" transformations can lower the number of samples required to accurately estimate the BER.

In an ideal world, one could rule out (1) by checking whether the BER estimator converged on the given number of samples. That is why Ease.ML/Snoopy provides insights in terms of convergence plots and finite-sample extrapolation numbers to help users understand the *source* of predicting UNREALISTIC, and increase the confidence in the system's prediction.

3.6.1.3 Data Quality Issues and the BER

The power of the BER, and the reason that Ease.ML/Snoopy focuses on estimating this quantity, is that it provides a link connecting data quality to the performance of (best possible) ML models. This link can be made more explicit, even in closed form, if we assume some noise model. Here we take one of the most prominent source of data quality issues, label noise, as an example, and illustrate this connection via a novel theoretical analysis.

[5]For instance, reproducing the state-of-the-art model performance for well-established benchmark datasets is often a highly non-trivial task.

Noise Model We focus on a standard noise model for label noise: class-dependent label noise Wei et al. (2022). We assume that we are given a noisy random variable Y_ρ through a transition matrix t, defined as

$$t_{\widetilde{y},y} := \mathbb{P}(Y_\rho = \widetilde{y} \,|Y = y, X = x) = \mathbb{P}(Y_\rho = \widetilde{y} \,|Y = y), \tag{3.6.2}$$

where the equality follows from the assumption that we are in the class-dependent label noise scenario, rather than in instance-dependent one. One can think of $\rho(y) = 1 - t_{y,y}$ to be the fraction of class y that gets flipped. Let $y_x := \arg\max_{y \in \mathcal{Y}} p(Y = y|x)$. We further assume that $y_x = \arg\max_{y \in \mathcal{Y}} p_\rho(Y_\rho = y|x)$, meaning that the maximal label per sample x is preserved after flipping, albeit possibly with lower probability (which then increases the BER). Our main result is the following theorem.

Theorem 3.6.1. *Let Y_ρ be a random variable taking values in $\mathcal{Y}$ that satisfies (3.6.2). Then*

$$R^*_{X,Y_\rho} = R^*_{X,Y} + \mathbb{E}_X[\rho(y_x)p(y_x|x)] - \sum_{y \neq y_x} \mathbb{E}_X\left[t_{y_x,y}p(y|x)\right]. \tag{3.6.3}$$

One can prove Theorem 3.6.1 using the law of total expectation, together with careful manoeuvring of the terms that involve the elements of the transition matrix. We provide the full proof and and additional discussion in Appendix A.2. Setting $\rho(y) = \rho \cdot (1 - 1/C)$, for all $y \in \mathcal{Y}$, and $t_{y,y'} = \rho/C$, for all $y' \neq y$, recovers Lemma 3.4.1, and one can further deduct the following valid bounds on the evolution of the BER:

$$(1 - s_{X,Y}) \min_y \rho(y) - s_{X,Y} \max_{y,y':y \neq y'} t_{y,y'} \leq R^*_{X,Y_\rho} \leq s_{X,Y} + \max_y \rho(y),$$

where $s_{X,Y}$ denotes the error of state-of-the-art model.

Other Data Quality Dimensions Whilst we assume that the BER for zero label noise is typically small, it does not have to be equal to zero (c.f., Renggli et al. (2021a) for examples). Nonetheless, by estimating the BER, we implicitly quantify the data quality issues along *all* dimensions (e.g., missing features, or combinations of feature and label noise). Deriving alternative noise models to theoretically and empirically disentangling these factors is a challenging task and left as future work.

3.6.2 Implementation

The core component of `Ease.ML/Snoopy` is a BER estimator, which estimates the irreducible error of a given task. The key design decision of `Ease.ML/Snoopy` is to *consult a collection*

of BER estimators and aggregate them in a meaningful way. More precisely, for a collection of feature transformations $\mathcal{F}$, e.g., publicly available pre-trained feature transformations (or last-layer representations of pre-trained neural networks) on platforms like TensorFlow Hub, PyTorch Hub, and HuggingFace Transformers, we define our main estimator of the BER (on n samples) using Equation 3.6.1 by

$$\widehat{R} = \min_{f \in \mathcal{F}} \widehat{R}_{f(X),n}.$$

Finally, the system's output is

$$\begin{aligned} \texttt{REALISTIC}, &\quad \text{if } \widehat{R} \leq 1 - \alpha_{\text{target}}, \\ \texttt{UNREALISTIC}, &\quad \text{otherwise.} \end{aligned}$$

3.6.2.1 "Just a Lightweight AutoML System?"

At first glance, our system might seem like a "lightweight AutoML system," which runs a collection of fast models (e.g., kNN classifiers) and takes the minimum to get the best possible classifier accuracy. We emphasize the difference — the accuracy of an AutoML system always corresponds to a concrete ML model that can achieve this accuracy; however, a BER estimator does *not* provide this concrete model. That is, Ease.ML/Snoopy does *not* construct a model that can achieve $\widehat{R}$. This key difference between AutoML and feasibility study makes the latter inherently more computationally efficient, with almost instantaneous re-running, which we will further illustrate with experiments in Section 3.6.4.

3.6.2.2 Theoretical Analysis

Given a collection of 1NN-based BER estimators over feature transformations, Ease.ML/Snoopy aggregates them by taking the minimum. This seemingly simple aggregation rule is far from trivial, raising obvious questions — *Why can we aggregate BER estimators by taking the minimum? When will this estimator work well and when will it not?*

In order to mathematically quantify different regimes, we need a few simple definitions, which we illustrate in Appendix A.3 of the supplementary material. We define the *asymptotic tightness* of our estimator for a fixed transformation f as

$$\Delta_f = R^*_{f(X)} - \lim_{n \to \infty} \widehat{R}_{f(X),n}. \tag{3.6.4}$$

Equation 3.2.3 implies $\Delta_f \geq 0$. We define the corresponding *transformation bias* by

$$\delta_f = R^*_{f(X)} - R^*_X, \tag{3.6.5}$$

with $\delta_f \geq 0$. Finally, the *n-sample gap (of the estimator)* is given by

$$\gamma_{f,n} = \widehat{R}_{f(X),n} - \lim_{n\to\infty} \widehat{R}_{f(X),n}, \tag{3.6.6}$$

with $\gamma_{f,n} \geq 0$ in expectation.

The fundamental challenge lies in the fact that none of the three quantities above can be derived in practice: Δ_f is dependent on the underlying unknown distribution, δ_f is intractable for complex neural networks, and $\gamma_{f,n}$ relies on the convergence of the estimator, which requires the number of samples to be exponential in the input dimension (Snapp and Xu, 1996), making it impossible to generalize to representations on real-world datasets. Nevertheless, the connection between the quantities, together with the empirical analysis from Section 3.4.2 and Section 3.6.4, allows us to define meaningful regimes next.

When is $\widehat{R}$ optimal? In other words, when does the transformation that yields the minimum outperform all the others? A sufficient condition is given by

$$\forall f \in \mathcal{F}: \quad \delta_f + \gamma_{f,n} - \Delta_f \geq 0. \tag{3.6.7}$$

We prove the condition to be sufficient in Appendix A.3. If the sum of finite-sample gap and transformation bias (i.e., the normalized constants of the second and third terms in Equation 3.5.3) is larger than the asymptotic tightness of the estimator (i.e., the normalized constants of the first term in Equation 3.5.3) for all transformations, then all estimators yield a number larger than the true BER, and therefore the minimum can be taken. Intuitively, this means that all the curves in the convergence plot are above the true BER. We note that this has to include the identity transformation, where there is no transformation bias. If Condition 3.6.7 holds, $\widehat{R}$ *will not underestimate the BER*. The above trivially holds if for all $f \in \mathcal{F}$ one has $\Delta_f = 0$. Note that any classifier accuracy (non-scaled, to be used as proxy) also trivially falls into this regime, although it is usually worse than $\widehat{R}$. Furthermore, the system is guaranteed to not predict YES when the target is unreachable, thus avoiding costly mistakes. If the system wrongly predicts UNREALISTIC, it is guaranteed that its predicted error is off by at most $\delta_f + \gamma_{f,n} - \Delta_f$.

We remark that all empirical evidence in Section 3.6.4 and in the evaluation of `FeeBee` in Section 3.4.2 suggests that we are in this regime for reasonable label noise (e.g., less than 80%) on a wide range of datasets and transformations.

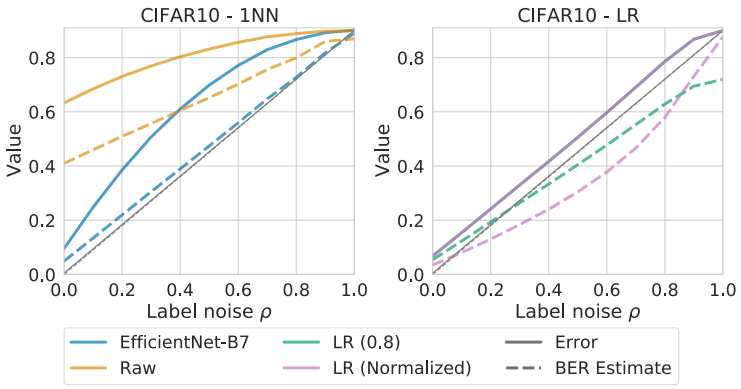


Figure 3.12: Theoretical justifications: (Left) 1NN error and its estimator values for raw features and the best transformation. (Right) Scaling down the logistic regression error on the best transformation and normalizing it by plugging it into the 1NN estimator of Equations 3.6.1.

What if $\widehat{R}$ is not optimal? We distinguish two different cases in this regime. In the first one, we suppose that the suggested estimator $\widehat{R}_{X,\infty}$ of Cover and Hart (1967) on the *raw* features performs well in the asymptotic regime, i.e. that Δ_{id} is small. In that case, a sufficient condition for $\widehat{R}$ to be at least as good as $\widehat{R}_{X,\infty}$ is $\Delta_f \leq \Delta_{id}$, for all $f \in \mathcal{F}$. Intuitively, this states that if all transformations do not increase the asymptotic tightness of the estimator by transforming the underlying probability distribution with respect to the raw distribution, taking the minimum over all transformations is no worse than running the estimator with 1NN on infinite samples. This condition can be seen empirically by inspecting the linear shape of the 1NN-based BER estimator values with increasing label noise for different transformations (c.f., Figure 3.12 on the left). One could weaken the condition for the finite-sample regime, resulting in a sufficient condition for $\widehat{R}$ to perform better than $\widehat{R}_{X,\infty}$:

$$\forall f \in \mathcal{F}: \quad \delta_f + \gamma_{f,n} - \Delta_f + \Delta_{id} \geq 0. \tag{3.6.8}$$

For the second case, when $\widehat{R}$ performs poorly, we ask: *What is the worst-case error of under-estimation?* Using the fact that 1NN error is trivially above the true BER (c.f., left inequality of Equation 3.5.2), we can bound the difference of the 1NN-Based estimator value. In fact, the estimator value is at most the scaling factor of Equation 3.6.1 away from the true BER (i.e., $1/2$ for a binary classification problem). However, our analysis and empirical verification reveals that our estimator of choice rarely ends up in the worst-case scenario. In fact, $\widehat{R}$ is usually the optimal choice and, when it is not, we often end up in the regime in which $\widehat{R}_{X,\infty}$ already performs well and $\widehat{R}$ outperforms it.

Scaling classifiers other than 1NN Since the worst-case error holds for scaling down any classifier accuracy, one could be tempted to use a scaled version (e.g., by a constant factor or by plugging the error value into the 1NN-based estimator of Equation 3.6.1) of other classifiers as a proxy. Contrary to the 1NN-based estimators, it is easy to show that for many datasets, any scaled version of a proxy model accuracy quickly falls into this worst-case regime (c.f., Figures 3.12 and 3.5 on the right), supporting the challenges of the Strawman outline at the beginning of this chapter.

3.6.2.3 Additional Guidance

To support users of `Ease.ML/Snoopy` in deciding whether to "trust" the output of the system, regardless of the outcome, additional information is provided. It comes in the form of (a) the estimated BER and, thus, the *gap* between the projected accuracy and the target accuracy, (b) the convergence plots indicating the estimated BER value with respect to increased number of training samples over all deployed BER estimators (as illustrated in Figure 3.11), and (c) an additional estimate of the required number of additional samples to reach the target accuracy for the minimal transformation. Such an estimate is fairly non-trivial. Although Snapp and Xu (1996) suggest how to approximate the kNN error by fitting a parametrized function to sampled data, the number of samples required to attain high confidence and accuracy is exponential in the feature dimension. This method is thus not practical for either finite-sample extrapolation, or estimating the BER (c.f., Section 3.4.2). Instead, to support users of `Ease.ML/Snoopy` beyond purely visual aids, we approximate the estimate based on the 1NN error using a simple log-linear function (Hashimoto, 2021)

$$\log\left((R_X)_{n,k}\right) \approx -\alpha \log(n) + C, \tag{3.6.9}$$

for two positive constants α and C. The idea of approximating the error is motivated by recent observations of scaling laws across different deep learning modalities (Kaplan et al., 2020; Rosenfeld et al., 2020). Notice that Equation 3.6.9 should only be used to extrapolate the convergence for a small number of additional data points. The function (i.e., the exponential of the righthand side of Equation 3.6.9) is known to converge to 0, implying that regardless of the label noise or true BER, it will always underestimate the BER for a too large number of samples. We show the benefits and failures of using this approximation in Section 3.6.4.

3.6.3 System Optimizations

The suggested and theoretically motivated estimator from the previous section relies on the 1NN classifier being evaluated on a possibly large collection of publicly available *pre-trained* feature transformations. We present optimizations that improve the performance, making it more practical and scalable. Note that we use the same optimization later in Chapter 4, where we focus more on the implementation for generic use-cases.

Algorithm There are five computational steps involved:

(i) Take user's dataset with n samples: features $X_1, X_2, \ldots, X_n$ and labels $Y_1, Y_2, \ldots, Y_n$.

(ii) For pre-defined m transformations $\mathcal{F} = \{f_1, f_2, \ldots, f_m\}$, calculate the corresponding features for every sample in the dataset by applying all the transformations in $\mathcal{F}$.

(iii) For each feature transformation $j \in [m]$, calculate the 1NN classifier error $R_j = (R_{f_j(X)})_{n,1}$ on the transformed features $f_j(X_i)$, for all samples $i \in [n]$ and the original labels.

(iv) Based on the 1NN classifier error, derive the lower-bound estimate for each transformation $\widehat{R}_{f_j(X),n}$ by applying Equation 3.6.1.

(v) Report the overall estimate by taking min

$$\widehat{R} = \min_{j \in [m]} \widehat{R}_{f_j(X),n}.$$

Note that the dataset is split into training samples and test samples. The test set is only used to estimate the accuracy of the classifier and is typically orders of magnitude smaller than the training set. The quality of `Ease.ML/Snoopy` depends heavily on the list of feature transformations that are fed into it. Since we take the minimum over all transformations in $\mathcal{F}$, increasing the size of the set only *improves* the estimator. On the downside, an efficient implementation of such a system is by no means trivial with an ever-increasing number of (publicly) available transformations.

Computational Bottleneck When analyzing the previously defined algorithm, we realize that the major computational bottleneck comes from transforming the features. Especially when having large pre-trained networks as feature extractors, running inference on large datasets, in order to get the embeddings, can be very time-consuming and result in running times orders of magnitude larger than the sole computation of the 1NN classifier accuracy. More concretely, given a dataset with n samples and m feature transformations, the worst case complexity is $O(mn)$, which highlights the importance of providing an efficient version of the algorithm.

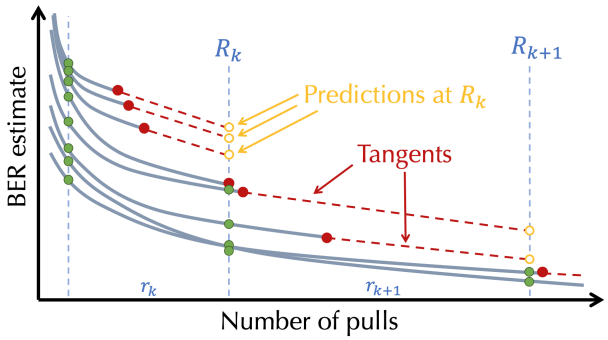


Figure 3.13: Improved Successive-Halving: At each point for each convergence curve we construct a tangent and check whether there are more than half of the remaining curves that are better than the tangent.

Multi-armed Bandit Approach Inspired by ideas for efficient implementation of the nearest-neighbor search on hardware accelerators Johnson et al. (2019), running inference on all the training data for all feature transformations simultaneously is not necessary. Rather, we define a streamed version of our algorithm by splitting the steps (ii) to (iv) into iterations of fixed batch size per transformation. This new formulation can directly be mapped to a *non-stochastic best arm identification* problem, where each arm represents a transformation. The successive-halving algorithm Jamieson and Talwalkar (2016), which is invoked as a subroutine inside the popular Hyperband algorithm Li et al. (2017), is designed to solve this problem efficiently. In a nutshell, we can summarize the idea of successive-halving as follows: Uniformly allocate a fixed initial budget across all transformations and evaluate their performance afterwards. Keep only the better half of the transformations, and repeat this until a single transformation remains. The full algorithm is explained in the supplementary material in Appendix B.3.

Improved Successive-Halving We develop a variant of successive-halving that further improves the performance. The main idea comes from observing the *convergence curve* of a kNN classifier. We know that under some mild assumptions, the kNN error decreases as a function of $n^{-2/d}$, where n is the number of samples Snapp et al. (1991). Therefore, we can assume that the convergence curve is decreasing and convex. This allows us to predict a simple lower bound for the convergence curve at the end of each step – using the tangent through the curve at the last known point, as illustrated in Figure 3.13. If the tangent line at the end point is worse than half of the remaining curves at the current point, the curve will not proceed to the next round. To simplify the implementation, we approximate the tangent by a line through the two last-known values of the convergence curve and develop a variant of successive-halving that uses this as a

stopping condition. An important property of our improvement is that the remaining transformations after each step are the same as the ones from the original successive-halving, which implies that all theoretical guarantees of successive-halving still hold.

Parameters of Successive-Halving We eliminate the dependency on the initial budget by implementing the *doubling-trick* (cf. Section 3 in Jamieson and Talwalkar (2016)). The batch size of the iterations has a direct impact on the performance and speedup of the algorithm. This is linked to properties of the underlying hardware and the fact that approximating the tangent for points that are further apart becomes less accurate. Hence, we treat the batch size as a single hyper-parameter, which we tune for all transformations and datasets.

Efficient Incremental Execution For the specific scenario of incrementally cleaning labels until a target accuracy is reachable, we provide a simple yet effective optimization that enables re-running `Ease.ML/Snoopy` almost instantly. After its initial execution, the system keeps track of the label of a single sample per test point – its nearest neighbor. As cleaning labels of test or training samples does not change the nearest neighbor, calculating the 1NN accuracy after cleaning an arbitrary number of training or test samples can be performed by iterating over the test set exactly once, thus, providing real-time feedback.

3.6.4 Experimental Results

We now present the results of our empirical evaluation by describing the benefits of performing a feasibility study in general, and using the binary output of `Ease.ML/Snoopy` over other baselines. We focus on a specific use-case scenario motivated in the introduction. We also show how the additional guidance can increase trust in the binary signal. We then analyze the generalization properties of our system on certain vision tasks and conclude this section by performing a detailed performance analysis of `Ease.ML/Snoopy`.[6]

3.6.4.1 Experimental Setup

We follow the same experimental setup as outlined in Section 3.3 with some extensions, which we outline next.

[6]The code of `Ease.ML/Snoopy` is available via `https://github.com/easeml/snoopy`.

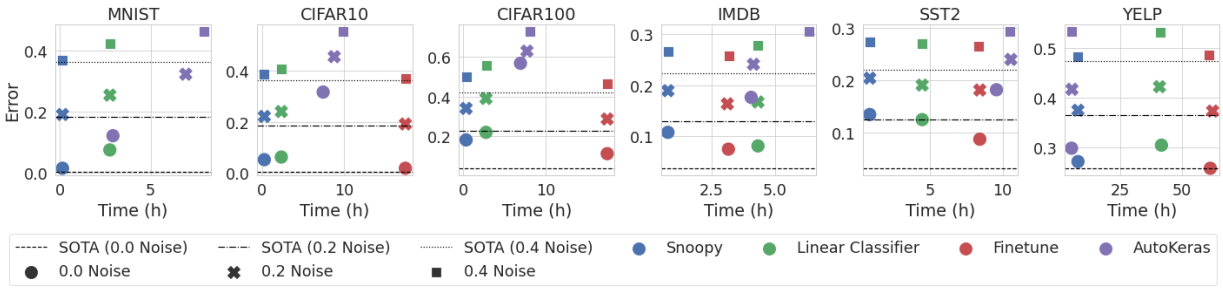


Figure 3.14: Error Estimations vs. Time - For feasibility study baselines on three synthetics noise levels. The dashed horizontal lines represent the expected increase of the SOTA using Lemma 3.4.1.

Naturally Noisy Datasets We mainly focus our study on datasets with noisy labels. The ML community usually works on high-quality, noise-free benchmark datasets. As an exception, Wei et al. (2022) published different noisy variants of the popular CIFAR datasets, called CIFAR-N. The noise levels vary between 10% and 40% (c.f., Table B.2). The datasets are provided with their noise transition matrix, allowing us to use the bounds derived from Theorem 3.6.1. Supported by our theoretical understanding of the impact of label noise on the BER and its evolution, we also synthetically inject uniform label noise for 20% and 40% of the label into all six datasets from Table B.1.

Settings of `Ease.ML/Snoopy` When running `Ease.ML/Snoopy`, we define the time needed to reach the lowest 1NN error across all embeddings based on multiple independent runs. These runtimes include the 1NN computation and running inference on a single GPU, with the latter being the most costly part, particularly for large NLP models. When re-running `Ease.ML/Snoopy` after having cleaned a fixed portion of the labels (set to 1% of the dataset size) we assume that the "best" embedding did not change and, therefore, no additional inference needs to be executed. This assumptions holds for all evaluated datasets.

We compare with a diverse set of baselines that estimate the BER: (i) training a logistic regression (LR) model on top of all pre-trained transformations, (ii) running AutoKeras, and (iii) fine-tuning a state-of-the-art (SOTA) foundation model Bommasani et al. (2021) for each data modality.

Baseline 1: LR Models As mentioned before, when training the logistic regression models we assume that the representations for all the training and test samples are calculated in advance exactly once. After having cleaned the same fixed portion of labels (i.e., 1% of test and train

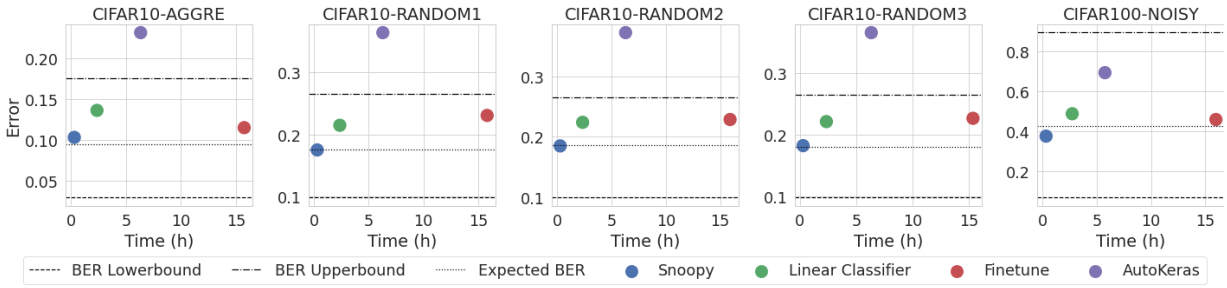


Figure 3.15: Error Estimations vs. Time - For feasibility study baselines on multiple real noisy variant of CIFAR10 and CIFAR100 (c.f., Table B.2). The dashed horizontal lines represent bounds and the expected increase of the SOTA using Theorem 3.6.1.

samples), re-training the LR models does not require any inference. We train all LR models on a single GPU using SGD with a momentum of 0.9, a mini-batch size of 64 and 20 epochs. We select the minimal test accuracy achieved over all combinations of learning rate in $\{0.001, 0.01, 0.1\}$ and L_2 regularization values in $\{0.0, 0.001, 0.01\}$. We calculate the average time needed to train a LR based on the best transformation, without label noise, on all possible hyper-parameters. The hyper-parameter search was conducted 5 independent times for any value of randomly injected label noise.

Baseline 2: AutoKeras To mimic the use of an AutoML system without any prior dataset-dependent knowledge, we run AutoKeras on a single GPU with the standard parameters of a maximum of 100 epochs and 2 trials on top of all datasets. We report the mean of 5 independent executions in terms of times and accuracy, noting little variance amongst the results.

Baseline 3: Finetune The goal of this baseline is to replicate the SOTA values achieved for all datasets. We remark that this baseline is equipped with a strong prior knowledge which is usually unavailable for performing a cheap feasibility study and it only serves as a reference point. Unfortunately, reproducing the exact SOTA values was not possible for any of the dataset involved in the study, which is mainly due to computational constraints and the lack of publicly available reproducible code. We therefore perform our best, mostly manual, efforts to train a model on the original non-corrupted data. For multi-channel vision tasks (i.e., CIFAR10 and CIFAR100, and its noisy variants), we fine-tune EfficientNet-B4 using the proposed set of hyper-parameters Tan and Le (2019), whereas for NLP tasks, we fine-tune BERT-Base with 3 different learning rates and for 3 epochs Devlin et al. (2018), using a maximal sequence length of 512, batch size of 6 and the Adam optimizer.

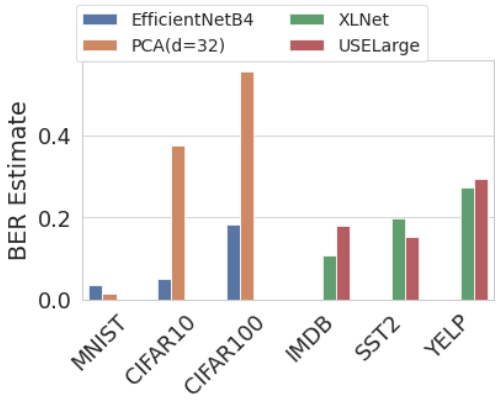


Figure 3.16: Impact of fixing a single feature transformation.

3.6.4.2 Evaluation of BER Estimations

We first evaluate `Ease.ML/Snoopy` by comparing its BER estimation of the best achievable accuracy with other baselines. In the next section, we will compare how this benefits an end-to-end scenario.

`Ease.ML/Snoopy` vs. Baselines on Synthetic Noise In Figure 3.14 we present our main findings on three levels of realistically injected label noise — 0%, 20% and 40%, which we visualize by adding the increase of the SOTA (at the time of writing) in expectation as horizontal lines to indicate a proxy of the ground truth BER error. We see that `Ease.ML/Snoopy` is much faster than all other methods, often by orders of magnitude. The only exception is YELP in which running over large models (e.g., GPT2 or XLNet) slows down `Ease.ML/Snoopy` in a fashion comparable to AutoKeras, whilst still outperforming it in terms of the estimated accuracy. It also produces BER estimations that are *comparable, if not better* than all other approaches. In fact, it is often better than both LR and, particularly, AutoKeras. It is only slightly worse than the LR classifier on text tasks for IMDB and SST2, while being orders of magnitude faster.

`Ease.ML/Snoopy` vs. Baselines on Real Noise In Figure 3.15 we run the same set of experiments for real noisy variants of CIFAR10 and CIFAR100 from (Wei et al., 2022). We realize that `Ease.ML/Snoopy` constantly outperforms all baselines both in terms of speed and estimation accuracy. When comparing the error values to the lower and upper bounds, we realize that whilst `Ease.ML/Snoopy` remains inside the bounds, there is a considerable gap between them. Nonetheless, `Ease.ML/Snoopy` produces estimates close to the expected increase of the SOTA using Theorem 3.6.1.

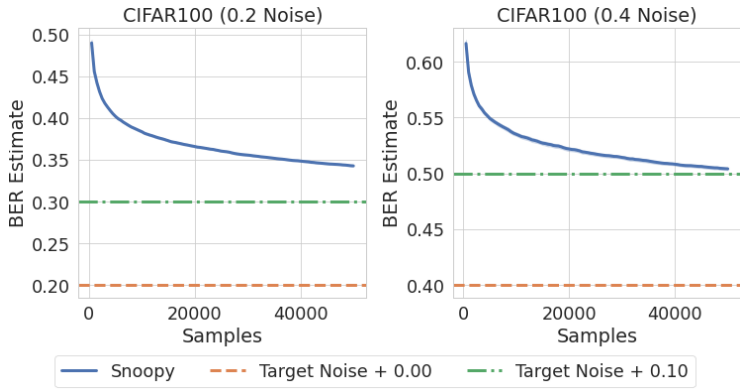


Figure 3.17: CIFAR100 - Different noise levels and targets.

Is Taking the Minimum Necessary? When analyzing the performance of the system with respect to the number of feature transformations, one might ask the question whether a single transformation always outperforms all the others and hence makes the selection of the minimal estimator obsolete. When conducting our experiments, we observed that selecting the *wrong* embedding can lead towards a large gap when compared to the optimal embedding, e.g., favoring the embedding USELarge over XLNet on IMDB doubles the gap of the estimated BER to the known SOTA value, whereas favoring XLNet over USELarge on SST2 increases the gap by 1.5×, making proper selection necessary (c.f., Figure 3.16).

3.6.4.3 Usefulness of the Additional Guidance

When evaluating the 1NN estimator accuracy for varying label noise, and its convergence under different feature transformations (c.f., Appendix B.4.1 in the supplementary material), we see that even the best transformations are constantly over-estimating the lower bound when increasing label noise, validating the key arguments for taking the minimum over all estimators. All the results indicate the median, 95% and 5% quantiles over multiple independent runs (i.e., 10 for YELP and 30 otherwise). We observe the presence of much more instability in SST2 when compared to other datasets, as seen in Figure B.6. This is not at all surprising since SST2 has a very small test set consisting of less than one thousand samples, as seen in Table B.1. This naturally results in higher variance and less confidence when estimating the 1NN classifier accuracy compared to the larger number of test samples for the other datasets.

Figure 3.17 illustrates a convergence plot for a fixed embedding (EfficientNet-B5) and the dataset CIFAR100 injected with 20% and 40% label noise respectively. The two target accuracies visualized by a horizontal line represent exactly the noise level, and the noise level plus 10%.

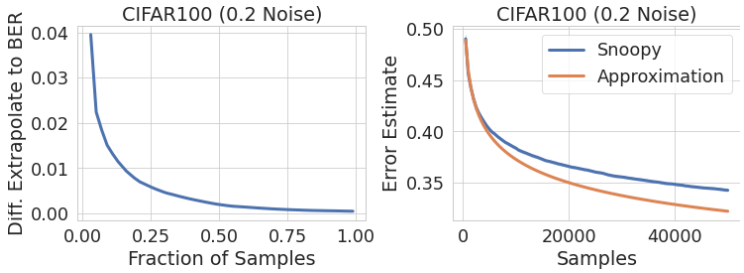


Figure 3.18: CIFAR100 - (Left) Extrapolation accuracy to the full dataset based on a fraction of the samples. (Right) Approximation based on 5% of the samples.

Note that the noise level is only reachable if the original BER of the dataset is zero. From the visualizations, the target of 0.5 on the dataset with 0.4 noise is highly likely. By using the approximation from Equation 3.6.9, we realize that less than 10K more samples should suffice to attain this accuracy. Conversely, for a target of 0.3 and 0.2 noise, although possibly realizable, the number of additional samples to verify the quality of the extrapolation is already more than 260K. Note that Equation 3.6.9 converges to zero, and therefore any target can be realizable. Targeting exactly the label noise for each of the datasets yields an extrapolated number of more than 16M and 84M, which both should be seen as not trustworthy approximations based on the much smaller number of available samples in the training set. This thus implies that the target accuracy is not achievable based on the given transformations and numbers of samples. To illustrate this fact, we subsample the low label noise dataset and the same embedding for a fixed fraction. We then extrapolate the achievable target for the full dataset (i.e., 50K training samples) and plot the difference between the extrapolated target and the true BER estimate in Figure 3.18 on the left. The right part of Figure 3.18 illustrates the extrapolation based on 5% of the samples.

3.6.4.4 End-to-end Use Case

How can we take advantage of `Ease.ML/Snoopy` *to help practical use cases?* In this section, we focus on a specific end-to-end use case of a feasibility study in which the user's task contains a target accuracy and a representative, but noisy dataset, which we simulate as described above. The goal is to reach the target accuracy. At each step, the user can perform one of the following three actions: (1) clean a portion of the labels, (2) train a high-accuracy model using AutoKeras or fine-tune a state-of-the-art pre-trained model, (3) perform a feasibility study by either using the cheap LR model or `Ease.ML/Snoopy`. Requiring the capability of *cleaning* labels, we only conduct these experiments on synthetic label noise, where we can simply *restore* the original

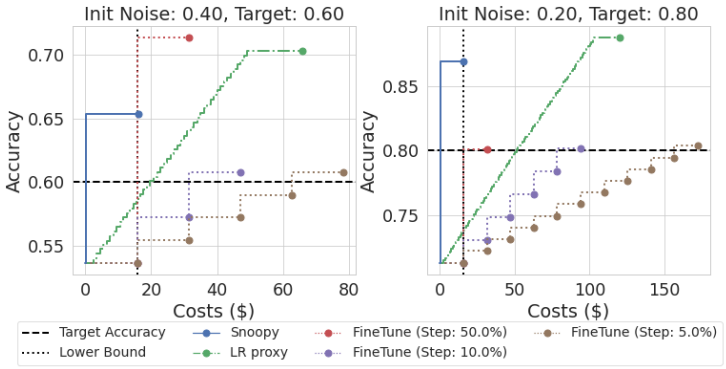


Figure 3.19: CIFAR100 - End-to-end use case, free labels.

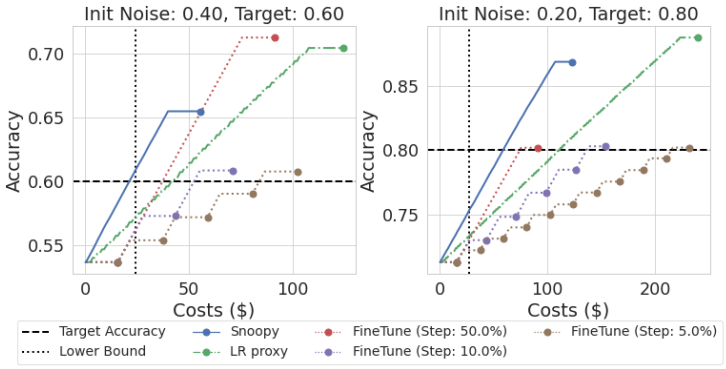


Figure 3.20: CIFAR100 - End-to-end use case, cheap labels.

label. All reported values represent the mean (accuracy and run-time) over at least 5 independent runs.

Different User Interaction Models We differentiate two main scenarios in our end-to-end experimental evaluation: (1) *without feasibility study* and (2) *with feasibility study*. Without a feasibility study, users will start an expensive, high-accuracy run (i.e., running the fine-tuning baseline) using the input data. If the achieved accuracy is below the desired target, users will clean a fixed portion of the data (1%, 5%, 10%, or 50%, which we call *steps*) and re-run the expensive training system. This is repeated until a model reaches the desired accuracy or all labels are cleaned. With a feasibility study, users alternate between running the feasibility study system and cleaning a portion of the data (set to 1% of the data) until the feasibility study returns a positive signal or all labels are cleaned. Finally, a single expensive training run is performed. The lower bound on computation is given by training the expensive model exactly once.

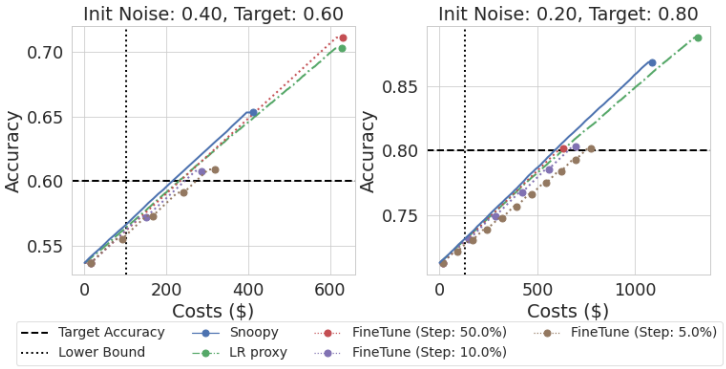


Figure 3.21: CIFAR100 - End-to-end use case, expensive labels.

Different Cost Scenarios We measure the cost in hypothetical "dollar price" for different regimes, depending on the *human-labeling* costs and on the *machine* costs. For the former, we define three scenarios: 'free', 'cheap' (0.002 dollars per label, resulting in 500 labels per dollar) and 'expensive' (0.02 dollars per label, resulting in 50 labels per dollar). For the latter, we fix the *machine* cost to 0.9$ per hour (the current cost of a single GPU Amazon EC2 instance).

Key Findings Due to the space constraints, we only present the results on CIFAR100 here and leave the rest to the supplementary material in Appendix B.4.2—we observe similar results on all datasets for a wide range of initial noise levels and target accuracies. We show the results in Figures 3.19, 3.20 and 3.21, for the 3 different cost setups described above (free, cheap, expensive), each over 2 values of the initial noise (0.40, 0.20) and, respectively, 2 target accuracies (0.60, 0.80). Each dot represents the result of one run of the expensive training process.

(I) Feasibility Study Helps. When comparing the costs of repetitively training an expensive model to those of using an efficient and accurate system that performs a feasibility study, such as `Ease.ML/Snoopy`, we see significant improvements across all results (c.f., blue vs. brown lines in Figure 3.19). Without a system that performs a feasibility study, users are facing a dilemma. On the one hand, if one does not train an expensive model frequently enough, it might clean up more labels than necessary to achieve the target accuracy, e.g., FineTune (step 50%), which makes it intense on the human-labeling costs. On the other hand, if one trains an expensive model too frequently, it becomes computationally expensive, wasting a lot of computation time. With a feasibility study, the user can balance these two factors better. As running low-cost proxy models is significantly cheaper than training an expensive model, the user can get feedback more frequently (having in mind the efficient incremental implementation described in Section 3.6.3). Finally, notice that when we enter the label-cost dominated regime (e.g., Figure 3.21), one seeks

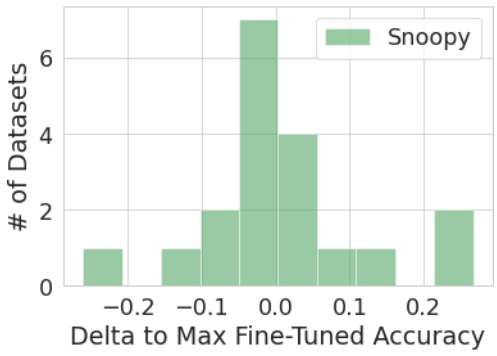


Figure 3.22: Ease.ML/Snoopy vs. Maximal fine-tune accuracies on VTAB (derived from the results in Chapter 4): Showing Ease.ML/Snoopy's generalization ability on small datasets and embeddings trained on different tasks.

at cleaning the minimum amount of labels necessary, ignoring the computational costs. Nevertheless, finding the right step size is critical, making it a difficult task.

(II) Ease.ML/Snoopy Outperforms Baselines. When comparing different estimators that can be used in a feasibility study, in most cases, Ease.ML/Snoopy is more effective compared to running a cheaper model such as LR, with its accuracy as a proxy. Ease.ML/Snoopy offers significant savings compared to LR when the labeling costs are high (c.f., Figure 3.21). The LR model will often be of a lower accuracy than an expensive approach; hence, it requires to clean more labels than necessary to reach the target. We note that there are cases (e.g., for IMDB) where the best LR model yields a lower error than the BER estimator used by Ease.ML/Snoopy (c.f., Figure B.10, expensive labels). In such cases, there exists a regime where the costs of using the LR proxy are comparable or superior to using Ease.ML/Snoopy despite being more expensive to compute. However, we see this as an exception and Figure 3.20 clearly show that the LR proxy is usually significantly more costly than using Ease.ML/Snoopy.

3.6.4.5 Generalization to Other Tasks

In this section we examine two potential limitations of our system, when deployed on a new task: (i) *its dependence on large datasets*, and (ii) *the necessity of having "good" pre-trained feature transformations for the given task*. For this, we use the results of Chapter 4 on the popular visual task adaptation benchmark (VTAB) (Zhai et al., 2019a) which is known to be a *diverse* collection of datasets (19 different tasks), each being *small* (1K training samples), and our collection of pre-trained transformations *does not contain* any trained on these datasets.

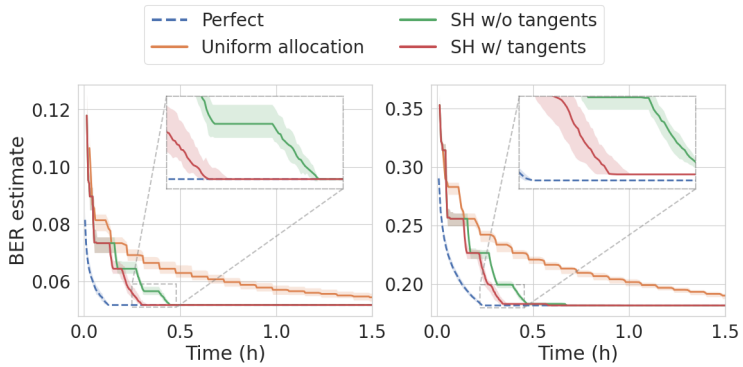


Figure 3.23: Different selection strategies over all transformations **(Left)** for CIFAR10, **(Right)** for CIFAR100.

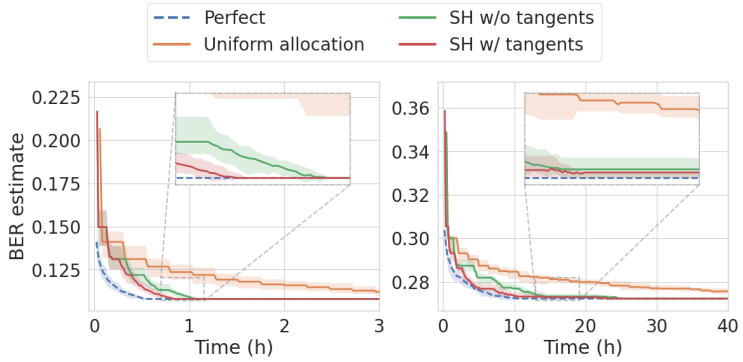


Figure 3.24: Different selection strategies over all transformations **(Left)** for IMDB, **(Right)** for YELP.

To validate that `Ease.ML/Snoopy` does not suffer from the above limitations, in Figure 3.22 we illustrate the difference between `Ease.ML/Snoopy`'s predictions and the best achieved post-fine-tune accuracies. We observe that on most datasets, `Ease.ML/Snoopy` produces a useful estimate of the fine-tune accuracy (except for some negative transfer results enabled by the low data regime). Even though this is sufficient to say that the currently available embeddings are supporting `Ease.ML/Snoopy`'s performance, we expect this figure to improve over time as more and better embeddings become publicly available.

3.6.4.6 Efficiency of `Ease.ML/Snoopy`

We saw that the gain of using `Ease.ML/Snoopy` comes from having an (i) *efficient estimator* of (ii) *high accuracy*. Those two requirements are naturally connected. While having access to more and "better" (pre-trained) transformations is key for getting a high accuracy of our estimator, it requires the implementation of our algorithm to scale with respect to the ever-increasing number of transformations.

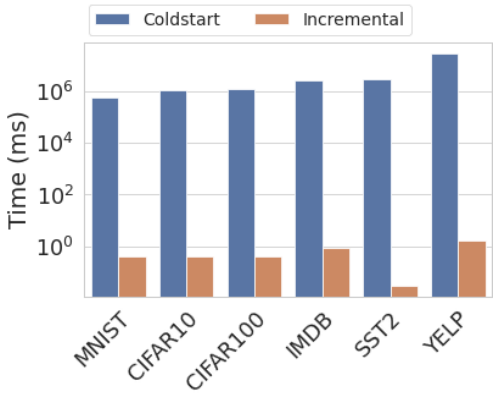


Figure 3.25: Incremental execution (log scale): On all datasets, rerunning `Ease.ML/Snoopy` incrementally is, unsurprisingly, by several orders of magnitude faster than running from scratch.

Runtime Analysis To showcase the importance of the *successive-halving* (SH) algorithm, with and without the *tangent method* presented in Section 3.6.3, we compare different strategies for deploying the 1NN estimator in Figures 3.23 and 3.24. The strategies are evaluated with respect to the runtime (averaged across multiple independent runs on a single Nvidia Titan Xp GPU) needed to reach an estimation within 1% of the *best* possible value using all the training samples. Running the estimator only on the transformation yielding the minimal result is referred to as the *perfect* strategy providing a lower bound, whereas we also test the *uniform allocation* baseline described in (Jamieson and Talwalkar, 2016). We report the runtime by selecting the best batch size out of 1%, 2%, or 5% of the training samples. We observe that running the entire feasibility study using `Ease.ML/Snoopy` on CIFAR100 on a single GPU takes slightly more than 16 minutes, whereas the largest examined NLP dataset YELP requires almost 8.5 hours, with a clear improvement of SH with the tangent method over the one without. Putting these numbers into context, fine-tuning EfficientNet-B4 on CIFAR100 on the same GPU with one set of hyper-parameters (out of the 56 suggested by the authors (Tan and Le, 2019)) requires almost 10 hours (without knowing whether other embeddings would perform better), whereas training large NLP models is usually only achievable in a reasonable time with several hundred accelerators Rasley et al. (2020).

Incremental Execution In Figure 3.25, not surprisingly, we see the benefit of using our system to quickly perform an incremental evaluation described in Section 3.6.3. Comparing it with running from scratch, we see that the incremental execution is faster by several orders of magnitude on all datasets.

3.7 Discussion

In this chapter we tackle on the challenging endeavor of estimating the *irreducible error* for the task definition and data acquisition process, originating from data quality issues. We focus on label noise, representing one of the most prominent source for non-zero irreducible error. The exploration of other aspects of poor data quality, such as noisy or incomplete features, are left as future work. We by no means provide a conclusive solution to prevent unrealistic or very costly endeavors of training ML models with finite data. Rather, we view our contribution as a first step towards a practical treatment of this problem, which is the key for enabling a systematic feasibility study for ML.

3.8 Summary

We started by introducing `FeeBee`, the first system to systematically evaluate lower and upper bound estimators of the BER on real-world data. By providing a thorough analysis of existing estimators using `FeeBee`, we observe that GHP and 1NN are consistently outperforming the other lower bound estimators whilst enabling easy hyper-parameter selection on a single point estimate. We also observe that currently there does not exist an estimator that is able to simultaneously outperform other estimators on both lower and upper bound. We are open-sourcing the framework which is easily extendable with any new BER estimator, and hope to further motivate and enable progress in building better and more practical methods in the future.

Then, we provided novel theoretical results aimed towards understanding the convergence behavior of kNN classifiers over transformations, bridging the existing gap between the current state of the theory and kNN applications used in practice. We provide a rigorous analysis of properties of transformations that are highly correlated with the performance of kNN classifiers, yielding explainable results of kNN classifiers over transformations. We believe that optimizing the upper bound presented here and extending results to classifiers other than logistic regression could form an interesting line of future research.

Finally, we presented `Ease.ML/Snoopy`, a novel system that provides the functionality of performing a systematic feasibility study for ML application development based on the theoretical finding of `FeeBee` and the convergence behavior of the kNN classifier. By consulting a range of estimators of the Bayes error and aggregating them in a theoretically justified way, `Ease.ML/Snoopy` suggests whether a predefined target accuracy is achievable. We demonstrate system optimizations that support the usability of `Ease.ML/Snoopy`, and scale with the increase in the number and diversity of available pre-trained embeddings in the future.

Chapter 4

SHiFT: An Efficient, Flexible Search Engine for Transfer Learning

Transfer learning (Weiss et al., 2016; Mensink et al., 2021) is an emerging paradigm of building machine learning applications, which can potentially have a profound impact on the architecture of today's machine learning systems and platforms. In a nutshell, transfer learning aims at training ML models with high quality without having to collect enormous datasets or spend a fortune on computation for training from scratch. Instead, models are first *pre-trained* on typically large and possibly private *upstream* datasets and are then made available to model repositories such as TF-Hub, PyTorch Hub, and Huggingface. Given a new dataset, a user picks *some* of these pre-trained models to fine-tune, which typically requires adding randomly initialized layers to parts of the pre-trained deep network, and tuning all the parameters using the limited amount of downstream data. Transfer learning has been successfully applied to many domains and tasks (Azizpour et al., 2015; Long et al., 2015; Ruder et al., 2019; Houlsby et al., 2019).

This process, despite being cheaper compared to training from scratch (i.e., with a fully ran-

The content of this chapter are largely based on the following large empirical study (1) Renggli et al. (2022a). Which Model to Transfer? Finding the Needle in the Growing Haystack. *CVPR*. A paper describing the system is currently under review and can be accessed via: (2) Renggli et al. (2022c). SHiFT: An Efficient, Flexible Search Engine for Transfer Learning. *Under Review*.

Table 4.1: Example search queries supported by `SHiFT` (* is proposed in this thesis), returning an ordering of registered models by their...

Q1: best reported upstream accuracy (Kornblith et al., 2019)
Q2: best nearest neighbour classifier accuracy (Meiseles and Rokach, 2020; Puigcerver et al., 2021)
Q3: best linear classifier accuracy (Bao et al., 2019; Nguyen et al., 2020; Tran et al., 2019)
Q4*: Q1 and Q3 (on models excluding the result of Q1)
Q5: highest (fine-tune) accuracy on the most similar task (Achille et al., 2019)

domly initialized network), still requires all parameters to be updated several times which can be computationally demanding. With the growing number of pre-trained models available online, it is computationally infeasible to fine-tune *all* models to find the one that performs best for a downstream task. As a result, a key defining component of a transfer learning application is a *model search strategy*, which provides cheaper ways to identify promising models to use. One challenge is that *different tasks might require very different search strategies* (Zhai et al., 2019b; Mensink et al., 2021) and Table 4.1 illustrates several popular ones.

Today, a user of these model repositories conducting transfer learning needs to manage these models and implement these search strategies all by themselves. Over the years, in the context of our Ease.ML (Karlaš et al., 2018; Li et al., 2018; Aguilar Melgar et al., 2021) effort, we have observed several challenges that our users face:

Challenge 1. Our first observation is that the amount of data and computation a user needs to perform quickly exceeds what most users, even competent software engineers, can handle. Some of the challenges are more on the engineering side — today's pre-trained models are scattered in different repositories, including HuggingFace, TF-Hub, and PyTorch Hub, lacking a unified abstraction. However, many others are technological — HuggingFace, TF-Hub, and PyTorch Hub contain 34,762, 1,198, and 48 models, respectively, and sum up to more than 100GB in size. Simply downloading all these models can take hours, not to mention running inference of all these models over a given dataset and implementing state-of-the-art search strategies. As a consequence, many users in our experience simply resort to only using the latest model — *this simply ignores the vast diversity of available models and, as we show Section 4.3, can lead to a significant quality gap.*

Challenge 2. The second challenge we observe is that today's data-centric development pattern of ML applications provides unique opportunities to speed up the model search process, which,

if left to the users, are quite hard to capture. In many cases, users will conduct iterations on the datasets—data cleaning, acquisition of both labels and features—and conduct model search for each of these datasets over time. Since these datasets are similar to each other, it is possible to save a significant amount of computation if we carefully design incremental maintenance strategies for different search queries — *all these opportunities to speed up model search are not being captured today by most, if not all, users.*

Challenge 3. The third challenge we observe is that model search is an active research area where new results are coming out quite frequently. Just in the last three years, researchers proposed at least eight new strategies (Kornblith et al., 2019; Achille et al., 2019; Bao et al., 2019; Nguyen et al., 2020; Meiseles and Rokach, 2020; Puigcerver et al., 2021; Deshpande et al., 2021). Having users to catch up with these latest developments can be tedious and potentially a huge waste of resources. *We are in a dire need for a unified framework that can be extended in a flexible way to support and automatically optimize newly proposed search strategies.*

All three challenges, in our opinion, lead to the same hypothesis:

We should shift the responsibility of querying and conducting model search over today's model repositories from individual users to a data management system, which can (1) provide a unified abstraction to connect all major model repositories, (2) provide a flexible, extendable way of specifying search strategies, (3) automatically optimize its execution, and (4) support efficient incremental execution.

4.1 Contributions

In this chapter, we start by providing a large-scale, systematic empirical study of existing model search strategies (i.e., the queries Q1-Q3 and Q5 defined in Table 4.1), by conducting the following steps:

1. We define and motivate the model search problem through a notion of regret. We conduct the first study of this problem in a realistic setting, by considering 19 downstream tasks on a heterogeneous set of 46 models grouped into 5 representative sets.
2. We highlight the dependence of the performance of each strategy on the constraints of the model pool, and show that, perhaps surprisingly, all strategies suffer a large regret on a significant fraction of downstream tasks.

3. As a consequence, we develop a hybrid approach (Q4) which generalizes across model pools as a practical alternative.

Then, we present SHiFT, one of the first data management systems for transfer learning — SHiFT is definitely *not* the first "model management system", an area that has attracted a lot of interest from the data management community (Vartak et al., 2016; Zaharia et al., 2018; Orr et al., 2021; Kumar et al., 2017; Schelter et al., 2018a); however, to our best knowledge, it is the first one that focuses on this specific workload, i.e., model search for transfer learning, within the context of model management systems. SHiFT aims at supporting efficiently diverse model search queries to find the (near-)optimal pre-trained model to use for fine-tuning a downstream transfer learning task. Our technical contributions for building SHiFT are as follows:

- **(C1) System Design.** We propose SHiFT-QL, a novel query language that can model all existing model search strategies that we are aware of and is extendable with respect to new models, model repositories, as well as potential new search strategies.
- **(C2) System Optimization.** We carefully studied several system optimization opportunities, both for a single query run and for incremental maintenance. **(C2.1)** For a single query run, we propose several optimizations to speed up the model search process, including a novel strategy based on successive halving (Jamieson and Talwalkar, 2016) to use resources more efficiently. **(C2.2)** Furthermore, we develop a cost model to *automatically* decide whether to apply different system optimizations, given a search query. **(C2.3)** SHiFT also efficiently supports incremental query execution, as required by the iterative nature of the model development process. This step is non-trivial, especially in the context of successive-halving. To this end, we propose a carefully designed incremental maintenance strategy.
- **(C3) Evaluation.** We evaluate SHiFT over a diverse set of queries across three datasets and 100 diverse models. We show that SHiFT outperforms a baseline implementation of the same search strategy by up to 1.57×, and in the incremental scenario up to 4.0× for 10% feature changes. Compared with fine-tuning all models without any search strategies, SHiFT can be up to 22.6×-45.0× faster.
- **(C4) Open Source.** We make SHiFT publicly available by open-sourcing it under `https://github.com/DS3Lab/shift`.

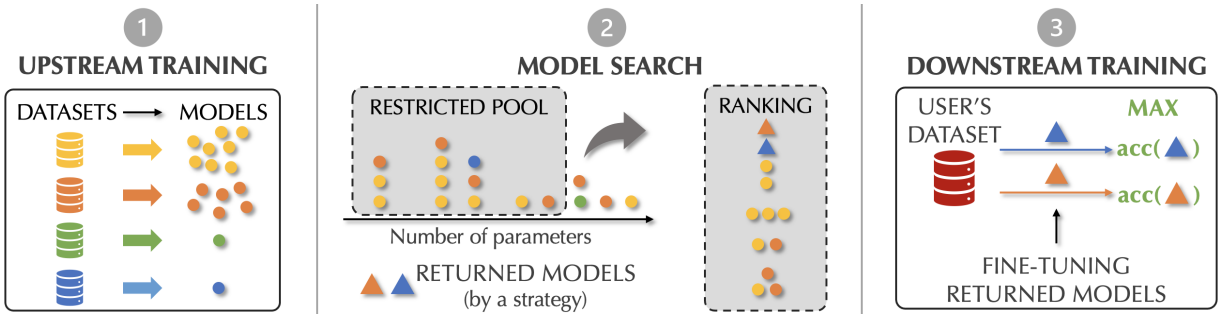


Figure 4.1: Three stages of a transfer learning setup: (1) Models are trained *upstream* for various architecture and datasets, (2) the models are restricted into a *pool*, and ranked by a search strategy, and (3) a subset of the models based on the ranking are fine-tuned on user's dataset in a downstream process. `SHiFT` focuses on supporting the middle stage (2) efficiently and in a flexible, generic way.

4.2 Background and Related Work

We start by describing the background and motivation for building `SHiFT`, enabling our users to make better use of today's available pre-trained models.

4.2.1 Transfer Learning

Transfer learning has been applied very successfully on popular deep learning modalities such as computer vision (CV) and natural language process (NLP) over the last few years (Pan and Yang, 2009; Tan et al., 2018; Weiss et al., 2016). Transfer learning is typically divided into three steps as illustrated in Figure 4.1: (1) An upstream, or pre-training part, where a machine learning model is trained using a well-established approach (e.g., randomly initialized weights and using mini-batch stochastic gradient descent as an optimizer) and an upstream dataset. (2) Having access to multiple such models trained on various datasets or model architectures, users pick a single or a subset of the available models, through some model search strategy, for the subsequent part. Note that users typically do not have access to the upstream datasets for this model search part. (3) The chosen models are fine-tuned in a downstream part on users' datasets. There are multiple strategies on how to fine-tune a pre-trained model, the most prominent being illustrated in Figure 4.2. The pre-trained model is split into two parts: a *feature extractor,* typically the entire network until the last layer, and the classification *head* consisting of the final linear classification layer. The new model is then a copy of the feature extractor (i.e., weights and architecture), and a new randomly initialized head. Replacing the pre-trained linear head by

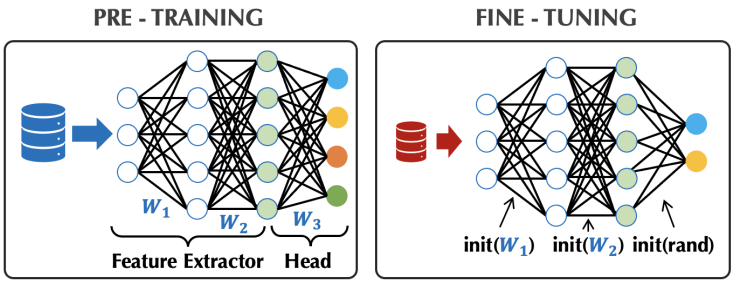


Figure 4.2: The difference between pre-training (left) and fine-tuning (right). The features stem from the last layer of a (pre-)trained model.

a new one is usually unavoidable, as the number of classes often changes from the upstream task to the downstream one. All the weights of the network are then refined for multiple iterations using an iterative optimization algorithm like stochastic gradient descent (SGD).

The benefits of this three-step transfer learning process over training from scratch (i.e., training a randomly initialized network using the downstream dataset only) are twofold: (A) Using transfer learning, one can fine-tune a very large and complex network (i.e., many parameters forming a highly non-convex space to optimize in) even for limited amount of downstream data. (B) The fine-tuning process requires a much smaller number of steps in the iterative optimization process. Both (A) and (B) ensure a good initial condition to start the fine-tune optimization process. They are enabled by the fact that much larger, sometimes private upstream datasets are used to train for many iterations. The choice of which pre-trained model to fine-tune has a high impact on the final accuracy one can expect (e.g., a sub-optimal pick in our experiments can lead to a 43% downstream accuracy gap). If we are not concerned with optimization compute and power, we could fine-tune all models and pick the best one afterwards. This brute-force approach is usually infeasible in practice given the amount of currently available pre-trained models, along with the fact that this will not scale to more models in the future. Therefore, we require a more efficient model search strategy, which limits the number of models we need to fine-tune.

4.2.2 Existing Pre-Trained Model Hubs

There exist multiple prominent online pre-trained model repositories, most notably Tensorflow Hub, PyTorch Hub and HuggingFace.[1] Each repository has its own interface for accessing the

[1] `https://tfhub.dev`, `https://pytorch.org/hub`, and `https://huggingface.co/models`

fully trained model, using the corresponding deep learning framework, with an optional direct access to the (last-layer) feature extractor in the case of TF-Hub. Huggingface has an interface for both Tensorflow and PyTorch. These existing online repositories enable easy access to pre-trained models along with their weights and some additional meta-data such as the domain and tasks this model is designed for (e.g., vision and classification), number of parameters, name of the dataset used to pre-train, or performance on this upstream dataset. Using plain search fields or filters, the models can easily be retrieved by their name or some meta-data properties. As we will see in the next section, this enables restricting the model pool and (some) task-agnostic search queries. The implementation and support of any other search strategy is left to the user.

4.2.3 Model Search Strategies

A search query is a function $m(\mathcal{M}, \mathcal{D}, B)$ with a budget B, a set of models $\mathcal{M}$, and a downstream task represented by a dataset $\mathcal{D}$ as its input. The function outputs a set $\mathcal{S}_m \subset \mathcal{M}$ with $|\mathcal{S}_m| \leq B$. The input set $\mathcal{M}$ either represents all models registered in `SHiFT`, or a restricted subset, which we call a *model pool*. The quality of a query is not measured by the metric that the query itself uses (e.g., proxy accuracy or task similarity), but rather by the maximal accuracy attained when fine-tuning the resulting models, those in $\mathcal{S}_m$. We define the difference between this accuracy and the maximal achievable accuracy if one would fine-tune all available models to the user as the *absolute regret*, formally:

$$\max_{m_i \in \mathcal{M}} \mathbf{E}[t(m_i, \mathcal{D})] - \mathbf{E}\left[\max_{s_i \in \mathcal{S}_m} t(s_i, \mathcal{D})\right], \tag{4.2.1}$$

where $t(m, \mathcal{D})$ is the test accuracy achieved when fine-tuning model m on dataset $\mathcal{D}$. Clearly, the budget B, which restricts the size of the subset $\mathcal{S}_m$, has a direct impact on the regret. A large and diverse set $\mathcal{S}_m$, with any diversity measure, is much more likely to result in small regret, while having a budget of $B = 1$ makes the task of finding exactly the best model challenging. The first expectation is taken over the randomness in the $t(\cdot)$ operator, that is, the randomness in fine-tuning and due to a finite sampled test set. In addition to the randomness in $t(\cdot)$, the second expectation also accounts for any potential randomization in the algorithm that computes $\mathcal{S}_m$. We define $s(m)$ as the expected maximal test accuracy achieved by any model in the set $\mathcal{S}_m$, the set of models returned by a fixed strategy m. In our case, kNN is deterministic as all the downstream data is used, whereas the linear model depends on the randomness of stochastic gradient descent. To enable comparability between datasets of different difficulty as well as a

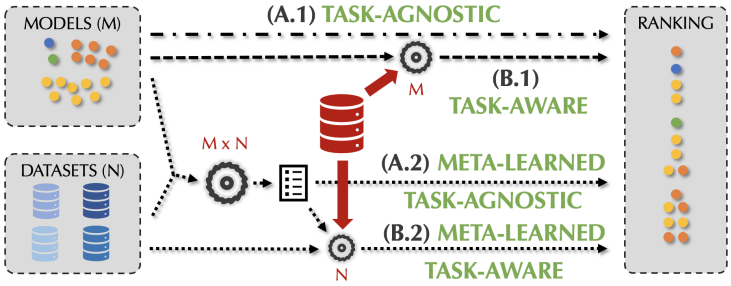


Figure 4.3: Categorization of model search strategies with their computational complexity. The N benchmark datasets are different from the ones used to train upstream, and the computation of the cross-product fine-tune table ($M \times N$) is decoupled from the search complexity.

comparison between two selection strategies m_1, and m_2, we define their *relative delta* as

$$\Delta(m_1, m_2) := \frac{s(m_1) - s(m_2)}{1 - \min(s(m_1), s(m_2))}, \tag{4.2.2}$$

with $s(\cdot) \in [0, 1]$ as defined in Equation 4.2.1. Substituting $s(m_1)$ by the ORACLE value, and $s(m_2)$ by $s(m)$ leads to the *relative regret* $r(m)$.

Pool Restriction The downstream task itself defines the input modality (e.g., image or text) that needs to be supported by the models. Users then usually have use-case specific constraints, such as the framework (e.g., TensorFlow or Pytorch) one is restricted to, or the total number of parameters.

We do not consider the pool restriction itself as a search strategy but rather as an integral part of any other search strategy that we outline next. Note that in the illustration in Figure 4.3, the search strategies do not return a subset of the model pool $\mathcal{M}$, but rather rank models in $\mathcal{M}$. We can map this ranking to our formal description of a search query $m(\mathcal{M}, \mathcal{D}, B)$, by selecting the *top-B* models according to the ranking, randomly breaking ties.

Strategies Figure 4.3 illustrates the different *model search strategies* along with their computational requirements. We divide model search strategies into two main categories: (A) task-agnostic strategies, which are those that ignore the downstream dataset, and (B) task-aware strategies, those that do take the downstream dataset into consideration.

(A.1) Task-Agnostic Search The first category ranks the models in the pool by completely ignoring the downstream dataset. This can range from naively ordering the models by their

name, size, or date of appearance, to more prominent strategies suggested by related work: (a) ranking models trained on ImageNet based on the upstream accuracy (Kornblith et al., 2019) (for images), or the average GLUE (Wang et al., 2018) performance (for text), and (b) favoring more robust models (Deng et al., 2021).

(A.2) Meta-Learned Task-Agnostic Search Instead of restricting to meta-data reported by the model publishers, one could fine-tune every model registered in the system on a fixed set of N benchmark datasets (e.g., the 19 from VTAB (Zhai et al., 2019b)), or on a subset (e.g., only natural datasets in VTAB) and the aggregation metric used to rank the models (e.g., maximum over this subset) chosen by the user. Supporting such meta-learned search strategies requires `SHiFT` to run some computation upon registering a new model by fine-tuning it using all suitable benchmark datasets, whereas the retrieval phase remains independent of the downstream task.

(B.1) Task-Aware Search Using linear classifier accuracies as a proxy to rank the models is often regarded as the standard when incorporating the downstream task into the search process (Kornblith et al., 2019; Deshpande et al., 2021). Instead of fine-tuning the weights of the pre-trained network as described previously, one *freezes* them and only learns the weights of a newly initialized linear head. In Section 4.3, we show that such a linear proxy can suffer from a relatively high regret when trusting this search strategy over exhaustively fine-tuning all models and then picking the best one. Nevertheless, this approach still represents one of the most powerful known search strategy. To improve computation time, researchers have proposed faster proxy methods by approximating the linear classifier accuracy (Bao et al., 2019; Nguyen et al., 2020; Tran et al., 2019), or relying on a cheaper classifier like the k-nearest neighbor (Puigcerver et al., 2021) and its approximations (Meiseles and Rokach, 2020).

(B.2) Meta-Learned Task-Aware Search Here the goal is to favor models that perform well on benchmark datasets *similar* to the downstream one. A prominent way of determining the similarity between datasets representing an ML task was introduced via learned task representations by Achille et al. (2019). To return a ranking of the models, one has to compute a Task2Vec representation for the new dataset and then find the nearest task (i.e., via the distance metric introduced by Achille et al. (2019)). Registering a new model can be computationally demanding as it requires fine-tune accuracies of this new model on all benchmark dataset, but is decoupled from the model search performed by the user.

Hybrid Search Empirically, we show in Section 4.3 that with currently publicly available pre-trained models, for every single method there exists a case in which it suffers from a relatively high regret (i.e., returning a suboptimal model to fine-tune). We proposed to extend the returned set to two or more models, where one can start mixing strategies (e.g., best task-agnostic and best task-aware model). In particular, we show that a simple hybrid search strategy that suggests to fine-tune the top-1 task-agnostic and top-$(B-1)$ task-aware model, often leads to much superior results compared to fine-tuning the best B models based on a single strategy.

4.2.4 Other Related Work

ML Specific Data Management The data-management community has been working on improving the usability of ML in a flurry of work over the last decade by focusing on different components of the ML development process. A few examples include data acquisition with *weak supervision* (e.g., Snorkel (Ratner et al., 2017)), *debugging and validation* (e.g., TFX (Baylor et al., 2017; Polyzotis et al., 2019)), *Model deployment* (e.g., MLFlow (Zaharia et al., 2018)), *knowledge integration* (e.g., DeepDive (Zhang et al., 2017)), *data cleaning* (e.g., ActiveClean (Krishnan et al., 2016)), and *interaction* (e.g., NorthStar (Kraska, 2018)). All these systems facilitate the ML development process, yet none of these focuses on transfer learning specifically.

Model Management The data management community has also seen an intriguing line of work around *model management*. Systems like Cerebro (Nakandala et al., 2020) or ModelDB (Vartak et al., 2016), and follow up works (Kumar et al., 2017; Schelter et al., 2018a), are part of the main motivation to work on this new, transfer learning specific model management system. We hope that by lying the initial conceptual foundation of a model management system specifically for transfer learning, and by open sourcing `SHiFT`, we are able to trigger and facilitate future research in this area.

Dataset Search Neural Data Server (Yan et al., 2020; Cao et al., 2021) takes the approach of searching for *datasets* instead of pre-trained models to improve transfer learning. Other works such as Data2Vec (Baevski et al., 2022) follow a similar goal by embedding a dataset and searching for similarity. Both approaches are orthogonal to our work, since we do not require access to the upstream datasets used to pre-train the models registered in `SHiFT`. Furthermore, it is unclear how these techniques can be used to distinguish models trained on the *same* upstream dataset.

Other Search Strategies There are other search strategies omitted operating on semantical level (i.e., via a learned taxonomy) (Zamir et al., 2018). These methods are not well suited for searching in a pre-trained model hub, mainly given the fact that they assume the input domain to remain fix, and datasets only to be different in their labels.

4.3 A Study on Which Model to Transfer

4.3.1 Experimental Design

Our goal is to assess which model search strategies achieve low regret when presented with a diverse set of models. As discussed, there are three key variables: (i) The set of downstream tasks, which serve as a proxy for computing the expected regret of any given strategy, (ii) the *model pool*, namely the set we explore to find low-regret models, and (iii) the transfer-learning algorithms.

Datasets We use VTAB-1K, a few-shot learning benchmark composed of 19 tasks partitioned into 3 groups – ●*natural*, ●*specialized*, and ●*structured* (Zhai et al., 2019b). The *natural* image tasks include images of the natural world captured through standard cameras, representing generic objects, fine-grained classes, or abstract concepts. *Specialized* tasks contain images captured using specialist equipment, such as medical images or remote sensing. The *structured* tasks are often derive from artificial environments that target understanding of specific changes between images, such as predicting the distance to an object in a 3D scene (e.g. DeepMind Lab), counting objects (e.g. CLEVR), or detecting orientation (e.g. dSprites for disentangled representations). Each task has 800 training examples, 200 validation examples, and the full test set. This allows us to evaluate model search strategies on a variety of tasks and in a setting where transfer learning offers clear benefits with respect to training from scratch (Zhai et al., 2019b).

Models The motivation behind the model pools is to simulate several use-cases that are ubiquitous in practice. We collect 46 classification models:

- 15 models trained on the ILSVRC 2012 (ImageNet) classification task (Russakovsky et al., 2015), including Inception V1-V3 models (Szegedy et al., 2016), ResNet V1 and V2 (depth 50, 101, and 152) (He et al., 2016), MobileNet V1 and V2 (Howard et al., 2017), NasNet (Zoph et al., 2018) and PNasNet (Liu et al., 2018) networks.

- 16 ResNet-50-V2 models trained on (subsets of) JFT (Puigcerver et al., 2021). These models are trained on different subsets of a larger dataset and perform significantly better on a small subset of downstream tasks we consider (i.e. they can be considered as *experts*).
- 15 models from the VTAB benchmark[2], with a diverse coverage of losses (e.g. generative, self-supervised, self-supervised combined with supervised, etc.) and architectures. In all cases the upstream dataset was ImageNet, but the evaluation was performed across the VTAB benchmark which does not include ImageNet.

Model Pools (A) Identifying good resource-constrained models (ResNet-50, Dim2048). Here we consider two cases: (i) ResNet-50: All models with the number of parameters smaller or equal to ResNet50-V2. While the number of parameters is clearly not the ideal predictor, this set roughly captures the models with limited memory footprint and inference time typically used in practice. Most notably, this pool excludes the NasNet and PNasNet architectures, and includes the *expert* models. (ii) Dim2048: The transfer strategies discussed in Section 4.2 might be sensitive to the size of the representation. In addition, restricting the representation size is a common constraint in practical settings. Hence, we consider a model pool where the representation dimension is limited to a maximum of 2048.

(B) Identifying expert models in presence of non-experts (Expert). We consider a pool of 16 ResNet-50-V2 models from (Puigcerver et al., 2021). These models, which are considered as *experts*, are trained on different subsets of a larger dataset. As the number of available models and the upstream training regimes increase, the number of such experts is likely to increase. As such, this presents a realistic scenario in which an expert for the target task may be present, but it is hard to identify it due to the presence of other models, some of which might perform well *on average*.

(C) Do better task-agnostic models transfer better (ImNetAccuracies)? This pool offers the ability to choose an upstream representation-learning technique that is best suited for a specific downstream task. This pool is mainly used to validate the idea that (a) ImageNet models transfer well across different tasks (He et al., 2019; Huh et al., 2016) and that (b) better ImageNet models transfer better (Kornblith et al., 2019).

(D) All models (All). Finally, we consider the hardest setting, namely when the model pool contains all 46 models and no conceptual nor computational restrictions are in place. We note that: Expert $\subset$ ResNet-50 $\subset$ Dim2048 $\subset$ All and ImNetAccuracies $\subset$ All.

[2] `https://tfhub.dev/vtab`

Evaluation procedures **Fine tuning.** To assign a downstream test accuracy to each pair of model and task, we use the median test performance of 5 models obtained as follows: (i) Add a linear layer followed by a softmax layer and train a model on all examples of the training set. (ii) Fine-tune the obtained model twice, considering two learning rates, and 2500 steps of SGD and a batch size of 512 (Zhai et al., 2019b). (iii) Return the model with the highest validation accuracy. Note that in this case, the entire model, and not only the linear layer, is retrained. As a result, there are 10 runs for each model and we obtain 8740 trained models ($46 \times 19 \times 5 \times 2$).

Linear evaluation. We train a logistic regression classifier added to the model representations (fixed) using SGD. We consider two learning rates (0.1 and 0.01) for 2500 steps and select the model with the best validation accuracy. For robustness we run this procedure 5 times and take the median validation accuracy out of those resulting values. As a result, we again obtain 8740 (different) models.

***k*NN evaluation.** We compute the validation accuracy by assigning to each of the 200 validation samples the label of the nearest training example (i.e. $k = 1$) using the Euclidean distance.

4.3.2 Key Experimental Results

In this section we challenge common assumptions and highlight the most important findings of this study. We remark that we only consider three main pools of models – ALL , RESNET-50 and EXPERT, as we see them as the most representative ones. Since DIM2048 behaves very similarly to RESNET-50, whereas IMNETACCURACIES is used only to confirm the findings of (Kornblith et al., 2019), the results of ablation studies involving these two pools can be found in the full publication (Renggli et al., 2022a). Finally, in this section we mainly investigate linear evaluation as the task-aware choice; all the corresponding plots for *k*NN can as well be found in the full publication (Renggli et al., 2022a).

High regret of task-agnostic strategies Figure 4.4 shows the results for task-agnostic methods with budget $B = 1$ and $B = 2$ on the ALL, RESNET-50, and EXPERT pools. Obviously, having a larger budget makes the model search easier at the cost of being computationally more expensive. We therefore set B to be reasonably small, aligned with what practitioners typically do. We observe a significant regret, particularly for RESNET-50 and EXPERT pools (30% of the datasets have a relative regret larger than 25% on those two pools). This highlights the fact that task-agnostic methods are not able to pick expert models, in particular on natural and structured

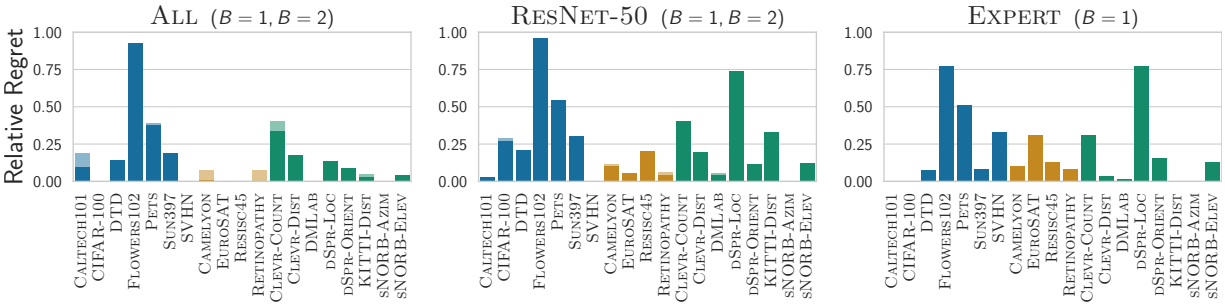


Figure 4.4: Task-agnostic strategies. Relative regret ($r(m)$, cf. Section 4.2) with $B = 1$ (transparent) and $B = 2$ (solid) on the ALL, RESNET-50 and EXPERT pools, bearing in mind that there is only one task-agnostic model in EXPERT. By definition, task-agnostic strategies exclude experts yielding high regret on the RESNET-50 and EXPERT pools, particularly on natural or structured datasets.

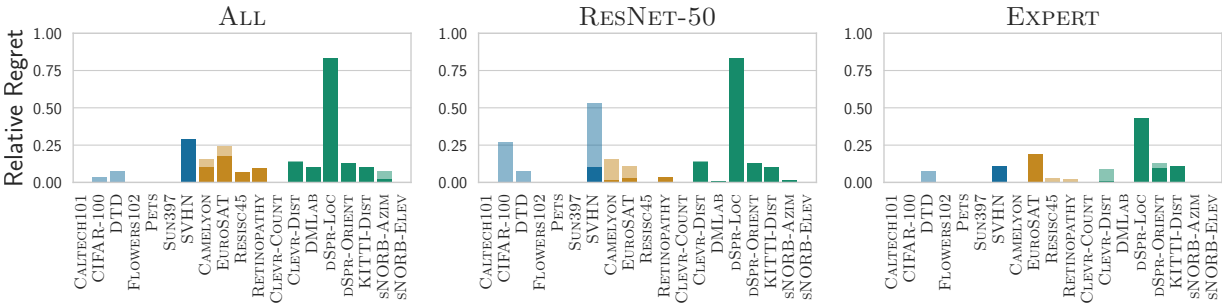


Figure 4.5: Task-aware strategies (linear). Relative regret for $B = 1$ (transparent) and $B = 2$ (solid) on the ALL, RESNET-50, and EXPERT pools. Compared to task-agnostic strategies, we observe improvement on natural datasets (except SVHN) and on restricted pools (except DSPR-LOC), due to its ability to properly choose experts.

datasets. As more experts become available, this gap is likely to grow, making it clear that task-agnostic strategies are inadequate on its own.

Are task-aware always a good predictor? Intuitively, having access to the downstream dataset should be beneficial. Figure 4.5 provides our overall results for the linear model. The method struggles on some structured datasets (in particular on DSPR-LOC).

Compared to the task-agnostic strategy, as presented in Figure 4.6 for $B = 1$, we observe significant improvements on restricted model pools. The EXPERT pool benefits the most: linear evaluation outperforms task-agnostic methods on almost every dataset (task-aware is only outperformed on three datasets by more than 1%, and by 10% in the worst case on the KITTI-DIST dataset). On the other hand, task-agnostic and task-aware strategies seem to outperform each

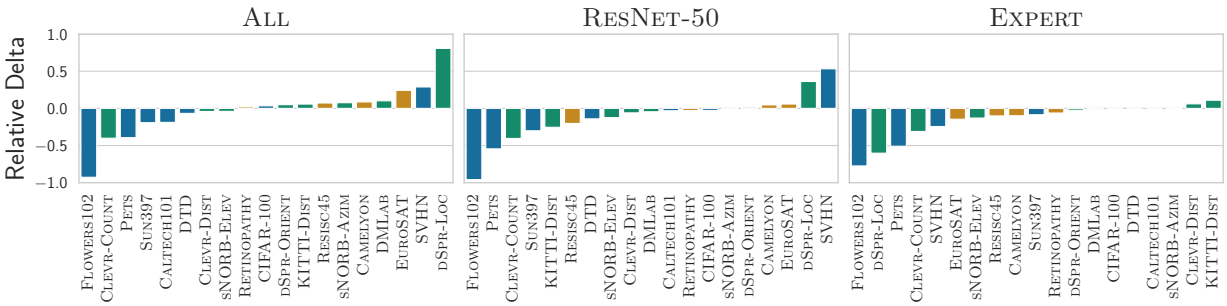


Figure 4.6: Task-agnostic (positive if better) **vs Task-aware (linear)** (negative if better) for $B = 1$. On the All pool, the methods perform in a similar fashion, with respect to the number of datasets and the amount in which one outperforms the other. When one restricts the pool to ResNet-50 or Expert task-aware methods outperform the task-agnostic method on most datasets. The relative delta is defined in Equation 4.2.2 in Section 4.2.

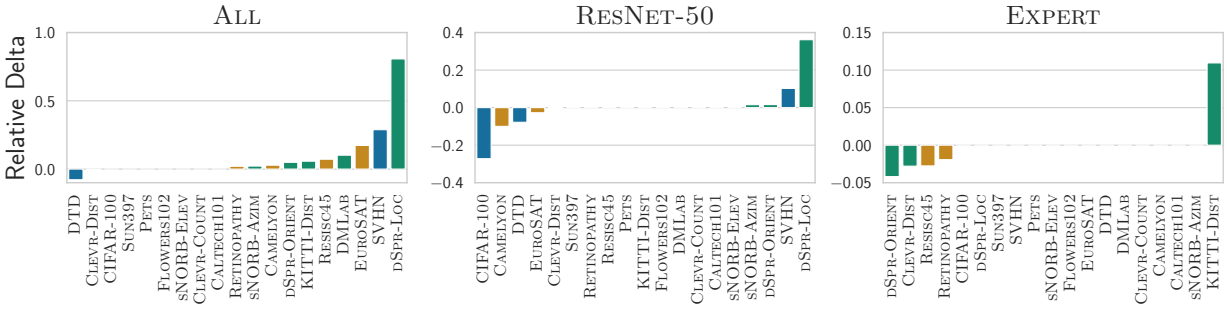


Figure 4.7: Hybrid linear (positive if better) **vs Linear evaluation** (negative if better) for $B = 2$. We observe that hybrid linear significantly outperforms linear with the same budget on the All pool. Even though for ResNet-50 and Expert pools there are datasets on which linear performs better than hybrid, the amounts in which it does are usually small. We note that most significant gains of hybrid come on certain structured datasets, the hardest task for every strategy.

other on a similar number of datasets and by a comparable magnitude in the All pool. This suggests that no single strategy uniformly dominates all the others across pools.

Hybrid approach yields the best of both worlds A hybrid approach that selects both the top-1 task-agnostic model and the top-$(B-1)$ task-aware models leads to strong overall results. Figure 4.7 shows how the hybrid approach with linear evaluation as the task-aware method significantly outperforms its linear counterpart with $B = 2$. This is most noticeable in the All pool where the task-agnostic model provides a large boost on some datasets.

As we saw in Figure 4.6, when looking at the All pool, the task-agnostic candidate tends to beat the linear one on datasets such as dSpr-Loc, SVHN or EuroSAT. Similarly, the linear candidate model clearly outperforms its task-agnostic counterpart on many natural datasets such

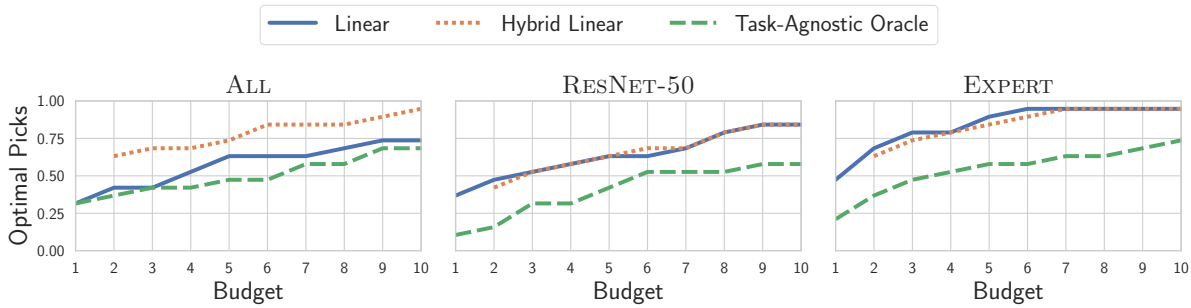


Figure 4.8: Optimal picks as a function of the computational budget. The number of picked models (relative) with zero regret across three representative pools. We note that hybrid linear outperforms all other methods on ALL, whilst being comparable with the linear strategy on restricted pools where linear alone already performs well. Here, the task-agnostic oracle refers to a method which ranks models based on their average accuracy across all datasets.

as FLOWERS or PETS. A comparison of Figures 4.6 and 4.7 reflects how the dominance of linear-only strategy vanishes on most datasets when confronted with the hybrid approach. For the RESNET-50 and EXPERT pools, as expected, the hybrid algorithm preserves the good picks of the linear proxy. That said, we observe an increase of 36% on DSPR-LOC in the RESNET-50, and 11% on KITTI-DIST. Both are structured datasets on which the linear proxy task performs poorly, as shown in Figure 4.5.

The hybrid strategy requires to fine-tune at least two models. Given that it performs well across all model pools and datasets, this is a reasonable price to pay in practice, and we suggest its use as the off-the-shelf approach.

How does the computational budget impact the findings? We have seen that for a limited budget of $B = 2$ the proposed hybrid method outperforms the other strategies. A natural question that follows is: how do these methods perform as a function of the computational budget B? In particular, for each budget B, we compute how frequently does a strategy pick the best model. The results are shown in Figure 4.8. We observe that the hybrid linear strategy outperforms all individual strategies on the ALL pool. Furthermore, it also outperforms a strong impractical task-agnostic oracle which is allowed to rank the models by the average fine-tune accuracy over all datasets. Our hybrid strategy achieves an on par performance with the linear approach on pools on which linear performs well. When task-aware strategies perform badly (e.g. pools without expert models), hybrid linear is significantly stronger. These empirical results demonstrate that the hybrid strategy is a simple yet effective practical choice for a model search strategy.

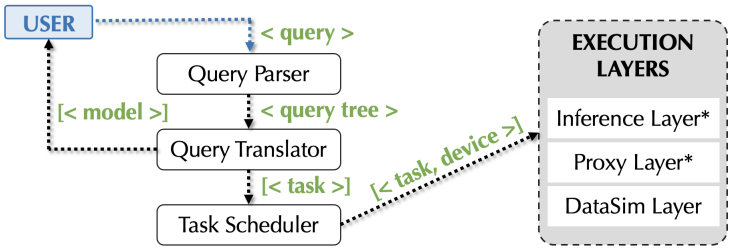


Figure 4.9: The logical components and different types that connect them that compose SHiFT, where (*) represents components that we optimize in Section 4.4.2 and 4.4.3.

4.3.3 Need For SHiFT

While the hybrid strategy outlined above represents the most robust choice for searching models, users may have diverse search requirements. Hence, we require a system that gives users the *flexibility* to express a model search query that reflects the user's most important criteria. The system should also *efficiently* execute search queries, applying query execution optimizations under the hood. Bearing in mind that model development is typically an iterative process, if a user is not happy with the returned models or their fine-tuning results, she may iterate by changing the data or the query, and then use the newly returned models. Thus, we need a system that can support various search strategies with efficient initial and iterative executions.

4.4 Searching Models with SHiFT

4.4.1 Systems Architecture

We now present the architecture of SHiFT, allowing us to efficiently and flexibly execute the search queries outlined in the previous section. We abstract our system into multiple logical components visualized in Figure 4.9. The labels on the arrows indicate *types* of the input for each component.

Like many other data systems, SHiFT is designed as a server-client architecture connected by standard HTTP requests. From the client perspective, the input to the system is a SHiFT-QL query, and the output is the corresponding result, i.e., a list of models, together with the information whether the query was executed successfully or not. On the server side, SHiFT takes the parsed SHiFT-QL query tree as the input, and determines how to provide the results. SHiFT

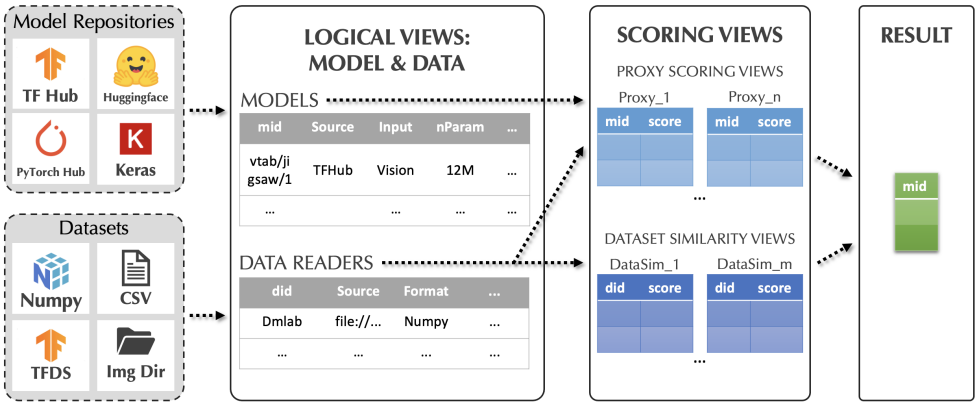


Figure 4.10: SHiFT's Logical Model of Transfer Learning

internally caches intermediate results within and across queries to optimize execution time. We elaborate on caching and other system optimizations in Section 4.4.2.

4.4.1.1 Logical Model of Transfer Learning

One of the most important challenges in building SHiFT is to provide a clean abstraction for the search process of transfer learning, which needs to be flexible enough to model most popular search processes that users are using in practice, but in the meantime, needs to be high-level enough for us to capture the opportunities of system optimizations and incremental maintenance. In the following, we describe the SHiFT's logical model, which is based not only on our own experience in model search (c.f., Section 4.3) but also a comprehensive survey of existing popular search strategies (Kornblith et al., 2019; Achille et al., 2019; Bao et al., 2019; Nguyen et al., 2020; Meiseles and Rokach, 2020; Puigcerver et al., 2021; Deshpande et al., 2021; Mensink et al., 2021).

Unified Logical Views for Model Repositories and Datasets During the model search process, there are two key players: (1) a diverse collection of model repositories (e.g., TF-Hub, PyTorch Hub, and Huggingface) and (2) a collection of datasets stored in a diverse set of formats (e.g., numpy, CSV, TFDS, etc.). The starting point of SHiFT is to provide a unified view for both models and datasets.

The '`Models`' view is a relational view containing information of models across various sources. As illustrated in Figure 4.10, each model corresponds to a single row in the 'Models' view, which contains "meta-data" about its source, modality, number of parameters, etc. Each model ID also

is associated with various functions to deal with tasks such as inference and fine-tuning, all of which are virtualized in dockerized environments to unify the API differences of different model repositories, which we formalize later. In practice, we observe that having a relational view for 'Models' is particularly useful as users often conduct specific filtering queries over all models (e.g., to only keep models with # parameters smaller than a given constant to ensure inference latency). These can be done via standard SQL queries over the 'Models' view.

The 'DataReaders' view is a relational view containing information of datasets stored in different formats. Each dataset corresponds to a single row in the 'DataReaders' view, which contains its meta-data. Each unique DataReader is also associated with an *iterator* that enumerates (x, y) pairs where x is a Numpy array for a single feature vector and y is a Numpy array for a single label.

SHiFT-QL Query A SHiFT-QL query defines a unique search strategy for transfer learning. In our design, a SHiFT-QL query consists of two components: (1) a collection of "scoring views" and (2) a generic SQL query over these scoring views. Note that all scoring views are *lazily evaluated up to various fidelity and precision*, which renders the query execution and optimization non-trivial and unique for SHiFT.

Lazily Materialized top-K Scoring Views Given the 'Models' view and the 'DataReaders' view, a user can define scoring views of two types. The first type, which we call *proxy scoring views* aims to capture proxy tasks that are used to assess a model's transferability. A proxy scoring view extends the SQL syntax and can be defined as

```
CREATE PROXY SCORING VIEW name
SQLQUERY          -- e.g., SELECT * FROM Models
                  --       WHERE nParam < 12M
<SHIFT>
  ORDER BY ScoringAlgorithm [DESC | ASC] LIMIT K
  [TESTED ON DataReader1]
  [TRAINED ON DataReader2]
  [WITH DataReader3 ...]
</SHIFT>
```

where SQLQUERY is a standard SQL query whose output has the same schema as the Models view. In this way, a user can use arbitrary SQL queries to conduct different filtering strategies or join with auxiliary information about models to select models.

Given all models that SQLQUERY returns, a ScoringAlgorithm is associated with a function that maps a single Model, and a series of DataReader into a real-valued score:

$$\texttt{ScoringAlgorithm} : \texttt{Model}[\times\texttt{DataReader} \times ... \times \texttt{DataReader}] \rightarrow \mathbb{R}$$

By default, a proxy scoring view will only returns the top-K models according to the output of the ScoringAlgorithm. This is often the case in most search strategies that we see in practice, and as we will see later, will open up novel opportunities for system optimizations.

The second type of scoring views are what we call *dataset similarity views*, which are used to compute similarities between different datasets, an important signal in many model search strategies — a model performing well on a *similar* datasets is likely to perform better in many scenarios, if we are able to compute datasets similarities reliably. A dataset similarity view also extends the SQL syntax and can be defined as

```
CREATE DATASET SIMILARITY VIEW name
SQLQUERY          -- e.g., SELECT * FROM DataReaders
                  --       WHERE Modality = Image
<SHIFT>
  ORDER BY DataSimMetric [DESC | ASC] LIMIT K
  TESTED AGAINST TargetDataReader
</SHIFT>
```

where SQLQUERY is a standard SQL query whose output has the same schema as the DataReaders view. In this way, a user can use arbitrary SQL queries to conduct different filtering strategies or join with auxiliary information about datasets to the select dataset.

DataSimMetric is associated with a function that computes the similarity between a pair of datasets:

$$\texttt{DataSimMetric} : \texttt{DataSet} \times \texttt{DataSet} \mapsto \mathbb{R}$$

Given all datasets returned by SQLQUERY, SHiFT computes its similarity with the target TargetDataReader. Similar to a proxy scoring view, a dataset similarity view also by default keeps the top-K datasets according to its similarity with the TargetDataReader.

SHiFT-QL Query: Putting Things Together Given a collection of scoring views, a SHiFT-QL query is a generic SQL queries querying these views. This allows flexible aggregation and voting strategies and are crucial for many search strategies.

We allow several syntax sugars to make the query more succinct. When there is no ambiguity, we often ignore the <SHIFT> and </SHIFT> tags. Moreover, we also support creating scoring views implicitly if such a query is nested in another SQL query.

As an example, to specify the hybrid search strategy developed earlier: *Return the vision model with fewer than 10M parameters and the best upstream accuracy and another vision model that has the best linear classifier accuracy,* a user can write the following SHiFT-QL query:

```
(SELECT ModelId FROM Models
WHERE Input = 'Vision' AND nParam <= 10M
ORDER BY UpstreamAccuracy DESC LIMIT 1) Q1

                    UNION

(SELECT ModelId FROM Models
WHERE Input = 'Vision' AND ModelId NOT IN Q1
ORDER BY Linear(lr=0.1) ASC LIMIT 1
TESTED ON TestReader TRAINED ON TrainReader) Q3
```

Tracking Data Changes via Change- and Add-Readers To support efficient incremental executions over data changes, we extend this simple concept of a data reader to a *mutable* reader, allowing a reader to be a composite of an initial data reader and a list of *change-* or *add-readers*. Every change-reader is accompanied by a list of indices of same length as the initial reader itself, indicating which samples to replace. The change- and add-readers are then processed in a linear order to build the final mutable data reader.

The advantages of representing our data as such are extensibility to other data sources and the ability to cache inferred features on a per-reader (initial, change, or add) level. On the flip side, removing samples from a data reader requires users of SHiFT to define a new reader, resulting in a new execution from scratch. We plan to support deletions in the future.

It is also important to note that our current data provenance system that tracks the changes is a rather naive one. In the future, we should provide the support of modern data provenance

systems for ML, e.g., mlinspect (Grafberger et al., 2021), and ArgusEyes (Schelter et al., 2022). We are optimistic that such an integration should be natural and easy.

Supporting New Search Strategies One key design goal of SHiFT-QL is to make it easier for researchers to provide new search strategies in the future. In our current system, this can be done by registering a new ScoringAlgorithm or a new DataSimMetric. We are optimistic that this could support many new search strategies in the near future (e.g., the "more robust model is more transferable" strategy (Deng et al., 2021)). Nevertheless, it is possible that there will be new search strategies that break our current abstraction — we will continue to monitor the state-of-the-art research and adapt to them.

4.4.1.2 Examples

Given the flexible logical abstraction of SHiFT, we are able to express a diverse range of search strategies, while allowing a user to conduct her own filtering and selections operations using standard SQL queries. As an example, the five popular search strategies in Table 4.1 can be expressed in SHiFT as follows:

```
Q1 := SELECT ModelId FROM Models
      WHERE Input = 'Vision'
      ORDER BY UpstreamAccuracy DESC LIMIT 1

Q2 := SELECT ModelId FROM Models
      WHERE Input = 'Vision'
      ORDER BY CosineNN ASC LIMIT 1
      TESTED ON TestReader TRAINED ON TrainReader

Q3 := SELECT ModelId FROM Models
      WHERE Input = 'Vision'
      ORDER BY Linear(lr=0.1) ASC LIMIT 1
      TESTED ON TestReader TRAINED ON TrainReader
```

```
Q4 := Q1
      UNION
      SELECT ModelId FROM Models
      WHERE Input = 'Vision' AND ModelId NOT IN Q1
      ORDER BY Linear(lr=0.1) ASC LIMIT 1
      TESTED ON TestReader TRAINED ON TrainReader

Q5 := SELECT ModelId FROM Models
      WHERE Input = 'Vision' AND
            Readers.ReaderId IN
            (
                SELECT DataReaderId FROM DataReaders
                ORDER BY Task2Vec LIMIT 1
                TESTED ON TestReader
            ) Q6
      ORDER BY FineTune LIMIT 1
```

4.4.1.3 Query Parser and Translator

Our query parser takes a possibly complex SHiFT-QL query as an input, and generates a parsed query tree, where every node is either a proxy scoring or dataset similarity view. The tree is then traversed in a bottom-up approach, by evaluating the leaf nodes until completion before evaluating the parent nodes. We leave query tree optimizations such as push-down operations or balancing compute across different nodes for future work. Every SQLQUERY is evaluated directly against our database. Task-agnostic search strategies represent SHiFT-QL queries using neither proxy scoring nor dataset similarity views. Meta-learned task-agnostic queries can be defined by the user by using dataset similarity views and filtering models using populated benchmark fine-tune results in it. For queries relying on proxy scoring views, the query translator will check if the result for the specified method name is known in the corresponding view (i.e., a tuple for each model and the specified readers exists in the database). If so, the system will directly return the results, or pass them to a parent node in the parsed query tree. Otherwise, the system will dispatch a list of SHiFT tasks for missing values to the task scheduler. Task-aware search queries are split into two inference tasks per model, one for the test and another for the train data source, and an additional single proxy task per model. The outputs of the inference tasks are used as input to the corresponding proxy task. This allows us to possibly reuse the cached feature representations per pre-trained model. To support better load balancing, we split both

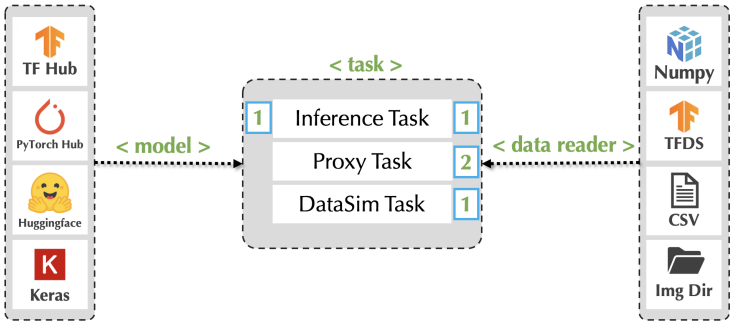


Figure 4.11: SHiFT tasks and their dependencies to model or data sources. The numbers specify on how many objects of each type a specific task depends on (e.g. Proxy tasks depends on 2 data readers).

inference tasks for partitions of the datasets (c.f., Section 4.4.2). Meta-learned task-aware search queries rely on dataset similarities. If the embedding for the downstream task reader is not in the database, the system will dispatch a DataSim task to compute it. The embeddings are then used to compute the distance between tasks and rank the readers.

4.4.1.4 Task Scheduler and Execution

A SHiFT task represents the smallest computational unit of our system. The scheduler assigns every task to a single hardware device. We define three different tasks SHiFT supports: (1) inference tasks, (2) proxy tasks, and (3) DataSim tasks. The number and types of tasks executed depend on the query provided by the user, then parsed and translated by SHiFT (e.g., Q2 will only create inference and proxy tasks). Each of them has dependencies on models or data readers, or both (c.f., Figure 4.11). The scheduler of SHiFT is fairly simple. In a first-in-first-out (FIFO) basis, every task gets assigned to the next free device as soon as its dependency tasks are successfully terminated. Every GPU on a single machine represents a device, and a subset of the CPU cores forms another one. Costly inference tasks are assigned to GPU devices, if any are available. Proxy and DataSim tasks are also handled by the CPU. Further optimizations, including scaling SHiFT to multiple machines, are left as future work.

Inference Task To support various pre-trained model sources and frameworks, we define a minimal interface used by inference tasks. All supported model sources require a simple forward function for a batch of samples originating from every data source visualized in Figure 4.11. This function returns a 2D Numpy array with each sample (out of n) in the data reader representing a row. The feature dimension is determined by the pre-trained model. The resulting extracted

features are stored on the disk, and references to the corresponding reader and model combination, using a hash of the earlier, and the model name as a unique identifier of the later are saved. Formally, the following interface needs to be specified for each combination of models and data sources.

```
def extract(pre_trained_model: model,
            source: data_reader) ->
            np.array(shape=[n,dim])
```

Most of these 16 possible combinations are natively supported by the frameworks and API (e.g., using the Keras `fit` function for parts of a Keras model, or `KerasLayer` with TF-Hub models), or by casting the data sources into a supported format (e.g, using tensorflow data sources for both Keras and TF Hub). PyTorch Hub models typically only store the models with their PyTorch code and no standardize interface. Every model registered in SHiFT therefore needs to specify a hook function to extract the (last-layer) features, which requires to know the internal structure of the models (i.e., the layer names). Custom trained or fine-tuned models can be exported as Keras models to disk, and then used for subsequent search queries upon registration into the database.

Proxy Task By splitting task-aware search queries into inference and proxy tasks, we bypass the requirement of implementing the proxy computation for every combination of model and data source. The proxies are defined over Numpy arrays, where the extracted n train and m test features (ending with `_X`) stem from a model and data source combination after having performed an inference task, and the labels (ending with `_y`) are independent of the models.

```
def compute_proxy(train_X: np.array(shape=[n,dim]),
                  test_X: np.array(shape=[m,dim]),
                  train_y: np.array(shape=[n,]),
                  test_y: np.array(shape=[m,])) ->
                  proxy_value: float
```

We implement two different proxy estimators: (1) the nearest neighbor (NN) accuracy for two different distance functions (cosine dissimilarity, and Euclidean, L2 distance), and (2) a linear classifier accuracy trained with stochastic gradient descent (SGD) and arbitrary hyper-parameters, such as learning rate, L2 regularizer, mini-batch size and number of epochs.

DataSim Task A dataset similarity (DataSim) task computed embeddings of a data reader representing a machine learning task (e.g., Task2Vec (Achille et al., 2019) to compute 8512 dimensional vectors). The embedding is then stored along with the meta-data of the reader in the database. Using any distance function (e.g., the non-symetric one suggested by Achille et al. (2019)), a subset of registered data readers in the database, for which the embedding are pre-computed, can be ordered and limited in a straightforward fashion. The final meta-learned task-aware search query is then no different from the meta-learned task-agnostic. We use the code provided by Achille et al. (2019) for running the DataSim tasks, leaving optimization as future work.

4.4.2 System Optimizations

We describe multiple optimization incorporated into `SHiFT` next.

4.4.2.1 Successive-Halving (SH)

In `SHiFT`, all proxy scoring views consist of a top-`K` query over a list of scores; furthermore, each score is computed as a function over `DataReader`s which consist of a set of data examples. This structure opens up unique opportunities for system optimizations — since many of these scoring functions are relatively stable with respect to sub-sampled datasets, we can *approximate* this top-`K` view with a scoring function evaluated over only a *subset* of data examples.

One key optimization is to estimate the proxy value only for a small subset of the (training) data on most models, and a large fraction of the data only on a small subset of the models. There can be various ways for this. Currently, `SHiFT` uses the same approach as `Ease.ML/Snoopy` (c.f., Section 3.6.3) by impementing the successive-halving (SH) (Jamieson and Talwalkar, 2016) algorithm (without the tangent optimization), which is invoked as a subroutine inside the popular Hyperband algorithm (Li et al., 2017). Algorithm 1 outlines the algorithm, noting that an *arm* in our context represents a model, and *pulling an arm* corresponds to running inference on more data and estimating the proxy using the extracted features for *all* the data seen by the model so far. In a nutshell, we can summarize the idea of SH as follows: Start by uniformly allocating a fixed initial budget ($B/\log_2(M)$) to all M models and then evaluating their performance. Keep only the better half of the models and repeat this until a single model remains. The algorithm has two different hyper-parameters: a chunk size (how many sample represent an arm pull) and the overall budget B. Both can be specified by the user and have an impact on the accuracy

of the results and compute time. We propose a chunk size guaranteeing that the last model has processed the entire dataset as a default for SHiFT, and use the minimal budget required to return a fixed number of models.

SH Minimal Budget and Chunk Size In order to preserve the semantic of the queries whilst performing successive halving (i.e., not sub-sampling the data), we need to guarantee $r_k > 0, \forall k \iff \frac{B}{L\lceil \log_2(|M|)\rceil} > 1$. Let us assume a top-q queries with $q = 1$ (the derivation can simply be extended to arbitrary values of q). We need $B \geq L\lceil \log_2(|M|)\rceil, \forall L$. The largest L is reached at the first step where $L = |M|$. Hence, we need to have

$$B \geq |M|\lceil \log_2(|M|)\rceil$$

Conversely, at kth step, each remaining model is given $r_k \times C$ additional training samples, in total each remaining model has processed $\sum_{j=0}^{k} r_j \times C$ training samples. When $k = \lceil \log_2(|M|)\rceil - 1$, the remaining models have processed $C \times \sum_{j=0}^{\lceil \log_2(|M|)\rceil - 1} r_j$ training samples. Hence, the minimal chunk size C_{min} such that the remaining models have processed all training samples is given by

$$C_{min} = \frac{N}{\sum_{j=0}^{\lceil \log_2(|M|)\rceil - 1} r_j}$$

where $r_j = \lfloor \frac{B}{\lceil M/2^j \rceil \lfloor \log_2(|M|) \rfloor} \rfloor$.

4.4.2.2 Cost Model for SH

Successive-halving, while always being able to decrease the amount of examples processed, does not always outperform the baseline strategies in wall clock time. Moreover, as we show in the experiments, it can sometimes even be slower. This might seem counter-intuitive, but the main reason lies in hardware accelerators, such as GPUs, which offer the ability to massively parallelize tasks up to a fixed number of samples. For instance, running inference for one sample through a deep neural network requires roughly the same time as a mini-batch of multiple samples. The maximum mini-batch size is often limited by the device memory. Therefore, one should not split very small readers into multiple chunks to speed up a task, rendering SH inappropriate for small datasets. Additionally, SH introduces sequential dependencies between tasks, which can render the algorithm inefficient or unable to scale to multiple GPUs. One such a cause lies in the repeated model loading, or access to the extracted test features, noticing that we always use the entire test set to estimate a proxy value after a pull.

Therefore, we introduce another key component into SHiFT: a cost-based decision making process that automatically decides whether to use successive halving. To this end, we derive a cost model for SHiFT with and without SH. Our cost model requires a few variables, either precomputed or available based on the query. Let N be the size of our training dataset, O the size of the test dataset, and M the number of models. Furthermore, let P represent the number of equal devices (e.g., GPUs). The time required to load model i onto a device is given by L_i. Furthermore, the time to load the (training) dataset is represented by T_N, whereas the time required to load the inferred test representations for model i is given by T_i. We simplify this by assuming a global T_O, since the representations only differ in their dimensions. We neglect the time to load the raw test dataset as this is equal regardless of the optimization. The time to run inference for k samples and model i on the device is given by $I_i(k)$, which we assume to be linear, hence $I_i(k) = I_i k$, for some constant I_i. The time to compute a proxy $E^i_{Proxy}(k)$ follows the same principle, although we assume that it is independent of the model (neglecting the dimension of the representations), hence $E^i_{Proxy}(k) = E_{Proxy}k$.

The cost for running a top-1 query without SH on multiple GPUs is computed with

$$T_{\text{w/o}} := \frac{1}{P}\sum_{i=1}^{M}\Big(\underbrace{T_N + T_O}_{\text{Load data}} + \underbrace{2L_i}_{\text{Load model}} + \underbrace{I_i O}_{\text{Test inference}} + \underbrace{I_i N}_{\text{Train inference}} + \underbrace{E_{Proxy} N}_{\text{Proxy estimation}}\Big),$$

Notice that we need to load the train dataset again every time when there is a new model (request), as every task is executed independently. The counter part, running a top-1 query with SH on multiple GPUs, where we assume perfect parallelization which is harder to achieve for heterogeneous models and small chunks C, is given by

$$T_{\text{w/}} := \underbrace{\frac{1}{P}\sum_{i=1}^{M}(L_i + I_i O)}_{\text{Test inference}} + \underbrace{\sum_{k=1}^{\lceil \log_2(M)\rceil}\frac{1}{\min(P, |S_k|)}}_{\text{SH iterations}} \times \Big(\underbrace{\sum_{j\in S_k}\big(L_j + T_N + I_j C r_k\big)}_{\text{Train inference}} + \underbrace{\sum_{j\in S_k}\Big(T_O + E_{Proxy}\sum_{l=1}^{k}(C r_l)\Big)}_{\text{Test load and proxy estimation}}\Big),$$

where C is specified by the user or taken as C_{min}, S_k and r_k are taken from Algorithm 1. Clearly, the sets S_k of models *surviving* during the SH algorithm have an impact on the runtime. Following the trend of larger and slower models surviving the longest, we define S_k to be the set of k models with the largest inference time I_j for all $j \in [M]$.

SHiFT will use this cost model (i.e., the mininum of $T_{w/o}$ and $T_{w/}$) to automatically decide, based on system's specifications (e.g., hardware devices and model inference times), number of samples, and models in the restricted pool, whether or not to use the SH algorithm.

SH for Other Queries We only use the SH optimization for task-aware queries for which a fraction of the samples can be used for *ranking* models with high confidence. Classifier accuracies (e.g., NN or linear classifier) are known to satisfy this property (Rimanic et al., 2020). Furthermore, the overhead of running multiple sequential tasks when using the SH algorithm is kept small for task-aware queries, thanks to the two-stage approach (i.e., the inference and proxy estimation phase), which is not the case for the other search queries. Inference tasks are typically much more time consuming compared to the proxy estimation. Nonetheless, when *pulling* an arm an additional time, the data from the previous arm pulls are not required to be run through the network again as they can be fetched from the disk for the subsequent proxy estimation tasks.

4.4.2.3 Other Optimizations

We implement two other, more system specific optimizations into SHiFT.

Caching Caching is crucial for rapid incremental query executions. SHiFT internally caches dataset similarities, feature vectors, and proxy values in order to reuse intermediate results within and across queries as much as possible. When dispatching task requests, SHiFT ensures that only necessary requests are executed. For example, if a user sends a request for a proxy estimator (e.g., the nearest neighbor accuracy) and a single model, SHiFT will create two inference requests, one for the test and train reader, and a proxy estimator request. The former requests turn the input data into feature vectors and store them on the disk. If, at a later stage, the user requests another proxy estimator for the same model (e.g., the linear classifier accuracy), SHiFT will notice that it can reuse the cached feature vectors. Therefore, SHiFT will only dispatch a proxy estimation request, which uses the feature vectors to calculate the proxy value. The dispatched requests are then handled by the task scheduler asynchronously. Once a certain request is done, the results will be written into the database and ultimately returned to the user upon running the same query.

Load-Balancing To automatically load-balance heterogeneous workloads, mainly stemming from large discrepancies in inference times between models, to multiple hardware devices, we automatically split readers for every inference task. The number of partitions is equal to the number of GPU devices, unless a reader is smaller than a fixed threshold. This guarantees fast executions especially when running inference on small readers (i.e., change-readers).

4.4.3 Incremental Executions

If a user is not satisfied with the results of the query, either in terms of the proxy values or other properties of the models returned (e.g., diversity or downstream performance), she typically iterates by incrementally executing another, often similar query, or re-run the same query performing one of the following changes:

1. add a model to the database,
2. change the data (features or labels) to test or train readers by using change-readers,
3. add data to the test or train reader using an add-reader.

Incremental Scenario 1: Changing Queries or Adding Models Changing a query or adding a model to the database will naturally benefit for the caching mechanisms we introduced in the previous section. For example, a task-aware search query will only require `SHiFT` to dispatch two inference tasks and a single proxy task per additional model, retrieving the other results directly from the database. When using the SH optimization, the same idea of reusing the caches for intermediate results (features and proxy estimation values) applies. Data manipulations are slightly more involved. Based on the defined mutable data reader concept in Section 4.4.1, costly inference operations only need to be executed for the changed features. The cheaper proxy task is then evaluated on the final mutable reader (i.e., after having iterated over all change- and add-readers) only once, or for every subsequent iteration in the SH algorithm.

Incremental Scenario 2: Adding Data Adding data to the test reader requires us, similarly to changing data, to run inference on all models of the query using these additional data samples, and then rerun the proxy estimation value for every intermediate step in the SH algorithm. However, blindly appending training data at the end of the data reader can result in an undesired behavior. Concretely, all models that did not process the entire dataset will not benefit from the appended samples. This can be problematic if the added data stems from a different distribution, thus possibly *eliminating* different arms while running the SH algorithm. We construct and

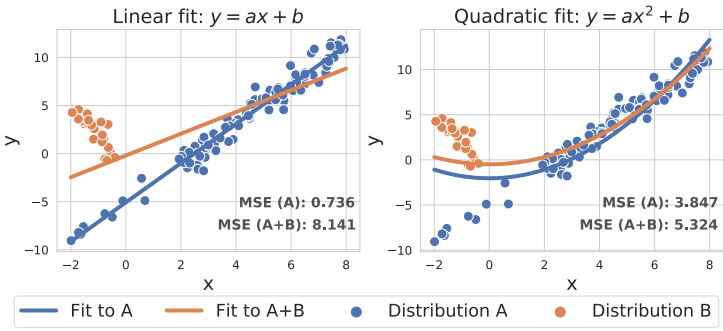


Figure 4.12: Two models (linear and quadratic), for which the order (i.e., based on the minimal MSE) changes if they have access to data only from the first distribution (A), compared to both distributions (A+B).

illustrate one such case in Figure 4.12, where we assume to have two models (a linear one on the left and a quadratic one on the right), and data coming from two distributions (A in blue and B in orange). If the models were ranked on the basis of the errors of distribution A alone, the quadratic model would be eliminated. If then data from distribution B is appended to the reader (via an add-reader), the winning linear model from before would actually be inferior to the quadratic one.

To address this issue, ideally, one would randomly reshuffle the full data reader with appended data. Despite being favorable from a statistical point of view, managing the cache and preventing a complete re-execution is far from trivial. The alternative solution implemented in SHiFT uniformly distributes the new samples into the existing chunks.

In SHiFT we choose this alternative strategy by default, which enables high performance and results in little difference compared to the fully shuffle approach from a statistical point of view. As a justification of its statistical property, let us assume that we have an initial data reader of size N together with an add-reader that contains αN samples, with $\alpha \geq 0$. Moreover, assume that the chunk sizes used by the SH algorithm on the initial data reader is of size βN, with $0 \leq \beta \leq 1$. We want to compare the two strategies: (1) randomly inserting the new samples anywhere between the existing ones and re-partitioning the samples into buckets afterwards, and (2) uniformly distributing the new samples amongst all existing buckets. Notice that the number of buckets remains constant after handling the new samples, thus increasing the size of the buckets to $\beta N\,(1 + \alpha)$. Furthermore, it is obvious that from an implementation perspective, as outlined in the optimization Section 4.4.3, the second strategy is superior to the first, whereas the first strategy introduces less bias into the sampling process. However, both approaches yield the same number of samples from both distributions in expectation. We define a random variable

Table 4.2: Execution time for fine-tuning (FT) all the models compared to running Q2-Q4 using `SHiFT`. The baseline for each query represents the runtime of `SHiFT` without any automatic optimization.

			1 GPU		8 GPU	
			Runtime (Hours)	Speedup (vs. FT)	Runtime (Hours)	Speedup (vs. FT)
CIFAR	FT	Baseline	251.8		31.5	
	Q2	Baseline	8.7	28.8x	1.6	19.8x
		`SHiFT`	5.9	**42.9x**	1.3	**24.1x**
	Q3	Baseline	9.1	27.6x	1.6	19.2x
		`SHiFT`	5.8	**43.4x**	1.3	**24.3x**
	Q4	Baseline	7.9	31.7x	1.0	**30.0x**
		`SHiFT`	5.6	**45.0x**	1.2	25.6x
Dmlab	FT	Baseline	314.8		39.4	
	Q2	Baseline	14.3	22.0x	2.4	16.2x
		`SHiFT`	9.6	**32.7x**	2.0	**19.7x**
	Q3	Baseline	14.6	21.6x	2.5	16.0x
		`SHiFT`	9.9	**31.8x**	1.9	**21.2x**
	Q4	Baseline	13.3	23.7x	1.7	**22.6x**
		`SHiFT`	9.3	**33.9x**	2.0	20.0x
Flowers	FT	Baseline	9.6		1.2	
	Q2	Baseline	2.9	**3.3x**	0.4	**2.8x**
		`SHiFT`	2.9	**3.3x**	0.4	**2.8x**
	Q3	Baseline	3.1	**3.1x**	0.4	**2.7x**
		`SHiFT`	3.1	**3.1x**	0.4	**2.7x**
	Q4	Baseline	2.8	**3.4x**	0.4	**3.1x**
		`SHiFT`	2.8	**3.4x**	0.4	**3.1x**

C, which represents the number of initial samples in the first bucket when the first strategy is run. Coincidentally, C follows a hyper-geometric distribution with an expectation of

$$\mathbb{E}[C] = \beta N,$$

which is exactly the number of samples in the first bucket when applying the second strategy.

4.4.4 Evaluation

Experimental Setup We conduct our experimental study next on a single modality by focusing on computer vision tasks, representing one of the most prominent application of machine learning and transfer learning specifically (Zhai et al., 2019b). Nonetheless, SHiFT is flexible and the code-base supports workload beyond this single and even to multiple modalities.

We compile a diverse list of 100 models, whose details, including additional configuration such as mini-batch size per model and inference time for the GPU type needed in the next paragraph, are given in the supplementary material in Appendix B.1.3. We fine-tune the models for 20 epochs, using a mini-batch size of 16, momentum of 0.9, and learning rate of 0.01, with the Adam optimizer.

We conduct our experiments with 3 datasets representing different downstream tasks: (1) Oxford Flowers 102 (Nilsback and Zisserman, 2008) (Flowers) with 1K training and 6K test samples, (2) CIFAR-100 (Krizhevsky et al., 2009) (CIFAR), with 50K training and 10K test samples, and (3) Dmlab (Zhai et al., 2019b) with 65K training and 23K test samples.

We use a GPU cluster (single machine) with 8 NVIDIA TITAN Xp for SHiFT. The system is configured to either use a single or all eight GPUs. For fine-tuning models, we use a different cluster with slightly more performant NVIDIA GeForce RTX 2080 Ti GPUs.

We evaluate the performance of the 3 queries Q2-Q4 from Table 4.1, with a focus on computational efficiency. Q1 is directly evaluated against the database, whereas Q5 uses the Task2Vec code to find the nearest benchmark task. Optimizing Q5 is left for future work. Whenever the SH algorithm is used, depending on the cost model and specified in the experiments, we set the budget and chunk size to be minimal according to Section 4.4.2. This ensures that the semantics of the queries are kept intact, i.e. the data is not sub-sampled.

End-to-end Performance We start by validating the end-to-end performance of SHiFT. Table 4.2 compares the runtimes of fine-tuning all the models to running and using the output of queries Q2-Q4 with SHiFT, with and without automatic optimization. The latter is referred to as "Baseline" in the table. Our cost model, which we validate later in this section, suggests not to use SH for the small Flowers dataset, which is why the SH optimization is enabled only for CIFAR and Dmlab. When it comes to accuracy, Albeit being up to 1.5 orders of magnitude faster (c.f., Table 4.2), Figure 4.13 shows that the queries manage to retrieve near-optimal models for all datasets (i.e., suffering from very small regret). Furthermore, the SH optimizations for CIFAR and Dmlab retain the query semantics, not affecting the accuracy over the baseline for each query. Finally, Figure 4.14 shows the runtime for incrementally running SHiFT on 10%

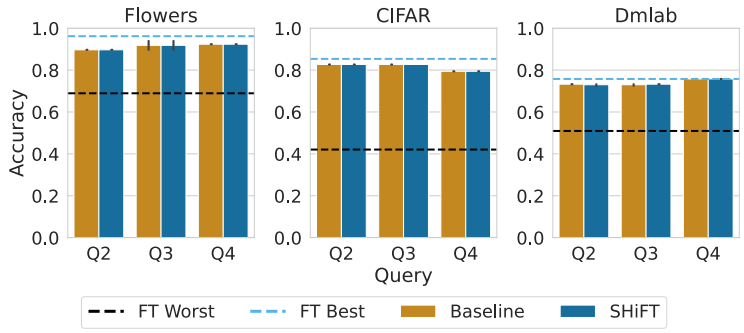


Figure 4.13: Fine-tune (FT) accuracy of returned model for all settings. The variance in black illustrates the min and max over 4 independent runs, showing some fluctuations for the small Flower dataset and the linear proxy which is sensitive to its hyper parameters. The baseline for each query represents the accuracy of using `SHiFT` without optimizations.

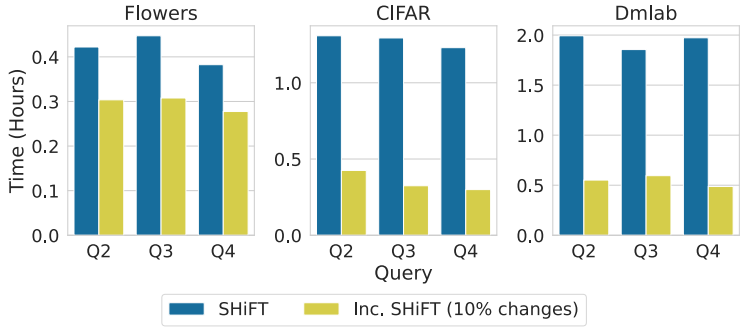
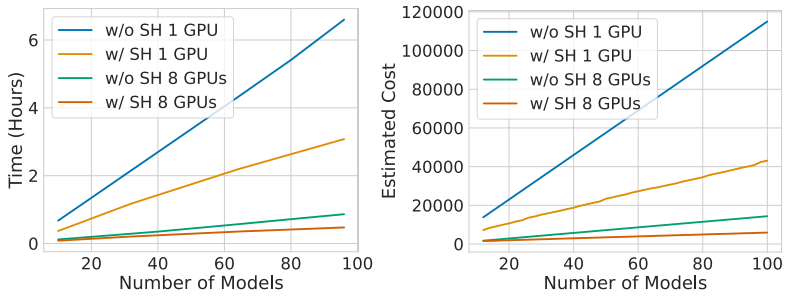


Figure 4.14: Incremental execution using `SHiFT` on 8 GPUs. 10% of the samples are randomly replaced.

randomly changed samples, leading to significant speedups for larger datasets, where the GPUs are fully utilized.

Scalability of `SHiFT` With the experimental setting described, we implicitly analyze the scaling behavior of `SHiFT` for an increasing number of GPUs (one and eight), and (training) samples (1K, 50K, and 62K). Increasing number of models is analysed in Figure 4.15 (left), where we deliberately chose a homogeneous setting of replicating the same model architecture multiple times.

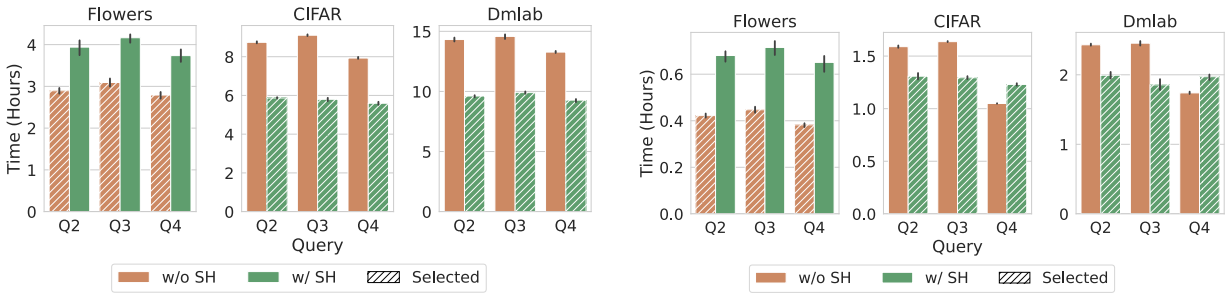
Cost Model: SH Trade-offs Figure 4.15 (right) validates the relative performance of our cost model for a set of homogeneous models. In Figure 4.16 we show the different runtimes with and without SH for 1 and 8 GPU and all queries along with the configuration picked by `SHiFT` on the 100 diverse models. On the Flowers dataset, our cost model accurately predicts the relative



(a) Execution times of Q2 for CIFAR.

(b) Cost model for Q2 on 50K training samples.

Figure 4.15: Increasing number of (homogeneous) models. The ResNet-101 V2 model is replicated for the experiments and the cost model.



(a) 1 GPU

(b) 8 GPUs

Figure 4.16: Execution times for all settings. The variance in black illustrates the min and max over 4 independent runs, showing little fluctuation. The hatched bar indicates the selected plan based on the cost model.

improvements to be expected when not using SH over using SH for a single (1.68x) and multiple GPUs (1.95x). On the larger datasets, CIFAR and Dmlab, the ratio for using 1 GPU (both 1.8x) is validated by our experiments. For multiple GPUs and the larger datasets, SHiFT predicts that SH should outperform non-SH by 1.2x, e.g. on CIFAR for all three queries, while the performance only matches Q2 and Q3, as visible in Figure 4.16. The hybrid query Q4 removes a very large model, however, our cost model overestimates the benefits of SH in such a case. The reason lies in the heterogeneity of the models (e.g., the largest model takes up almost 10% of the overall inference time) and the order of execution currently neglected in the cost model for both with and without SH on multiple GPUs. This explains the gaps visible in Figure 4.16b. Fusing these aspects into the cost model is left as future work.

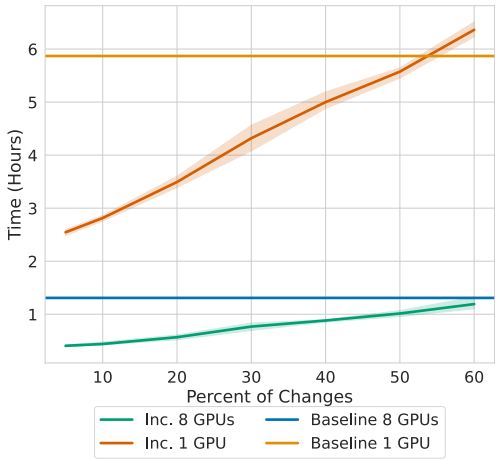


Figure 4.17: Execution time with respect to different number of feature changes for Q2 on CIFAR.

Incremental Execution In Figure 4.17, we randomly change a fixed percentage of the samples (i.e., manipulating the features) and plot the time required to perform an incremental execution of Q2 on CIFAR. Unsurprisingly, after a significant fraction of changes (e.g., > 50%), users might want to enforce a re-execution from scratch (i.e., by building a new initial reader instead of using a change-reader). However, when a small fraction of the samples are changed and the query is run incrementally, `SHiFT` offers a significant speedup over the baseline. The computational performance of adding data follows the same trend as changing data. Assessing the accuracy of such incremental query executions is left as future work, noting that the performance is heavily dataset- and distribution-dependent in such cases. The compute time for adding models to the query corresponds to the time needed to run a second independent query on these new models, due to the independence between the computation on the new models and the old ones.

4.5 Discussion

The focus of this chapter is transfer learning. It is important to notice though, that `SHiFT` can be used for various other tasks. As an example, the resulting proxy values of the queries Q2 and Q3, not their models, directly translate to the baseline and NN error (1 - acccuracy) used in Chapter 3. By designing our system and query language as generic and flexible as possible, we are confident that many other use-cases for `SHiFT` will emerge. We believe that this is specifically important in the are of data-centric AI, where *good* transformation (with respect to any measure), can be used to reduce the dimension of raw data, and thus enable well-established methods, currently failing to scale to high dimensional inputs.

4.6 Summary

We presented SHiFT, the first downstream task-aware search engine for transfer learning. Using our custom query language SHiFT-QL, users can generically define different model search strategies. Based on a cost-model, we automatically optimize prominent search queries and show significant speedups. Furthermore, by caching intermediate results, we allow our users to efficiently execute similar queries incrementally. In the future, we hope that SHiFT will enable researches to easily implement and evaluate newer search strategies.

Chapter 5

Ease.ML/CI: Continuous Integration of ML Models

In modern software engineering (Van Vliet et al., 2008), continuous integration (CI) is an important part of the best practice to systematically manage the life cycle of the development efforts. With a CI engine, the practice requires developers to integrate (i.e., commit) their code into a shared repository at least once a day (Duvall et al., 2007). Each commit triggers an automatic build of the code, followed by running a pre-defined test suite. The developer receives a `pass`/`fail` signal from each commit, which guarantees that every commit that receives a `pass` signal satisfies properties that are necessary for product deployment and/or presumed by downstream software.

Developing machine learning models is no different from developing traditional software, in the sense that it is also a full life cycle involving design, implementation, tuning, testing, and deployment. As machine learning models are used in more task-critical applications and are more

The content of this chapter are largely based on the following publication: (1) Renggli et al. (2019b). Continuous Integration of Machine Learning Models with ease.ml/ci: Towards a Rigorous Yet Practical Treatment. *SysML*. A running system has been demonstrated in (2) Renggli et al. (2019a). Ease.ml/ci and Ease.ml/meter in Action: Towards Data Management for Statistical Generalization. *VLDB Demo*. And a followup system work, contributed by the author of this thesis, but not part of this chapter, has been published in (3) Karlaš et al. (2020a). Building Continuous Integration Services for Machine Learning. *KDD*.

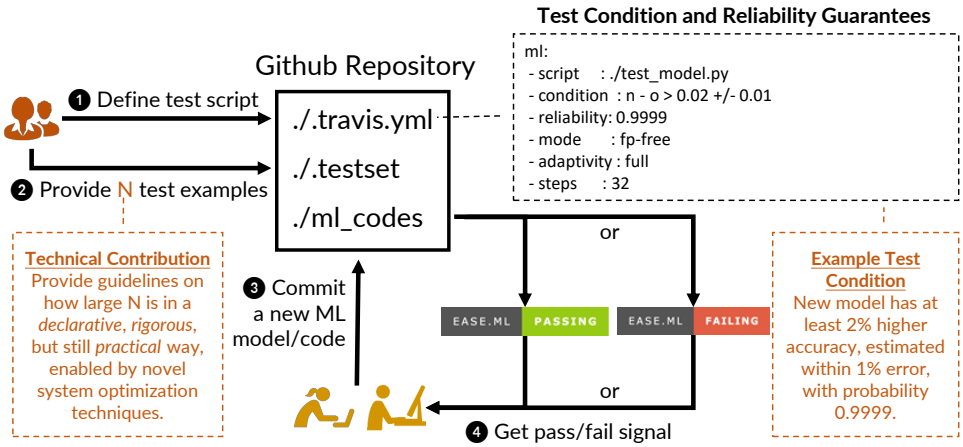


Figure 5.1: The workflow of `Ease.ML/CI`.

tightly integrated with traditional software stacks, it becomes increasingly important for the ML development life cycle also to be managed following systematic, rigid engineering discipline.

In this chapter, we describe the steps required to build, to our best knowledge, the first continuous integration system for machine learning with statistical guarantees. The workflow largely follows the traditional CI systems (Figure 5.1), while the system allows the user to define machine-learning specific test conditions such as *the new model can only change at most 10% predictions of the old model* or *the new model must have at least 1% higher accuracy than the old model.* After each commit of a machine learning model/program, the system automatically tests whether these test conditions hold, and return a `pass/fail` signal to the developer. Unlike traditional CI, CI for machine learning is inherently *probabilistic*. As a result, all test conditions are evaluated with respect to a (ϵ, δ)-reliability requirement from the user, where $1 - \delta$ (e.g., 0.9999) is the probability of a valid test and ϵ is the error tolerance (i.e., the length of the $(1 - \delta)$-confidence interval). The goal of the CI engine is to return the `pass/fail` signal that satisfies the (ϵ, δ)-reliability requirement.

Technical Challenge: Practicality At the first glance of the problem, there seems to exist a trivial implementation: For each committed model, draw N labeled data points from the testset, get an (ϵ, δ)-estimate of the accuracy of the new model, and test whether it satisfies the test conditions or not. The challenge of this strategy is the practicality associated with the label complexity (i.e., how large N is). To get an $(\epsilon = 0.01, \delta = 1 - 0.9999)$ estimate of a random variable ranging in $[0, 1]$, if we simply apply Hoeffding's inequality, we need more than 46K labels from the user (similarly, 63K labels for 32 models in a non-adaptive fashion and 156K labels in a fully adaptive fashion.

5.1 Contributions

In this chapter, we make contributions from both the system and machine learning perspectives.

1. **System Contributions.** We propose a novel system architecture to support a new functionality compensating state-of-the-art ML systems. Specifically, rather than allowing users to compose adhoc, free-style test conditions, we design a domain specific language that is more restrictive but expressive enough to capture many test conditions of practical interest.
2. **Machine Learning Contributions.** On the machine learning side, we develop simple, but novel techniques to optimize for test conditions that can be expressed within the domain-specific language that we designed. Our techniques cover different modes of interaction (fully adaptive, non-adaptive, and hybrid), as well as many popular test conditions that industrial and academic partners found useful. For a subset of test conditions, we are able to achieve up to two orders of magnitude savings on the number of labels that the system requires.

Beyond these specific technical contributions, conceptually, this chapter illustrates that enforcing and monitoring an ML development life cycle in a rigorous way *does not need to be expensive*. Therefore, ML systems in the near future could afford to support more sophisticated monitoring functionality to enforce the "right behavior" from the developer.

This chapter starts by describing related work in Section 5.2, then proceeds by presenting the design of `Ease.ML/CI` in Section 5.3. We then show estimation techniques that can lead to strong probabilistic guarantees using test datasets with moderate labeling effort. We present the basic implementation in Section 5.4 and more advanced optimizations in Section 5.5. We further verify the correctness and effectiveness of our estimation techniques via an experimental evaluation (c.f., Section 5.6). We conclude with a discussion and a summary of this chapter in Section 5.7 and 5.8.

5.2 Background and Related Work

Continuous integration is a popular concept in software engineering (Duvall et al., 2007). Nowadays, it is one of the best practices that most, if not all, industrial development efforts follow. The emerging requirement of a CI engine for ML has been discussed informally in multiple blog posts and forum discussions (Lara, 2017; Tran, 2017; Stojnic, 2018a; Lara, 2018; Stojnic, 2018b). However, despite being promising attempts, none of these discussions produce any rigorous solutions to testing the quality of a machine learning model, which arguably is the most important aspect of a CI engine for ML. This chapter is motivated by the success of CI in industry, and aims for building the first prototype system for rigorous integration of machine learning models.

The baseline implementation of `Ease.ML/CI` builds on intensive previous work on generalization and adaptive analysis. The non-adaptive version of the system is based on simple concentration inequalities (Boucheron et al., 2013) and the fully adaptive version of the system is inspired by Ladder (Blum and Hardt, 2015). Comparing to the second, `Ease.ML/CI` is less restrictive on the feedback and more expressive given the specification of the test conditions. This leads to a higher number of test samples needed in general. It is well-known that the $O(1/\epsilon^2)$ sample complexity of Hoeffding's inequality becomes $O(1/\epsilon)$ when the variance of the random variable σ^2 is of the same order of ϵ (Boucheron et al., 2013). In this chapter, we develop techniques to adapt the same observation to a real-world scenario (c.f., Section 5.5). The technique of only labeling the difference between models is inspired by disagreement-based active learning (Hanneke, 2007), which illustrates the potential of taking advantage of the disagreement regions between models to decrease labeling complexity. In fact, the technique we develop implies that one can achieve $O(1/\epsilon)$ label complexity when the disagreement between two models is $p = O(\sqrt{\epsilon})$.

The key difference between `Ease.ML/CI` and a differential privacy approach (Dwork et al., 2014) for answering statistical queries lies in the optimization techniques we design. By knowing the structure of the queries we are able to considerably lower the number of samples needed.

Conceptually, this chapter is inspired by the seminal series of work by Langford (2005) and others (Kääriäinen and Langford, 2005) that illustrates the possibility for generalization bound to be practically tight. The goal of our work is to build a practical system to guide the user in employing complicated statistical inequalities and techniques to achieve *practical* label complexity.

5.3 System Design of Ease.ML/CI

We present the design of Ease.ML/CI in this section. We start by presenting the interaction model and workflow as illustrated in Figure 5.1. We then present the scripting language that enables user interactions in a declarative manner. We discuss the syntax and semantics of individual elements, as well as their physical implementations and possible extensions. We end up with two system utilities, a "sample size estimator" and a "new testset alarm," the technical details of which will be explained in Sections 5.4 and 5.5.

5.3.1 Interaction Model

Ease.ML/CI is a *continuous integration system* for machine learning. It supports a four-step workflow: (1) user describes test conditions in a *test configuration script* with respect to the quality of an ML model; (2) user provides N test examples where N is automatically calculated by the system given the configuration script; (3) whenever developer commits/checks in an updated ML model/program, the system triggers a build; and (4) the system tests whether the test condition is satisfied and returns a "pass/fail" signal to the developer. When the current testset loses its "statistical power" due to repetitive evaluation, the system also decides on when to request a new testset from the user. The old testset can then be released to the developer as a validation set used for developing new models.

We also distinguish between two teams of people: the integration team, who provides testset and sets the reliability requirement; and the development team, who commits new models. In practice, these two teams can be identical; however, we make this distinction for clarity, especially in the fully adaptive case. We call the integration team *the user* and the development team *the developer*.

5.3.2 An Ease.ML/CI Script

Ease.ML/CI provides a declarative way for users to specify requirements of a new machine learning model in terms of a set of test cases. Ease.ML/CI then compiles such specifications into a *practical* workflow to enable evaluation of test cases with rigorous theoretical guarantees. We present the design of the Ease.ML/CI scripting language, followed by its implementation as an extension to the `.travis.yml` format used by Travis CI.

Logical Data Model The core part of a `Ease.ML/CI` script is a user-specified condition for the continuous integration test. In the current version, such a condition is specified over three variables $\mathcal{V} = \{n, o, d\}$: (1) n, the accuracy of the new model; (2) o, the accuracy of the old model; and (3) d, the *percentage* of new predictions that are different from the old ones ($n, o, d \in [0, 1]$).

Syntax of a Condition To specify the condition, which will be tested by `Ease.ML/CI` whenever a new model is committed, the user makes use of the following grammar:

```
c   :- floating point constant
v   :- n | o | d
op1 :- + | -
op2 :- *
EXP :- v | v op1 EXP | EXP op2 c

cmp :- > | <
C   :- EXP cmp c +/- c

F   :- C | C /\ F
```

`F` is the final condition, which is a conjunction of a set of clauses `C`. Each clause is a comparison between an expression over $\{n, o, d\}$ and a constant, with an error tolerance following the symbol `+/-`. For example, two expressions that we focus on optimizing can be specified as follows:

```
n - o > 0.02 +/- 0.01 /\ d < 0.1 +/- 0.01
```

in which the first clause

```
n - o > 0.02 +/- 0.01
```

requires that the new model have an accuracy that is two points higher than the old model, with an error tolerance of one point, whereas the clause

```
d < 0.1 +/- 0.01
```

requires that the new model can only change 10% of the old predictions, with an error tolerance of 1%.

Semantics of Continuous Integration Tests Unlike traditional continuous integration, all three variables used in Ease.ML/CI, i.e., $\{n, o, d\}$, are *random variables*. As a result, the evaluation of an Ease.ML/CI condition is inherently *probabilistic*. There are two additional parameters that the user needs to provide, which would define the semantics of the test condition: (1) δ, the probability with which the test process is allowed to be incorrect, which is usually chosen to be smaller than 0.001 or 0.0001 (i.e., 0.999 or 0.9999 success rate); and (2) mode chosen from {fp-free, fn-free}, which specifies whether the test is *false-positive free* or *false-negative free*. The semantics are, with probability $1 - \delta$, the output of Ease.ML/CI is free of false positives or false negatives.

The notion of false positives or false negatives is related to the fundamental trade-off between the "type I" error and the "type II" error in statistical hypothesis testing. Consider

```
x < 0.1 +/- 0.01.
```

Suppose that the real *unknown* value of x is x^*. Given an estimator $\hat{x}$, which, with probability $1 - \delta$, satisfies

$$\hat{x} \in [x^* - 0.01, x^* + 0.01],$$

what should be the testing outcome of this condition? There are three cases:

1. When $\hat{x} > 0.11$, the condition should return False because, given $x^* < 0.1$, the probability of having $\hat{x} > 0.11 > x^* + 0.01$ is less than δ.
2. When $\hat{x} < 0.09$, the condition should return True because, given $x^* > 0.1$, the probability of having $\hat{x} < 0.09 < x^* - 0.01$ is less than δ.
3. When $0.09 < \hat{x} < 0.11$, the outcome cannot be determined: Even if $\hat{x} > 0.1$, there is no way to tell whether the real value x^* is larger or smaller than 0.1. In this case, the condition evaluates to Unknown.

The parameter mode allows the system to deal with the case that the condition evaluates to Unknown. In the fp-free mode, Ease.ML/CI treats Unknown as False (thus rejects the commit) to ensure that whenever the condition evaluates to True using $\hat{x}$, the same condition is always True for x^*. Similarly, in the fn-free mode, Ease.ML/CI treats Unknown as True (thus accepts the commit). The false positive rate (resp. false negative rate) in the fn-free (resp. fp-free) mode is specified by the error tolerance.

Adaptive vs. Non-adaptive Integration A prominent difference between `Ease.ML/CI` and traditional continuous integration system is that the statistical power of a test dataset will decrease when the result of whether a new model passes the continuous integration test is released to the developer. The developer, if she wishes, can adapt her next model to increase its probability to pass the test, as demonstrated by the recent work on adaptive analytics (Blum and Hardt, 2015; Dwork et al., 2015). As we will see, ensuring probabilistic guarantees in the adaptive case is more expensive as it requires a larger testset. `Ease.ML/CI` allows the user to specify whether the test is adaptive or not with a flag `adaptivity` (`full`, `none`, `firstChange`):

- If the flag is set to `full`, `Ease.ML/CI` releases whether the new model passes the test immediately to the developer.
- If the flag is set to `none`, `Ease.ML/CI` accepts all commits, however, sends the information of whether the model really passes the test to a user-specified, third-party, email address that the developer does not have access to.
- If the flag is set to `firstChange`, `Ease.ML/CI` allows full adaptivity before the first time that the test passes (or fails), but stops afterwards and requires a new testset (see Section 5.4 for more details).

Example Scripts A `Ease.ML/CI` script is implemented as an extension to the `.travis.yml` file format used in Travis CI by adding an `ml` section. For example,

```
ml:
  - script     : ./test_model.py
  - condition  : n - o > 0.02 +/- 0.01
  - reliability: 0.9999
  - mode       : fp-free
  - adaptivity : full
  - steps      : 32
```

This script specifies a continuous test process that, with probability larger than 0.9999, accepts the new commit only if the new model has two points higher accuracy than the old one. This estimation is conducted with an estimation error within one accuracy point in a "false-positive free" manner. The system will release the `pass`/`fail` signal immediately to the developer, and the user expects that the given testset can be used by as many as 32 times before a new testset has to be provided to the system.

Similarly, if the user wants to specify a non-adaptive integration process, she can provide a script as follows:

```
ml:
  - script     : ./test_model.py
  - condition  : d < 0.1 +/- 0.01
  - reliability: 0.9999
  - mode       : fp-free
  - adaptivity : none -> xx@abc.com
  - steps      : 32
```

It accepts each commit but sends the test result to the email address `xx@abc.com` after each commit. The assumption is that the developer does not have access to this email account and therefore, cannot adapt her next model.

Discussion and Future Extensions The current syntax of Ease.ML/CI is able to capture many use cases that our users find useful in their own development process, including to reason about the accuracy difference between the new and old models, and to reason about the amount of changes in predictions between the new and old models in the test dataset. In principle, Ease.ML/CI can support a richer syntax. We list some limitations of the current syntax that we believe are interesting directions for future work.

1. Beyond accuracy: There are other important quality metrics for machine learning that the current system does not support, e.g., F1-score, AUC score, adversarial robustness, etc. It is possible to extend the current system to accommodate these scores by replacing the Bennett's inequality with the McDiarmid's inequality, together with the sensitivity of F1-score and AUC score, or any robustness metric. In this new context, more optimizations, such as using stratified samples, are possible for skewed cases.
2. Ratio statistics: The current syntax of Ease.ML/CI intentionally leaves out division ("/") and it would be useful for a future version to enable *relative* comparison of qualities (e.g., accuracy, F1-score, etc.).
3. Order statistics: Some users think that order statistics are also useful, e.g., to make sure the new model is among top-5 models in the development history.

Furthermore, Ease.ML/CI is currently not designed to detect domain drift or concept shift. In principle, this problem could be thought of as a similar process of CI – instead of fixing the testset and testing multiple models, monitoring concept shift is to fix a single model and test its generalization over multiple testsets overtime.

The current version of `Ease.ML/CI` does not provide support for all these features. However, we believe that many of them can be supported by developing similar statistical techniques (see Sections 5.4 and 5.5).

5.3.3 System Utilities

In traditional continuous integration, the system often assumes that the user has the knowledge and competency to build the test suite all by herself. This assumption is too strong for `Ease.ML/CI`— among the current users of `Ease.ML/CI`, we observe that even experienced software engineers in large tech companies can be clueless on how to develop a proper testset for a given reliability requirement. One prominent contribution of `Ease.ML/CI` is a collection of techniques that provide practical, but rigorous, guidelines for the user to manage testsets: *How large does the testset need to be? When does the system need a new freshly generated testset? When can the system release the testset and "downgrade" it into a development set?* While most of these questions can be answered by experts based on heuristics and intuition, the goal of `Ease.ML/CI` is to provide systematic, principled guidelines. To achieve this goal, `Ease.ML/CI` provides two utilities that are not provided in systems such as Travis CI.

Sample Size Estimator This is a program that takes as input a `Ease.ML/CI` script, and outputs the number of examples that the user needs to provide in the testset.

New Testset Alarm This subsystem is a program that takes as input a `Ease.ML/CI` script as well as the commit history of machine learning models, and produces an alarm (e.g., by sending an email) to the user when the current testset has been used too many times and thus cannot be used to test the next committed model. Upon receiving the alarm, the user needs to provide a new testset to the system and can also release the old testset to the developer.

An impractical implementation of these two utilities is easy — the system alarms the user to request a new testset after every commit and estimates the testset size using the Hoeffding bound. However, this can result in testsets that require tremendous labeling effort, often infeasible.

What is "Practical?" The practicality is certainly user dependent. Nonetheless, from our experience working with different users, we observe that providing $30,000$ to $60,000$ labels for every 32 model evaluations seems reasonable for many users: $30,000$ to $60,000$ is what 2 to 4 engineers can label in a day (8 hours) at a rate of 2 seconds per label, and 32 model evaluations

imply (on average) one commit per day in a month. Under this assumption, the user only needs to spend one day per month to provide test labels with a reasonable number of labelers. If the user is not able to provide this amount of labels, a "cheap mode", where the number of labels per day is easily reduced by a factor 10x, is achieved for most of the common conditions by increasing the error tolerance by a single or two percentage points.

Therefore, to make `Ease.ML/CI` a useful tool for real-world users, these utilities need to be implemented in a more practical way. The technical contribution of `Ease.ML/CI` is a set of techniques that we will present next, which can reduce the number of samples the system requests from the user by up to two orders of magnitude.

5.4 Baseline Implementation

We describe the techniques to implement `Ease.ML/CI` for user-specified conditions in the most general case. The techniques that we use involve standard Hoeffding inequality and a technique similar to Ladder (Blum and Hardt, 2015) in the adaptive case. This implementation is general enough to support all user-specified conditions currently supported in `Ease.ML/CI`, however, it can be made more practical when the test conditions satisfy certain conditions. We leave optimizations for specific conditions to Section 5.5.

5.4.1 Sample Size Estimator for a Single Model

Estimator for a Single Variable One building block of `Ease.ML/CI` is the estimator of the number of samples one needs to estimate one variable (n, o, and d) to ϵ accuracy with $1-\delta$ probability. We construct this estimator using the standard Hoeffding bound.

A sample size estimator $T : \mathcal{V} \times [0,1]^3 \mapsto \mathbb{N}$ is a function that takes as input a variable, its dynamic range, error tolerance and success rate, and outputs the number of samples one needs in a testset. With the standard Hoeffding bound,

$$T(v, r_v, \epsilon, \delta) = \frac{-r_v^2 \ln \delta}{2\epsilon^2}$$

where r_v is the dynamic range of the variable v, ϵ the error tolerance, and $1-\delta$ the success probability.

Recall that we make use of the exact grammar used to define the test conditions.

Estimator for a Single Clause Given a clause C (e.g. $n - o > 0.01$) with a left-hand side expression Φ, a comparison operator `cmp` (> or <), and a right-hand side constant, the sample size estimator returns the number of samples one needs to provide an (ϵ, δ)-estimation of the left-hand side expression. This can be done with a trivial recursion:

1. $T(\texttt{EXP = c * v}, \epsilon, \delta) = T(v, r_v, \epsilon/c, \delta)$, where c is a constant. We have $T(\texttt{c * v}, \epsilon, \delta) = \frac{-c^2 r_v^2 \ln \delta}{2\epsilon^2}$.
2. $T(\texttt{EXP1 + EXP2}, \epsilon, \delta) = \max\{T(\texttt{EXP1}, \epsilon_1, \frac{\delta}{2}), T(\texttt{EXP2}, \epsilon_2, \frac{\delta}{2})\}$, where $\epsilon_1 + \epsilon_2 < \epsilon$. The same equality holds similarly for $T(\texttt{EXP1 - EXP2}, \epsilon, \delta)$.

Estimator for a Single Formula Given a formula F that is a conjunction over k clauses $C_1, ..., C_k$, the sample size estimator needs to guarantee that it can satisfy each of the clause C_i. One way to build such an estimator is

3. $T(\texttt{F} = C_1 \wedge \ldots \wedge C_k, \epsilon, \delta) = \max_i T(C_i, \epsilon, \frac{\delta}{k})$.

Example Given a formula F, we now have a simple algorithm for sample size estimation. For

```
F :- n - 1.1 * o > 0.01 +/- 0.01 /\ d < 0.1 +/- 0.01
```

the system solves an optimization problem:

$$T(\texttt{F}, \epsilon, \delta) = \min_{\substack{\epsilon_1+\epsilon_2=\epsilon \\ \epsilon_1,\epsilon_2\in[0,1]}} \max\{\frac{-\ln\frac{\delta}{4}}{2\epsilon_1^2}, \frac{-1.1^2\ln\frac{\delta}{4}}{2\epsilon_2^2}, \frac{-\ln\frac{\delta}{2}}{2\epsilon^2}\}.$$

5.4.2 Non-Adaptive Scenarios

In the non-adaptive scenario, the system evaluates H models, without releasing the result to the developer. The result can be released to the user (the integration team).

Sample Size Estimation Estimation of sample size is easy in this case because all H models are independent. With probability $1 - \delta$, `Ease.ML/CI` returns the right answer for each of the H models, the number of samples one needs for formula F is simply $T(\texttt{F}, \epsilon, \frac{\delta}{H})$. This follows from the standard union bound. Given the number of models that user hopes to evaluate (specified in the `steps` field of a `Ease.ML/CI` script), the system can then return the number of samples in the testset.

New Testset Alarm The alarm for users to provide a new testset is easy to implement in the non-adaptive scenario. The system maintains a counter of how many times the testset has been used. When this counter reaches the pre-defined budget (i.e., `steps`), the system requests a new testset from the user. In the meantime, the old testset can be released to the developer for future development process.

5.4.3 Fully-Adaptive Scenarios

In the fully-adaptive scenario, the system releases the test result (a single bit indicating pass/fail) to the developer. Because this bit leaks information from the testset to the developer, one cannot use union bound anymore as in the non-adaptive scenario.

A trivial strategy exists for such a case — for every model, use a different testset. In this case, the number of samples required is $H \cdot T(\mathtt{F}, \epsilon, \frac{\delta}{H})$. This can be improved by applying a adaptive argument similar to Ladder (Blum and Hardt, 2015) as follows.

Sample Size Estimation For the fully adaptive scenario, `Ease.ML/CI` uses the following way to estimate the sample size for an H-step process. The intuition is simple. Assume that a developer is deterministic or pseudo-random, her decision on the next model only relies on all the previous `pass`/`fail` signals and the initial model H_0. For H steps, there are only 2^H possible configurations of the past `pass`/`fail` signals. As a result, one only needs to enforce the union bound on all these 2^H possibilities. Therefore, the number of samples one needs is $T(\mathtt{F}, \epsilon, \frac{\delta}{2^H})$.

Is the Exponential Term too Impractical? The improved sample size $T(\mathtt{F}, \epsilon, \frac{\delta}{2^H})$ is much smaller than the one, $H \cdot T(\mathtt{F}, \epsilon, \frac{\delta}{H})$, required by the trivial strategy. Readers might worry about the dependency on H for the fully adaptive scenario. However, for H that is not too large, e.g., $H = 32$, the above bound can still lead to practical number of samples as the $\frac{\delta}{2^H}$ is within a logarithm term. As an example, consider the following simple condition:

```
F :- n > 0.8 +/- 0.05.
```

With $H = 32$, we have

$$T(\mathtt{F}, \epsilon, \frac{\delta}{2^H}) = \frac{\ln 2^H - \ln \delta}{2\epsilon^2}.$$

Take $\delta = 0.0001$ and $\epsilon = 0.05$, we have $T(\mathtt{F}, \epsilon, \frac{\delta}{2^H}) = 6,279$. Assuming the developer checks in the best model everyday, this means that every month the user needs to provide only fewer than seven thousand test samples, a requirement that is realistic. However, if $\epsilon = 0.01$, this blows up

to 156, 955, which is less practical. We will show how to tighten this bound in Section 5.5 for a sub-family of test conditions.

New Testset Alarm Similar to the non-adaptive scenario, the alarm for requesting a new testset is trivial to implement — the system requests a new testset when it reaches the pre-defined budget. At that point, the system can release the testset to the developer for future development.

5.4.4 Hybrid Scenarios

One can obtain a better bound on the number of required samples by constraining the information being released to the developer. Consider the following scenario:

1. If a commit fails, returns `Fail` to the developer;
2. If a commit passes, (1) returns `Pass` to the developer, and (2) triggers the new testset alarm to request a new testset from the user.

Compared with the fully adaptive scenario, in this scenario, the user provides a new testset immediately after the developer commits a model that passes the test.

Sample Size Estimation Let H be the maximum number of steps the system supports. Because the system will request a new testset immediately after a model passes the test, it is not really adaptive: As long as the developer continues to use the same testset, she can assume that the last model always fails. Assume that the user is a deterministic function that returns a new model given the past history and past feedback (a stream of `Fail`), there are only H possible states that we need to apply union bound. This gives us the same bound as the non-adaptive scenario: $T(\mathtt{F}, \epsilon, \frac{\delta}{H})$.

New Testset Alarm Unlike the previous two scenarios, the system will alarm the user whenever the model that she provides passes the test or reaches the pre-defined budget H, whichever comes earlier.

Discussion It might be counter-intuitive that the hybrid scenario, which leaks information to the developer, has the same sample size estimator as the non-adaptive case. Given the maximum number of steps that the testset supports, H, the hybrid scenario cannot always finish all H steps

as it might require a new testset in $H' \ll H$ steps. In other words, in contrast to the fully adaptive scenario, the hybrid scenario accommodates the leaking of information not by adding more samples, but by decreasing the number of steps that a testset can support.

The hybrid scenario is useful when the test is hard to pass or fail. For example, imagine the following condition:

```
F :- n - o > 0.1 +/- 0.01
```

That is, the system only accepts commits that increase the accuracy by 10 accuracy points. In this case, the developer might take many developing iterations to get a model that actually satisfies the condition.

5.4.5 Evaluation of a Condition

Given a testset that satisfies the number of samples given by the sample size estimator, we obtain the estimates of the three variables used in a clause, i.e., $\hat{n}$, $\hat{o}$, and $\hat{d}$. Simply using these estimates to evaluate a condition might cause both false positives and false negatives. In `Ease.ML/CI`, we instead replace the point estimates by their corresponding confidence intervals, and define a simple algebra over intervals (e.g., $[a, b] + [c, d] = [a + c, b + d]$), which is used to evaluate the left-hand side of a single clause. A clause still evaluates to {`True`, `False`, `Unknown`}. The system then maps this three-value logic into a two-value logic given user's choice of either `fp-free` or `fn-free`.

5.4.6 Use Cases and Practicality Analysis

The baseline implementation of `Ease.ML/CI` relies on standard concentration bounds with simple, but novel, twists to the specific use cases. Despite its simplicity, this implementation can support real-world scenarios that many of our users find useful. We summarize five use cases and analyze the number of samples required from the user. These use cases are summarized from observing the requirements from the set of users we have been supporting over the last two years, ranging from scientists at multiple universities, to real production applications provided by high-tech companies. (`[c]` and `[epsilon]` are placeholders for constants.)

(F1: Lower Bound Worst Case Quality)

```
F1         :- n > [c] +/- [epsilon]
adaptivity :- none
mode       :- fn-free
```

This condition is used for quality control to avoid the cases that the developer accidentally commits a model that has an unacceptably low quality or has obvious quality bugs. We see many use cases of this condition in non-adaptive scenario, most of which need to be false-negative free.

(F2: Incremental Quality Improvement)

```
F2         :- n - o > [c] +/- [epsilon]
adaptivity :- full
mode       :- fp-free
([c] is small)
```

This condition is used for making sure that the machine learning application monotonically improves over time. This is important when the machine learning application is end-user facing, in which it is unacceptable for the quality to drop. In this scenario, it makes sense for the whole process to be fully adaptive and false-positive free.

(F3: Significant Quality Milestones)

```
F3         :- n - o > [c] +/- [epsilon]
adaptivity :- firstChange
mode       :- fp-free
([c] is large)
```

This condition is used for making sure that the repository only contains significant quality milestones (e.g., log models after 10 points of accuracy jump). Although the condition is syntactically the same as `F2`, it makes sense for the whole process to be hybrid adaptive and false-positive free.

(F4: No Significant Changes)

```
F4         :- d < [c] +/- [epsilon]
adaptivity :- full | none
mode       :- fn-free
([c] is large)
```

This condition is used for safety concerns similar to `F1`. When the machine learning application is end-user facing or part of a larger application, it is important that its prediction will not change significantly between two subsequent versions. Here, the process needs to be false-negative free. Meanwhile, we see use cases for both fully adaptive and non-adative scenarios.

(F5: Compositional Conditions)

```
F5 :- F4 /\ F2
```

1-δ	ϵ	F1, F4		F2, F3	
		none	full	none	full
0.99	0.1	404	1340	1753	5496
0.99	0.05	1615	5358	7012	21984
0.99	0.025	6457	21429	28045	87933
0.99	0.01	40355	133930	175282	549581
0.999	0.1	519	1455	2214	5957
0.999	0.05	2075	5818	8854	23826
0.999	0.025	8299	23271	35414	95302
0.999	0.01	51868	145443	221333	595633
0.9999	0.1	634	1570	2674	6417
0.9999	0.05	2536	6279	10696	25668
0.9999	0.025	10141	25113	42782	102670
0.9999	0.01	63381	156956	267385	641684
0.99999	0.1	749	1685	3135	6878
0.99999	0.05	2996	6739	12538	27510
0.99999	0.025	11983	26955	50150	110038
0.99999	0.01	74894	168469	313437	687736

Figure 5.2: Number of samples required by different conditions, $H = 32$ steps. Red font indicates "impractical" number of samples (see discussion on practicality in Section 5.3.3).

One of the most popular test conditions is a conjunction of two conditions, `F4` and `F2`: The integration team wants to use `F4` and `F2` together so that the end-user facing application will not experience dramatic quality change.

Practicality Analysis How practical is it for our baseline implementation to support these conditions, and in which case that the baseline implementation becomes impractical?

When is the Baseline Implementation Practical? The baseline implementation, in spite of its simplicity, is practical in many cases. Figure 5.2 illustrates the number of samples the system requires for $H = 32$ steps. We see that, for both `F1` and `F4`, all adaptive strategies are practical up to 2.5 accuracy points, while for `F2` and `F3`, the non-adaptive and hybrid adaptive strategies are practical up to 2.5 accuracy points and the fully adaptive strategy is only practical up to 5 accuracy points. As we see from this example, even with a simple implementation, *enforcing a rigorous guarantee for CI of machine learning is not always expensive!*

When is the Baseline Implementation Not Practical? We can see from Figure 5.2 the strong dependency on ϵ. This is expected because of the $O(1/\epsilon^2)$ term in the Hoeffding inequality. As a result, none of the adaptive strategy is practical up to 1 accuracy point, a level of tolerance that is important for many task-critical applications of machine learning. It is also not surprising that the fully adaptive strategy requires more samples than the non-adaptive one, and therefore becomes impractical with higher error tolerance.

5.5 Optimizations

As we saw from the previous sections, the baseline implementation of `Ease.ML/CI` fails to provide a practical approach for low error tolerance and/or fully adaptive cases. In this section, we describe optimizations that allow us to further improve the sample size estimator.

High-level Intuition All of our proposed techniques in this section are based on the same intuition: Tightening the sample size estimator in the worst case is hard to get better than $O(1/\epsilon^2)$; instead, we take the classic system way of thinking — *improve the sample size estimator for a sub-family of popular test conditions*. Accordingly, `Ease.ML/CI` applies different optimization techniques for test conditions of different forms.

Technical Observation 1 The intuition behind a tighter sample size estimator relies on standard techniques of tightening Hoeffding's inequality for variables with small variance. Specifically, when the new model and the old model is only different on up to $(100 \times p)\%$ of the predictions, which could be part of the test condition anyway, for data point i, the random variable $n_i - o_i$ has small variance: $\mathbb{E}\left[(n_i - o_i)^2\right] < p$, where n_i and o_i are the predictions of the new and old models on the data point i. This allows us to apply the standard Bennett's inequality.

Proposition 5.5.1 (Bennett's inequality)**.** *Let $X_1, ..., X_n$ be independent and square integrable random variables such that for some nonnegative constant b, $|X_i| \leq b$ almost surely for all $i < n$. We have*

$$\Pr\left[\left|\frac{\sum_i X_i - \mathbb{E}[X_i]}{n}\right| > \epsilon\right] \leq 2\exp\left(-\frac{v}{b^2}h\left(\frac{nb\epsilon}{v}\right)\right),$$

where $v = \sum_i \mathbb{E}\left[X_i^2\right]$ and $h(u) = (1+u)\ln(1+u) - u$ for all positive u.

Technical Observation 2 The second technical observation is that, to estimate the difference of predictions between the new model and the old model, one does not need to have labels. Instead, a sample from the unlabeled dataset is enough to estimate the difference. Moreover, to estimate $n - o$ when only 10% data points have different predictions, one only needs to provide labels to 10% of the whole testset.

5.5.1 Pattern 1: Difference-based Optimization

The first pattern that `Ease.ML/CI` searches in a formula is whether it is of the following form

```
d < A +/- B /\ n - o > C +/- D
```

which constrains the amount of changes that a new model is allowed to have while ensuring that the new model is no worse than the old model. These two clauses popularly appear in test conditions from our users: For production-level systems, developers start from an already good enough, deployed model, and spend most of their time *fine-tuning* a machine learning model. As a result, the continuous integration test must have an error tolerance as low as a single accuracy point. On the other hand, the new model will not be different from the old model significantly, otherwise more engaged debugging and investigations are almost inevitable.

Assumption One assumption of this optimization is that it is relatively cheap to obtain unlabeled data samples, whereas it is expensive to provide labels. This is true in many of the applications. When this assumption is valid, both optimizations outlined next can be applied to this pattern; otherwise, both optimizations still apply but will lead to improvement over only a subset.

Optimization 1: Hierarchical Testing The first optimization is to test the rest of the clauses conditioned on `d < A +/- B`, which leads to an algorithm with two-level tests. The first level tests whether the difference between the new model and the old model is small enough, whereas the second level tests $(n - o)$.

The algorithm runs in two steps:

1. **(Filter)** Get an $(\epsilon', \frac{\delta}{2})$-estimator $\hat{d}$ with n' samples. Test whether $\hat{d} > A + \epsilon'$: If so, returns `False`;

2. **(Test)** Test `F` as in the baseline implementation (with $1 - \frac{\delta}{2}$ probability), conditioned on $d < A + 2\epsilon'$.

It is not hard to see why the above algorithm works — the first step only requires unlabeled data points and does not need human intervention. In the second step, conditioned on $d < p$, we know that $\mathbb{E}\left[(n_i - o_i)^2\right] < p$ for each data point. Combined with $|n_i - o_i| < 1$, applying Bennett's inequality we have $\Pr\left[\left|\widehat{n-o} - (n-o)\right| > \epsilon\right] \leq 2\exp(-nph\left(\frac{\epsilon}{p}\right))$.

As a result, the second step needs a sample size (for non-adaptive scenario) of

$$\frac{\ln H - \ln \frac{\delta}{4}}{ph\left(\frac{\epsilon}{p}\right)}.$$

When $p = 0.1, 1-\delta = 0.9999, d < 0.1$, we only need 29K samples for 32 non-adaptive steps and 67K samples for 32 fully-adaptive steps to reach an error tolerance of a single accuracy point — 10× fewer than the baseline (Figure 5.2).

Optimization 2: Active Labeling The previous example gives the user a way to conduct 32 fully-adaptive fine-tuning steps with only 67K samples. Assume that the developer performs one commit per day, this means that we require 67K samples per month to support the continuous integration service.

One potential challenge for this strategy is that all 67K samples need to be labeled before the continuous integration service can start working. This is sometimes a strong assumption that many users find problematic. In the ideal case, we hope to interleave the development effort with the labeling effort, and amortize the labeling effort over time.

The second technique our system uses relies on the observation that, to estimate $(n-o)$, only the data points that have a different prediction between the new and old models need to be labeled. When we know that the new model predictions are only different from the old model by 10%, we only need to label 10% of all data points. It is easy to see that, every time when the developer commits a new model, we only need to provide

$$\frac{-\ln \frac{\delta}{4}}{ph\left(\frac{\epsilon}{p}\right)} \times p$$

labels. When $p = 0.1$ and $1 - \delta = 0.9999$, this implies 2188 labels for an error tolerance of a single accuracy point. If the developer commits one model per day, the labeling team only needs to label 2,188 samples the next day. Given a well designed interface that enables a labeling

throughput of 5 seconds per label, the labeling team only needs to commit 3 hours a day! For a team with multiple engineers, this overhead is often acceptable, considering the guarantee provided by the system down to a single accuracy point.

Notice that active labeling, in a non-adversary setting, assumes a stationary underlying distribution. One way to enforce this in the system is to ask the user to provide a pool of unlabeled data points at the same time, and then only ask for labels when needed. In this way, we do not need to draw new samples over time.

5.5.2 Pattern 2: Implicit Variance Bound

In many cases, the user does not provide an explicit constraint on the difference between a new model and an old model. However, many machine learning models are not so different in their predictions. Take AlexNet, ResNet, GoogLeNet, AlexNet (Batch Normalized), and VGG for example: When applied to the ImageNet testset, these five models, developed by the ML community since 2012, only produce up to 25% different answers for top-1 *correctness* and 15% different answers for top-5 *correctness*! For a typical workload of continuous integration, it is therefore not unreasonable to expect many of the consecutive commits would have smaller difference than these ImageNet winners involving years of development.

Motivated by this observation, `Ease.ML/CI` will automatically match with the following pattern

```
n - o > C +/- D.
```

When the unlabeled testset is cheap to get, the system will use one testset to estimate d up to $\epsilon = 2D$: For binary classification task, the system can use an unlabeled testset; for multi-class tasks, one can either test the *difference of predictions* on an unlabeled testset or *difference of correctness* on a labeled testset. This gives us an upper bound of $n - o$. The system then tests $n - o$ up to $\epsilon = D$ on *another* testset (different from the one used to test d). When this upper bound is small enough, the system will trigger similar optimization as in `Pattern 1`. Note that the first testset will be 16× smaller than testing $n - o$ directly up to $\epsilon = D$ — 4× due to a higher error tolerance, and 4× due to that d has 2× smaller range than $n - o$.

One caveat of this approach is that the system does not know how large the second testset would be before execution. The system uses a technique similar to active labeling by incrementally growing the labeled testset every time when a new model is committed, if necessary. Specifically, we optimize for test conditions following the pattern

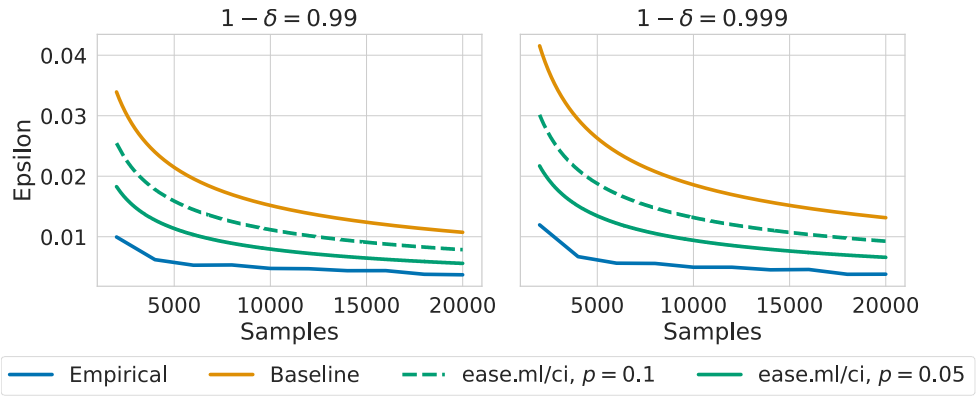


Figure 5.3: Comparison of Sample Size Estimators in the Baseline Implementation and the Optimized Implementation.

```
n > A +/- B,
```

when `A` is large (e.g., 0.9 or 0.95). This can be done by first having a coarse estimation of the lower bound of n, and then conducting a finer-grained estimation conditioned on this lower bound. Note that this can only introduce improvement when the lower bound is large (e.g., 0.9).

5.5.3 Tight Numerical Bounds

Following Langford (2005), having a test condition consisting of n i.i.d random variables drawn from a Bernoulli distribution, one can simply derive a tight bound on the number of samples required to reach a (ϵ, δ) accuracy. The calculation of number of samples require the probability mass function of the Binomial distribution (sum of i.i.d Bernoulli variables). Tight bound are solved by taking the minimum of number of samples n needed, over the max unknown true mean p. This technique can also be extended to more complex queries, where the binomial distribution has to be replaced by a multimodal distribution. The exact analysis has, as for the simple case, no closed-form solution, and deriving efficient approximations is left as further work.

5.6 Experiments

We focus on empirically validating the derived bounds and show `Ease.ML/CI` in action next.

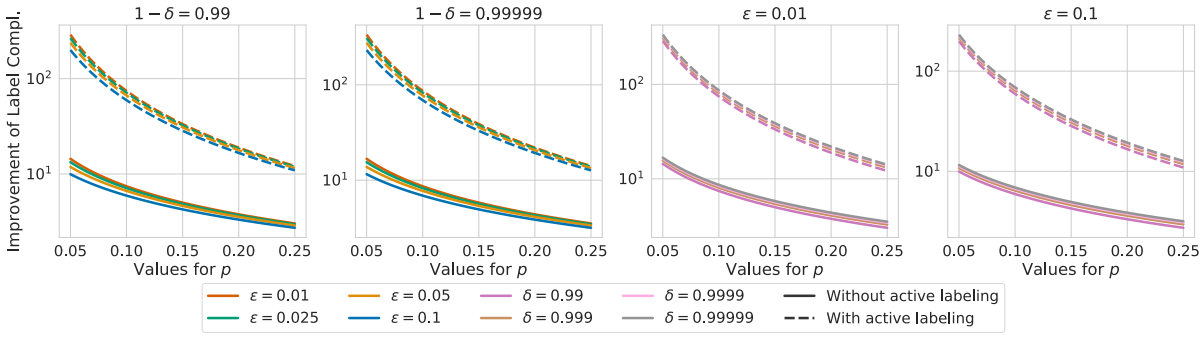


Figure 5.4: Impact of ϵ, δ, and p on the Label Complexity.

5.6.1 Sample Size Estimator

One key technique most of our optimizations relied on is that, by knowing an upper bound of the sample variance, we are able to achieve a tighter bound than simply applying the Hoeffding bound. This upper bound can either be achieved by using unlabeled data points to estimate the difference between the new and old models, or by using labeled data points but conducting a coarse estimation first. We now validate our theoretical bound and its impact on improving the label complexity.

Figure 5.3 illustrates the estimated error and the empirical error by *assuming* different upper bounds p, for a model with accuracy around 98%. We run GoogLeNet (Jia et al., 2014) on the infinite MNIST dataset (Bottou, 2016) and estimate the true accuracy c. Assuming a non-adaptive scenario, we obtain a range of accuracies achieved by randomly taking n data points. We then estimate the interval ϵ with the given number of samples n and probability $1 - \delta$. We see that, both the baseline implementation and `Ease.ML/CI` dominate the empirical error, as expected, while `Ease.ML/CI` uses significantly fewer samples.[1]

Figure 5.4 illustrates the impact of this upper bound on improving the label complexity. We see that the improvement increases significantly when p is reasonably small — when $p = 0.1$, we can achieve almost 10× improvement on the label complexity. Active labeling further increases the improvement, as expected, by another 10×.

[1]The empirical error was determined by taking different testsets (with the sample sample size) and measuring the gap between the δ and $1 - \delta$ quantiles over the observed testing accuracies.

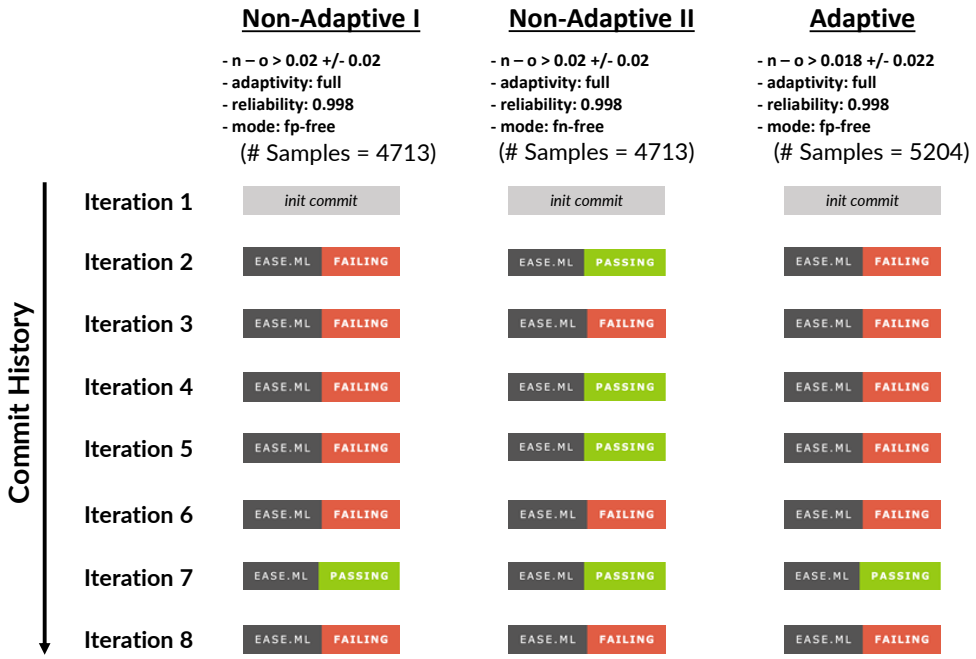


Figure 5.5: Continuous Integration Steps in `Ease.ML/CI`.

5.6.2 `Ease.ML/CI` in Action

We showcase three different test conditions for a real-world incremental development of machine learning models submitted to the SemEval-2019 Task 3 competition. The goal is to classify the emotion of the user utterance as one of the following classes: Happy, Sad, Angry or Others.[2] The eight models developed in an incremental fashion, and submitted in that exact order to the competition (finally reaching rank 29/165) are made available together with a corresponding description of each iteration via a public repository.[3] The test data, consisting of 5,509 items was published by the organizers of the competition after its termination. This represents a non-adaptive scenario, where the developer does not get any direct feedback whilst submitting new models.

Figure 5.5 illustrates three similar, but different test conditions, which are implemented in `Ease.ML/CI`. The first two conditions check whether the new model is better than the old one by at least 2 percentage points in a non-adaptive matter. The developer will therefore not get any direct feedback as it was the case during the competition. While query (I) does reject false pos-

[2]Competition website: `https://www.humanizing-ai.com/emocontext.html`
[3]Github repository: `https://github.com/zhaopku/ds3-emoContext`

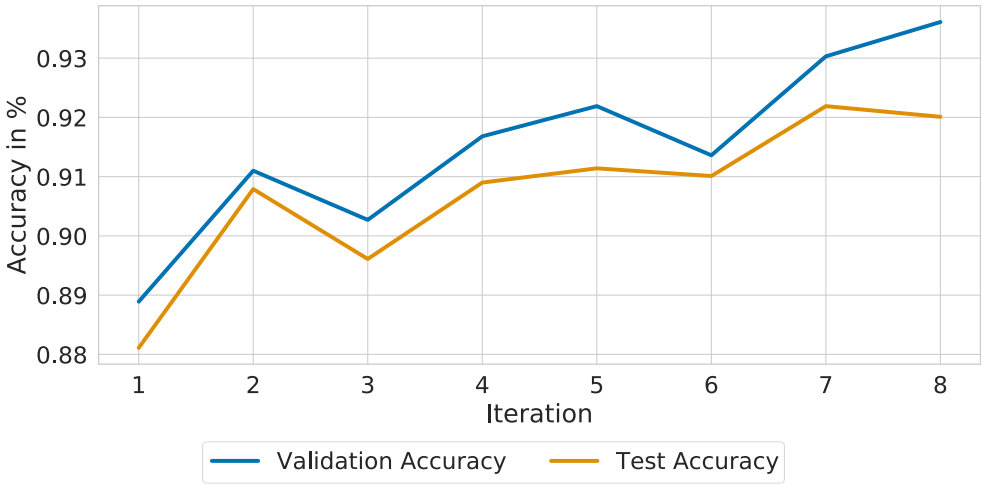


Figure 5.6: Evolution of Development and Test Accuracy.

itive, condition (II) does accept false negative. The third condition mimics the scenario where the user would get feedback after every commit without any false negative. All three queries were optimized by `Ease.ML/CI` using Pattern 2 and exploiting the fact that between any two submission there is no more than 10% difference in prediction.

Simply using Hoeffding's inequality does not lead to a practical solution — for $\epsilon = 0.02$ and $\delta = 0.002$, in $H = 7$ non-adaptive steps, one would need at lest

$$\frac{r_v^2(\ln H - \ln \frac{\delta}{2})}{2\epsilon^2} = 44,268$$

samples. This number even grows to up to 58K in the fully adaptive case!

All the queries can be supported rigorously with the 5.5K test samples provided after the competition. The first two conditions can be answered within two percentage point error tolerance and 0.998 reliability. The full-adaptive query in the third scenario can only achieve a 2.2 percentage point error tolerance, as the number of labels needed would be more than 6K, with the same error tolerance as in the first two queries.

We see that, in all three scenarios, `Ease.ML/CI` returns `pass/fail` signals that make intuitive sense. If we look at the evolution of the development and test accuracy over the eight iterations (c.f., Figure 5.6), the developer would ideally want `Ease.ML/CI` to accept her last commit, whereas all three queries will have the second last model chosen to be active, which correlates with the test accuracy evolution.

5.7 Discussion

With Ease.ML/CI, we provide a first-of-its-kind CI engine for ML models with strong statistical guarantees. Despite being published for quite a while, practitioners mainly keep on using classical CI engines, without replacing the test set or calculating confidence bounds. We believe the reason for this to be twofold: (a) no overfitting on large benchmark datasets, and (b) too conservative worst-case guarantees.

Overfitting on Benchmark Datasets Shortly after the initial Ease.ML/CI publication, Recht et al. (2019) showed in their work, that despite almost a decade of research on popular computer vision datasets such as ImageNet (Deng et al., 2009) with full access to the testset, models do not seem to overfitt on those specific datasets. This fact is surprising, given that thousands of researchers keep on trying to improve the state-of-the-art accuracy on these benchmark datasets. Nonetheless, one has to keep in mind that these researches are specialized in ML, being fully aware of the risks of overfitting. Practitioners new to ML could quickly ignore or forget this risk and push a seemingly better model to production.

Worst-Case Guarantees Along with the previous fact, our worst-case theoretical analysis models the developer as an adversarial player aiming at overfitting to the hidden test set. Luckily, this is rarely the case in practice. Modelling developers in a more realistic but still statistical sound way could offer the possibility to further reduce the number of test samples needed (or increase the number of supported evaluations) and thus make the proposed approach and system more suitable for a larger target audience in the future.

5.8 Summary

We presented Ease.ML/CI, a continuous integration system for machine learning. It provides a declarative scripting language that allows users to state a rich class of test conditions with rigorous probabilistic guarantees. We also studied the novel practicality problem in terms of labeling effort that is specific to testing machine learning models. Our techniques can reduce the amount of required testing samples by up to two orders of magnitude. We validated the soundness of our techniques, and showcased their applications in real-world scenarios.

Chapter 6

Conclusions and Future Work

MLOps has gained substantial interest and has become an active field of research over the past years. Motivated by the success of DevOps to guarantee fast and robust lifecycles in classical software development and deployment, practitioners in industry rely on best-practices and tools to achieve the same goal when developing and deploying ML artifacts to production. However, a variety of new online resources and books summarizing these aspects, often neglect the underlying fundamentals of MLOps. The inherent randomness of ML tasks and its finite-sample data dependency must be taking into account, in order for MLOps to become a principled way of handling the lifecycle management and validating ML applications in a reliable and statistically rigorous manner

6.1 Summary

ML as a workload, requires us to rethink well-established data management quantities (e.g., data quality), or software engineering principles in general (e.g., feasibility study). Furthermore, blindly applying existing DevOps techniques, such as continuous integration, to ML models can lead to unexpected failures.

In this thesis, we took a first step towards building novel data-centric systems for MLOps by answering three specific questions:

1. How to perform a theoretically founded feasibility study for ML?
2. How to search which model to fine-tune in a generic and efficient way?
3. How to re-use a fixed test dataset while keeping strong statistical guarantees?

We answer the first question in Chapter 3. The main result is `Ease.ML/Snoopy`, a system that supports the data scientist in assessing whether the data artifacts meet the quality requirements to train high accuracy models. By using pre-trained models to estimate the Bayes error rate and understanding the theoretical properties of given estimator, we are able to quantify the false positive and false negative regimes of `Ease.ML/Snoopy`, allowing for additional guidance to the users of our system.

The second question led to the definition of `SHiFT`, the first downstream task-aware search engine for pre-trained models presented in Chatper 4. To efficiently enable transfer learning, users can run their custom model search queries against our system using the `SHiFT-QL` query language. A cost-model aware decision maker will efficiently return the best models to the users, which then after fine-tuning lead to high accuracy models, even for small downstream datasets.

Understanding the last question was crucial for building `Ease.ML/CI` in Chapter 5. Our system uses statistical estimators to enable safe continuous integration of models into production. These estimators are optimized for specific regimes that we care about by inspecting the structure of custom model requirements encoded with a declarative scripting language.

Answering each of these questions and building system making use of these answers required substantial research and publications across two major field: (1) machine learning and (2) data management. Due to the lack of communities and conferences at the intersection between the two field, with MLSys being a prominent exception, researchers in systems for ML development and operations are currently "forced" to know the works and contribute to two research communities simultaneously. Getting to know two different areas and managing to pick the best of both worlds can certainly be challenging, but also enriching and rewarding. It is very likely that systems for MLOps researchers in the near future are required to extend the viewpoints to other areas such as human–computer interactions (e.g., CHI) for those specific aspects, where human decisions and feedback plays an important role during the ML lifecycle.

6.2 Research Outlook

By providing answers to these three questions, along with systems to make use of these answers and support ML development and operations workloads, we have only started scratching the surface of this promising research area. Many other data management aspects during the development phase need to be carefully defined and analyzed in a theoretically founded and principled way specifically for an ML workload (e.g., data cleaning, data collection, data privacy, or data

summarisation). The same holds for the operations phase, where requirements such as deployment with latency and bandwidth requirements, or how to monitor data drift and concept shift are largely underexplored.

In Chapter 3, we tackled a decades old problem by making use of new pre-trained embeddings available online. This seemingly simple idea, together with a theoretical understanding, allows us to apply well-established techniques on top of these transformations. We are confident, that this idea can be transferred to other applications in the future. While focusing on transfer learning when building `SHiFT` in Chapter 4, we designed the system and query language such that it can also easily support the `Ease.ML/Snoopy` workload. Furthermore, any alternative idea relying on pre-trained embeddings should be easily implementable and supported in the future.

All three main chapters of this thesis share some common limitations, which represent intriguing future research. First, the standard assumption of having datasets sampled i.i.d. from the same distribution quickly falls apart in practice. Samples are either intentionally (e.g., to ensure fairness by not under-representing minorities) or unintentionally not i.i.d (e.g., when collecting a sequence of satellite images). Moreover, techniques such as weak supervision (Ratner et al., 2017) allow rapid train set creation, but inevitably introduce some gap between the train and the test distribution. In addition, the only metric of interest in this thesis is accuracy. Maximizing the accuracy of a model is not always the only goal. Practitioners rather care about other alternative metrics such as F1 score or AUC, or aim at maximizing accuracy under other model constraints such as calibration. Extending all techniques proposed in this thesis to arbitrary metrics of interest, or combinations of them, can be fruitful for a wider application of the systems.

Finally, understanding the ML development and operations process for even more complex model requirements such as fairness, interpretability, or explainability, which are not that easily measurable, is not only desired for safety-critical applications, it might even be soon required by law. Yet, even before testing for such requirements, the question of how to encode them in a flexible manner remains an open challenge. We hope that our approach in Chapter 5 can serve as inspiration for this.

Chapter A

Supplementary Material – Proofs

A.1 Proofs for kNN over Feature Transformations

A.1.1 Safe Transformations

In this section we prove the claims from Section 3.5.1, in the same order as stated there. Even though the main part of the thesis is considering $\mathcal{Y} = \{0, 1\}$ only, which is the usual setting when convergence rates of a kNN classifier are discussed, we suppose that $\mathcal{Y} = \{1, \dots, C\}$, for some integer $C \geq 2$ for the analysis of safe transformations.

Safe Transformations via Injectivity

We discuss injective functions in Section 3.5.1, and provide examples of safe transformations that arise from injectivity, such as $x \mapsto (x, f(x))$, for any map f, and $x \mapsto (x^+, x^-)$. We will now prove this by providing a sufficient condition for a function to be δ-safe. We call this condition *δ-injectivity*.

Definition A.1.1. *Let $(\mathcal{X}, \mathcal{A}, p)$ and be a finite probability space, and let $(\widetilde{\mathcal{X}}, \tilde{\mathcal{A}})$ be a finite measurable space. We say that a measurable function $f\colon \mathcal{X} \to \widetilde{\mathcal{X}}$ is* δ-injective *if there exists a subset $I_{\mathcal{X}}(f) \subseteq \mathcal{X}$ on which f is injective, and that satisfies*

$$p(I_{\mathcal{X}}(f)) \geq 1 - \delta.$$

Lemma A.1.2. *Let $f_i\colon \mathcal{X} \to \widetilde{\mathcal{X}}_i$, for $i = 0,\dots,n$, be functions such that there exist $I_{\mathcal{X}}(f_0),\dots,I_{\mathcal{X}}(f_n) \subseteq \mathcal{X}$, sets on which $f_0,\dots,f_n$ are injective, respectively, and such that*

$$p\left(\bigcup_{i=0}^{n} I_{\mathcal{X}}(f_i)\right) \geq (1-\delta).$$

Then $(f_0,\dots,f_n)\colon \mathcal{X} \to \prod_{i=0}^{n} \widetilde{\mathcal{X}}_i$ is δ-safe.

In particular, if f_0 is δ-injective, then $(f_0, f_1,\dots,f_n)$ is δ-safe.

Proof: We first prove the claim for $n = 0$ and then extend it to $n \in \mathbb{N}$.

(1) $n = 0$:

In this case, we ought to prove that if f is δ-injective, then f is δ-safe. Let $I_{\mathcal{X}}(f)$ be a set on which f is injective and that satisfies $p(I_{\mathcal{X}}(f)) \geq 1-\delta$. Motivated by Equation 3.5.5, we define $\mathcal{X}_l := \{x \in \mathcal{X}\colon y_x \neq y_{f(x)}\}$. With the above definition, note that Equation 3.5.5 can be reduced to

$$\Delta^{*}_{f,X} = \mathbb{E}_{x\sim X}\left[\left(p(y_x|x) - p_{f^{-1}}(y_{f(x)}|f(x))\right)\cdot 1\{x \in \mathcal{X}_l\}\right] \leq p(\mathcal{X}_l). \tag{A.1.1}$$

We will now modify $I_{\mathcal{X}}(f)$ to get $I_{\mathcal{X}_l}$ that is of the same mass, and is disjoint from $\mathcal{X}_l$. If $\mathcal{X}_l \cap I_{\mathcal{X}}(f) = \emptyset$, then we are done. Therefore, let $x_l \in \mathcal{X}_l \cap I_{\mathcal{X}}(f)$, implying $y_{x_l} \neq y_{f(x_l)}$. Note that there has to exist $x \in f^{-1}(\{f(x_l)\})$ such that $y_x = y_{f(x_l)} = y_{f(x)}$, as otherwise $y_{f(x)}$ would not be the winning y for $f(x)$. We place x into $I_{\mathcal{X}_l}$, noting that $x \notin \mathcal{X}_l$, and repeat this for every element in $\mathcal{X}_l \cap I_{\mathcal{X}}(f)$. Finally, we add to $I_{\mathcal{X}_l}$ all the elements that are in $I_{\mathcal{X}}(f) \setminus \mathcal{X}_l$. By the construction we see that $I_{\mathcal{X}_l}$ is a set on which f is injective, since we always choose only one representative from each $f^{-1}(\widetilde{x})$, and

$$p(I_{\mathcal{X}_l}) = p(\mathcal{X}_l \sqcup (I_{\mathcal{X}}(f) \setminus \mathcal{X}_l)) = p(I_{\mathcal{X}}(f)) \geq 1-\delta,$$

where $\sqcup$ denotes a disjoint union. Since $\mathcal{X}_l$ and $I_{\mathcal{X}_l}$ are disjoint, this yields $p(\mathcal{X}_l) < \delta$, which together with (A.1.1) finishes the proof.

(2) $n > 0$:

Let $I_{\mathcal{X}}(f_0,\dots,f_n) := \bigcup_{i=0}^{n} I_{\mathcal{X}}(f_i)$. It suffices to prove that $(f_0,\dots,f_n)$ is injective on $I_{\mathcal{X}}(f_0,\dots,f_n)$, since we already have $p(I_{\mathcal{X}}(f_0,\dots,f_n)) \geq (1-\delta)$.

Define $I'_{\mathcal{X}}(f_0),\dots,I'_{\mathcal{X}}(f_n)$ inductively by $I'_{\mathcal{X}}(f_0) := I_{\mathcal{X}}(f_0)$, and

$$I'_{\mathcal{X}}(f_k) := I_{\mathcal{X}}(f_k) \setminus \left(\bigcup_{j=0}^{k-1} I_{\mathcal{X}}(f_j)\right), \quad k = 1,\dots,n.$$

Then

$$I_{\mathcal{X}}(f_0, \ldots, f_n) = \bigcup_{i=0}^{n} I_{\mathcal{X}}(f_i) = \bigsqcup_{i=0}^{n} I'_{\mathcal{X}}(f_i).$$

Therefore, it suffices to prove that $(f_0, \ldots, f_n)$ is injective on $\bigsqcup_{=0}^{n} I'_{\mathcal{X}}(f_i)$.

Let $x, \widetilde{x} \in \bigsqcup_{i=0}^{n} I'_{\mathcal{X}}(f_i)$, $x \neq \widetilde{x}$, and let k, l be such that $x \in I'_{\mathcal{X}}(f_k), \widetilde{x} \in I'_{\mathcal{X}}(f_l)$. Then $f_{\max\{k,l\}}(x) \neq f_{\max\{k,l\}}(\widetilde{x})$, for which we use that $I'_{\mathcal{X}}(f_{\max\{k,l\}})$ is injective, if $\widetilde{x} \in I'_{\mathcal{X}}(f_{\max\{k,l\}})$, or that $I'_{\mathcal{X}}(f_{\max\{k,l\}})$ is disjoint from all the previous ones. This proves that $(f_0, \ldots, f_n)(x) \neq (f_0, \ldots, f_n)(\widetilde{x})$, so $(f_0, \ldots, f_n)$ is injective on $\bigsqcup_{i=0}^{n} I'_{\mathcal{X}}(f_i)$.

To finish the proof, we note that if f_0 is δ-injective, then

$$p\left(\bigcup_{i=0}^{n} I_{\mathcal{X}}(f_i)\right) \geq p(I_{\mathcal{X}}(f_0)) \geq (1 - \delta),$$

implying that $(f_0, \ldots, f_n)$ is δ-safe, by the results in the previous paragraph. □

Safe Transformations via Information Theory

In this section we prove Lemmas 3.5.3 and 3.5.4. The *mutual information* between random variables X and Y, taking values in finite sets $\mathcal{X}$ and $\mathcal{Y}$, is defined as

$$I(X;Y) := D_{KL}(p(x,y) \,\|\, p(x)p(y)) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x,y) \log \frac{p(x,y)}{p(x)p(y)},$$

where the logarithm is in base 2. Lemma 3.5.3 can be understood as the bound on the allowed loss in the mutual information, by noting that

$$\begin{aligned} I(X;Y) - I(f(X);Y) &= D_{KL}(p(x,y) \,\|\, p(x)p(y)) - D_{KL}(p_{f^{-1}}(f(x),y) \,\|\, p_{f^{-1}}(f(x))p_{f^{-1}}(y)) \\ &= \mathbb{E}_{p(x,y)} \log \frac{p(x,y)}{p(x)p(y)} - \mathbb{E}_{p(x,y)} \log \frac{p_{f^{-1}}(f(x),y)}{p_{f^{-1}}(f(x))p_{f^{-1}}(y)} \\ &= \mathbb{E}_{p(x,y)} \log \frac{p(y|x)}{p_{f^{-1}}(y|f(x))} = D_{KL}\left(p(y|x) \,\|\, p_{f^{-1}}(y|f(x))\right), \end{aligned} \tag{A.1.2}$$

since $p_{f^{-1}}(y) = p(y)$. The proof of Lemma 3.5.3 starts by connecting the change in the Bayes error with the L^1-*norm* of the distance between probability distributions, which, in a finite space, equals twice the *total variation distance*. We conclude the proof by applying *Pinsker's inequality*. For a detailed analysis of all of these terms we refer an interested reader to Chapter 11 in Cover and Thomas (2006).

Proof of Lemma 3.5.3: Note that Equation 3.5.5 and the definitions of y_x and $y_{f(x)}$ yield

$$\begin{aligned}
\Delta^*_{f,X} &= \mathbb{E}_{x\sim X}\left[p(y_x \mid x) - p_{f^{-1}}(y_{f(x)}|f(x))\right] \\
&= \mathbb{E}_{x\sim X}\left[p(y_x \mid x) - p_{f^{-1}}(y_x|f(x))\right] + \mathbb{E}_{x\sim X}\underbrace{\left[p_{f^{-1}}(y_x|f(x)) - p_{f^{-1}}(y_{f(x)}|f(x))\right]}_{\leq 0} \\
&\leq \frac{1}{2}\left|\mathbb{E}_{x\sim X}\left[p(y_x \mid x) - p_{f^{-1}}(y_x|f(x))\right]\right| + \frac{1}{2}\left|\mathbb{E}_{x\sim X}\sum_{y\neq y_x}\left[p(y \mid x) - p_{f^{-1}}(y|f(x))\right]\right| \\
&\leq \frac{1}{2}\mathbb{E}_{x\sim X}\sum_{y\in\mathcal{Y}}\left|p(y \mid x) - p_{f^{-1}}(y \mid f(x))\right| \\
&= \frac{1}{2}\left\|p(x,y) - \frac{p(x)}{p_{f^{-1}}(f(x))}p_{f^{-1}}(f(x),y)\right\|_1,
\end{aligned}$$

where we introduced the sum by expanding

$$1 = p(y_x|x) + \sum_{y\neq y_x} p(y|x) = p_{f^{-1}}(y_x|f(x)) + \sum_{y\neq y_x} p_{f^{-1}}(y|f(x)),$$

whilst using the triangle inequality. Pinsker's inequality implies that

$$\left\|p(x,y) - \frac{p(x)}{p_{f^{-1}}(f(x))}p_{f^{-1}}(f(x),y)\right\|_1 \leq \sqrt{(2\ln 2)D_{KL}\left(p(y \mid x) \,\|\, p_{f^{-1}}(y \mid f(x))\right)} \leq 2\delta,$$

finishing the proof after dividing by 2. □

Finally, we provide a construction which shows that the bound in Lemma 3.5.3 is of the right order.

Proof of Lemma 3.5.4: Recall that we define $\eta(x) = p(1 \mid x)$. We start with $|\mathcal{X}| = 2$, since the general case will be a straightforward extension of it.

Let $\mathcal{X} = \{x_0, x_1\}$ and $\widetilde{\mathcal{X}} = \{\widetilde{x}\}$. We define p on $\mathcal{X}\times\mathcal{Y}$ by $p(x_0) = p(x_1) = 1/2$, and $\eta(x_0) = \frac{1}{2} - \delta$, $\eta(x_1) = \frac{1}{2} + \delta$, which defines $p(x,y)$. For the change in the Bayes error we have

$$\Delta^*_{f,X} = \frac{1}{2} - \sum_{x\in\{x_0,x_1\}} p(x)\min\{\eta(x), 1-\eta(x)\} = \delta.$$

For the KL-divergence, note that

$$\begin{aligned} D_{KL}(p(y \mid x) \,\|\, p_{f^{-1}}(y \mid f(x))) &= \sum_{x \in \{x_0, x_1\}} \sum_{y \in \{0,1\}} p(x,y) \log \frac{p(y \mid x)}{p_{f^{-1}}(y \mid f(x))} \\ &= \sum_{x \in \{x_0, x_1\}} p(x) \sum_{y \in \{0,1\}} p(y \mid x) \log 2p(y \mid x) \\ &= \sum_{x \in \{x_0, x_1\}} p(x) \left(\eta(x_0) \log 2\eta(x_0) + \eta(x_1) \log 2\eta(x_1)\right) \\ &= \frac{1}{2} \left((1-2\delta)\log(1-2\delta) + (1+2\delta)\log(1+2\delta)\right), \end{aligned}$$

where we used $\eta(x_0) = \frac{1}{2} - \delta = 1 - \eta(x_1)$. Taylor expansion for $|x| < 1$ gives

$$(1+x)\ln(1+x) + (1-x)\ln(1-x) = 2 \sum_{k \in \mathbb{N}} \frac{1}{(2k-1)2k} x^{2k},$$

which implies

$$\begin{aligned} D_{KL}(p(y \mid x) \,\|\, p_{f^{-1}}(y \mid f(x))) &= \frac{1}{\ln 2} \sum_{k \in \mathbb{N}} \frac{2^{2k}}{(2k-1)2k} \delta^{2k} \\ &= \frac{2}{\ln 2}\delta^2 + \frac{4}{3\ln 2}\delta^4 + \frac{32}{15\ln 2}\delta^6 + \ldots = (2/\ln 2)\delta^2 + O(\delta^4), \end{aligned}$$

finishing the proof for $|\mathcal{X}| = 2$.

Suppose that $|\mathcal{X}| > 2$. Since $|\widetilde{\mathcal{X}}| < |\mathcal{X}|$, we know that there exists a $\widetilde{x}$ such that $|f^{-1}(\widetilde{x})| \geq 2$, so let $x_0, x_1 \in f^{-1}(\widetilde{x})$ be distinct. We define p on $\mathcal{X} \times \mathcal{Y}$ as $p(x,y) = 0$ for $x \notin \{x_0, x_1\}$, while for $p(x_0, y), p(x_1, y)$ we do the above construction, which proves the lemma. □

For $|\mathcal{X}| > 2$ we used the most simple construction, however, one can extend the idea behind the proof for $|\mathcal{X}| = 2$ into a more general one. For example, we can define a probability distribution in which for all $x \in \mathcal{X}$ one has $\eta(x) \in \{\frac{1}{2} - \delta, \frac{1}{2} + \delta\}$, with the same proportion of each. In other words, each x is a bucket with either $\frac{1}{2} - \delta$ values being 1, or $\frac{1}{2} + \delta$ values being 1. Let

$$\mathcal{X}_0 := \left\{x \in \mathcal{X} \colon \eta(x) = \frac{1}{2} - \delta\right\}, \quad \mathcal{X}_1 := \left\{x \in \mathcal{X} \colon \eta(x) = \frac{1}{2} + \delta\right\},$$

thus $\mathcal{X} = \mathcal{X}_0 \sqcup \mathcal{X}_1$. Now f can either merge buckets of the same type, in which case neither do the Bayes error nor the KL-divergence change, or buckets of a different type, where the changes are δ and $2\delta^2 + O(\delta^4)$, respectively. Choosing the right proportion of each bucket in $f^{-1}(\widetilde{x})$ is now an easy task, yielding the construction.

Safe Transformations on Similar Probability Distributions

In this section we prove Theorem 3.5.5. As mentioned in the main body, transformations used for estimating the Bayes error might have been trained on a distribution different then the target one, and as such, might change the Bayes error in an unfavourable way when applied to the distribution of interest. We investigate this in the next few paragraphs.

Let $p_S(x, y)$ be the *source* probability distribution based on random variables $X_S \in \mathcal{X}_S$ and $Y_S \in \mathcal{Y}_S$, which is the probability distributions used for training a transformation f_S. With $p_T(x, y)$ we denote the *target* probability distribution, the one that serves as the basis for random variables $X_T \in \mathcal{X}_T$ and $Y_T \in \mathcal{Y}_T$. Theorem 3.5.5 provides a sufficient condition on the relationship between p_S and p_T, in terms of the Kullback-Leibler divergence, so that a δ-safe transformation with respect to p_S is a δ'-safe transformation with respect to p_T.

Before we start with the proof, let us argue why it makes sense to set $\mathcal{X}_S = \mathcal{X}_T = \mathcal{X}$ and $\mathcal{Y}_S = \mathcal{Y}_T = \mathcal{Y}$, as it is assumed in Theorem 3.5.5, even when we have more then two classes. When it comes to $\mathcal{X}$, any pre-trained feature transformation comes with a fixed input dimension. Therefore, in order to apply a feature transformation one usually needs to modify the input vector. When dealing with images, this often means resizing the image, whether it is by scaling the image, or by adding white/black pixels. This is an injective process as long as we do not reduce the dimension, which is reasonable to assume as we usually use transformations trained on larger inputs. Therefore, instead of $\mathcal{X}_S$ we can consider a probability distribution mapped through an injective map $g\colon \mathcal{X}_S \to \mathcal{X}$, which is a safe transformation. We will omit the mention of g for the ease of notation. For $\mathcal{Y}$, we first assume that $\mathcal{Y}_T \subseteq \mathcal{Y}_S$, since we want to use feature transformations that work well on more difficult tasks. When f_S is safe with respect to p_S on $\mathcal{X}_S \times \mathcal{Y}_S$, it is easy to see that f_S is also safe with respect to the restriction of p_S to $\mathcal{X}_S \times \mathcal{Y}_T$. This does not necessarily hold when we weaken the condition to δ-safe. In that case, our assumption is that f is δ-safe with respect to p_S on $\mathcal{X}_S \times \mathcal{Y}_T$ in the first place, thus taking $\mathcal{Y}_T$ as the source $\mathcal{Y}$.

Proof of Theorem 3.5.5: Note that

$$R^*_{f(X_T)} - R^*_{X_T} \leq \underbrace{\left|R^*_{f(X_T)} - R^*_{f(X_S)}\right|}_{I_1} + \underbrace{\left|R^*_{f(X_S)} - R^*_{X_S}\right|}_{I_2} + \underbrace{\left|R^*_{X_S} - R^*_{X_T}\right|}_{I_3}.$$

Since f is δ-safe with respect to p_S, we have $I_2 \leq \delta$.

For I_1, let $\tilde{p}_S := p^{(S)}_{f^{-1}}$ and $\tilde{p}_T := p^{(T)}_{f^{-1}}$ denote the corresponding measures with respect to $\widetilde{\mathcal{X}}$, and

let

$$y_{\tilde{x}}^{(S)} = \arg\max_{y\in\mathcal{Y}} \tilde{p}_S(\tilde{x}, y), \quad y_{\tilde{x}}^{(T)} = \arg\max_{y\in\mathcal{Y}} \tilde{p}_T(\tilde{x}, y).$$

For a fixed $\tilde{x}$ we can assume without loss of generality that $\tilde{p}_S(\tilde{x}, y_{\tilde{x}}^{(S)}) \geq \tilde{p}_T(\tilde{x}, y_{\tilde{x}}^{(T)})$. Then

$$\begin{aligned}\left|\max_{y\in\mathcal{Y}} \tilde{p}_S(\tilde{x}, y) - \max_{y\in\mathcal{Y}} \tilde{p}_T(\tilde{x}, y)\right| &= \tilde{p}_S(\tilde{x}, y_{\tilde{x}}^{(S)}) - \tilde{p}_T(\tilde{x}, y_{\tilde{x}}^{(T)}) \\ &\leq \tilde{p}_S(\tilde{x}, y_{\tilde{x}}^{(S)}) - \tilde{p}_T(\tilde{x}, y_{\tilde{x}}^{(S)}) \\ &\leq \sum_{y\in\mathcal{Y}} \left|\tilde{p}_S(\tilde{x}, y) - \tilde{p}_T(\tilde{x}, y)\right|.\end{aligned}$$

Summing the above over all $\tilde{x} \in \widetilde{\mathcal{X}}$ yields

$$\begin{aligned}I_1 &= \left|\sum_{\widetilde{x}\in\widetilde{\mathcal{X}}} \left[\max_{y\in\mathcal{Y}} \tilde{p}_S(\widetilde{x}, y) - \max_{y\in\mathcal{Y}} \tilde{p}_T(\widetilde{x}, y)\right]\right| \\ &\leq \sum_{\widetilde{x}\in\widetilde{\mathcal{X}}} \sum_{y\in\mathcal{Y}} \left|\tilde{p}_S(\widetilde{x}, y) - \tilde{p}_T(\widetilde{x}, y)\right| \\ &\overset{\triangle}{\leq} \sum_{\widetilde{x}\in\widetilde{\mathcal{X}}} \sum_{y\in\mathcal{Y}} \sum_{x\in f^{-1}(\widetilde{x})} \left|p_S(x, y) - p_T(x, y)\right| \\ &= \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} \left|p_S(x, y) - p_T(x, y)\right| = \|p_S - p_T\|_1.\end{aligned}$$

Repeating the same calculation for I_3 implies $I_3 \leq \|p_S - p_T\|_1$. Combining the bounds for I_1, I_2 and I_3 yields

$$R^*_{f(X_T)} - R^*_{X_T} \leq \delta + 2\|p_S - p_T\|_1.$$

As in the previous section, Pinsker's inequality implies

$$R^*_{f(X_T)} - R^*_{X_T} \leq \delta + 2\sqrt{(2\ln 2) D_{KL}(p_S \parallel p_T)} \leq \delta + \varepsilon,$$

concluding the proof. □

Safety and the g-squared loss

In this section we provide a characterization of δ-safe functions in terms of the g-squared loss of f, by proving Theorem 3.5.6. Since this will serve as a connecting point between the rates of convergence of a kNN classifier and the safety of a transformation, from this point onwards we restrict ourselves to binary classification, assuming that $\mathcal{Y} = \{0, 1\}$.

We start by proving an auxiliary lemma that is used both in the proof of Theorem 3.5.6 and in the proof of Theorem 3.5.1, presented in the main body, which is the main result of Section 3.5. It states that the g-squared loss of f on X can only be reduced by performing a change of variables to the identity function acting on $f(X)$.

Lemma A.1.3. *For any function f, one has $\mathcal{L}_{g,f(X)}(id) \leq \mathcal{L}_{g,X}(f)$.*

Proof: Let $\widetilde{X} = f(X)$. Note that for a fixed $\widetilde{x} \in \widetilde{\mathcal{X}}$,

$$\eta_{f^{-1}}(\widetilde{x}) = p_{f^{-1}}(1|\widetilde{x}) = p\left(1|X \in f^{-1}(\widetilde{x})\right) = \frac{\mathbb{E}_X \eta(X)\mathbf{1}_{\{X\in f^{-1}(\widetilde{x})\}}}{\mathbb{E}_X \mathbf{1}_{\{X\in f^{-1}(\widetilde{x})\}}}.$$

Hence,

$$\begin{aligned}
\mathcal{L}_{g,f(X)}(id) &= \mathbb{E}_{\widetilde{x}}\left((g\circ id)(\widetilde{X}) - \eta_{f^{-1}}(\widetilde{X})\right)^2 \\
&= \mathbb{E}_{\widetilde{X}}\left(g(\widetilde{X}) - \frac{\mathbb{E}_X \eta(X)\mathbf{1}_{\{X\in f^{-1}(\widetilde{X})\}}}{\mathbb{E}_X \mathbf{1}_{\{X\in f^{-1}(\widetilde{X})\}}}\right)^2 \\
&= \mathbb{E}_{\widetilde{X}}\left(\frac{\mathbb{E}_X((g\circ f)(X) - \eta(X))\mathbf{1}_{\{X\in f^{-1}(\widetilde{X})\}}}{\mathbb{E}_X \mathbf{1}_{\{X\in f^{-1}(\widetilde{X})\}}}\right)^2,
\end{aligned}$$

since for all $x, x' \in f^{-1}(\widetilde{x})$ one has $(g\circ f)(x) = (g\circ f)(x') = g(\widetilde{x})$. The Cauchy-Schwarz inequality yields

$$\begin{aligned}
\mathcal{L}_{g,f(X)}(id) &\leq \mathbb{E}_{\widetilde{X}} \frac{\mathbb{E}_X((g\circ f)(X) - \eta(X))^2 \mathbf{1}_{\{X\in f^{-1}(\widetilde{X})\}}}{\mathbb{E}_X \mathbf{1}_{\{X\in f^{-1}(\widetilde{X})\}}} \\
&= \mathbb{E}_X((g\circ f)(X) - \eta(X))^2 = \mathcal{L}_{g,X}(f),
\end{aligned}$$

proving the claim. □

We conclude this section by proving Theorem 3.5.6, the final ingredient for connecting the convergence rates of a kNN classifier with the Bayes error in the original space.

Proof of Theorem 3.5.6: As in the proof of Lemma 3.5.3, we know that

$$\Delta^*_{f,X} \leq \mathbb{E}_X\left(p(y_x \mid x) - p_{f^{-1}}(y_x \mid f(x))\right) \leq \mathbb{E}_X\left|\eta(X) - \eta_{f^{-1}}(f(X))\right|.$$

The triangle and the Cauchy-Schwarz inequality, once for each term, yield

$$\begin{aligned}
\Delta^*_{f,X} &\leq \mathbb{E}_X\left|\eta(X) - (g\circ f)(X)\right| + \mathbb{E}_X\left|(g\circ f)(X) - \eta_{f^{-1}}(f(X))\right| \\
&= \mathbb{E}_X\left|\eta(X) - (g\circ f)(X)\right| + \mathbb{E}_{\widetilde{X}}\left|g(\widetilde{X}) - \eta_{f^{-1}}(\widetilde{X})\right| \\
&\leq \Big(\underbrace{\mathbb{E}_X\left|\eta(X) - (g\circ f)(X)\right|^2}_{\mathcal{L}_{g,X}(f)}\Big)^{1/2} + \Big(\underbrace{\mathbb{E}_{\widetilde{X}}\left|g(\widetilde{X}) - \eta_{f^{-1}}(\widetilde{X})\right|^2}_{\mathcal{L}_{g,f(X)}(id)}\Big)^{1/2}.
\end{aligned}$$

The claim now follows by Lemma A.1.3. □

A.1.2 Convergence Rates of a kNN Classifier over Transformed Features

We now present the proof of Theorem 3.5.8, mimicking the proof of Theorem 6.2 from Györfi et al. (2002). We insert our (weaker) probabilistic Lipschitz assumption where appropriate. It allows us to remove any additional constraint on f, leaving us with a statement dependent only on $\mathcal{L}_{g,X}(f)$. As discussed in Section 3.5.3, for $g(x) = \text{softmax}(W^T x + b)$ this can be used to rank various transformations f by simply reporting the mean squared error of the test set, denoted by $MSE_g(f, W, b)$. The price we need to pay is an additional additive error term. However, since an unavoidable error term as a function of $\mathcal{L}_{g,X}(f)$ already exists in Theorem 3.5.6, we accept it here, having in mind the flexibility it gives us. Optimizing this additive error term could form an interesting path for further research.

Proof of Theorem 3.5.8: It is well known (see Chapter 1 in Györfi et al. (2002)) that

$$\mathbb{E}_n[(R_X)_{n,k}] - R_X^* \leq 2\mathbb{E}_n\mathbb{E}_X\left|\eta_{n,k}(X) - \eta(X)\right| \leq 2\sqrt{\mathbb{E}_n\mathbb{E}_X\left|\eta_{n,k}(X) - \eta(X)\right|^2}, \tag{A.1.3}$$

where the last inequality is a simple application of the Cauchy-Schwarz inequality. With the assumptions as above, it suffices to prove that for all $w \in \mathbb{R}^d$,

$$\mathbb{E}_n\mathbb{E}_X\left|\eta_{n,k}(X) - \eta(X)\right|^2 \leq \frac{1}{k} + cL\left(\frac{k}{n}\right)^{2/d} + \delta + 2\varepsilon^2, \tag{A.1.4}$$

for some $c > 0$. Let $(X_1, Y_1), \ldots, (X_n, Y_n)$ be the set of n-samples distributed using $p(x, y)$. For $x \in \mathcal{X}$, let $n(i, x)$ denote the index of the i-th nearest neighbor of x in $X_1, \ldots, X_n$. Then

$$\mathbb{E}_n\left|\eta_{n,k}(x) - \eta(x)\right|^2 = \underbrace{\mathbb{E}_n\left|\eta_{n,k}(x) - \frac{1}{k}\sum_{i\in[k]}\eta(X_{n(i,x)})\right|^2}_{J_1(x)} + \underbrace{\mathbb{E}_n\left|\frac{1}{k}\sum_{i\in[k]}\eta(X_{n(i,x)}) - \eta(x)\right|^2}_{J_2(x)}.$$

For $J_1(x)$ note that

$$J_1(x) = \mathbb{E}_n\left|\frac{1}{k}\sum_{i\in[k]}\left(\eta_{n,k}(x) - \eta(X_{n(i,x)})\right)\right|^2 = \frac{1}{k^2}\sum_{i\in[k]}\mathbb{E}_n\left|Y_{n(i,x)} - \eta(X_{n(i,x)})\right|^2 \leq \frac{1}{k}. \tag{A.1.5}$$

For $J_2(x)$ we have

$$\mathbb{E}_X J_2(X) = \mathbb{E}_X\mathbb{E}_n\left|\frac{1}{k}\sum_{i\in[k]}\left(\eta(X_{n(i,X)}) - \eta(X)\right)\right|^2 \leq \frac{1}{k}\sum_{i\in[k]}\mathbb{E}_n\mathbb{E}_X\left|\eta(X_{n(i,X)}) - \eta(X)\right|^2,$$

by the Cauchy-Schwarz inequality. Let $\texttt{GOOD}_{\varepsilon,L} := \{(X, X') \colon |\eta(X) - \eta(X')| \leq \varepsilon + L\|X - X'\|\}$. Since η is (ε, δ, L)-probably Lipschitz and $(a+b)^2 \leq 2a^2 + 2b^2$, we have that

$$\begin{aligned}\mathbb{E}_X J_2(X) &\leq \frac{1}{k}\sum_{i\in[k]} \mathbb{E}_n \left(1 - \mathbb{P}\left((X, X_{n(i,X)}) \in \texttt{GOOD}_{\varepsilon,L}\right)\right) + \frac{1}{k}\sum_{i\in[k]} \mathbb{E}_n \mathbb{E}_X \left(2\varepsilon^2 + 2L^2\|X_{n(i,X)} - X\|^2\right) \\ &\leq \delta + 2\varepsilon^2 + 2L^2 \mathbb{E}_X \underbrace{\mathbb{E}_n \frac{1}{k}\sum_{i\in[k]} \left\|X_{n(i,X)} - X\right\|^2}_{J_3(X)}.\end{aligned}$$

The term $J_3(X)$ is exactly the same as the upper bound for $I_2(X)$ in the proof of Theorem 6.2 in Györfi et al. (2002), where it is shown that there exists a $c > 0$ such that $\mathbb{E}_X J_3(X) \leq c(k/n)^{2/d}$.

Combining the bounds for J_1, J_2 and J_3 proves the claim. □

The final result of this section establishes the probabilistic Lipschitz condition in terms of the g-squared error of f. It is the glue that brings all the pieces together, having in mind that it is applied on $f(X)$.

Proof of Lemma 3.5.9: Note that the triangle inequality implies

$$|\eta(X) - \eta(X')| \leq \underbrace{|\eta(X) - g(X)|}_{I_1(X)} + \underbrace{|g(X) - g(X')|}_{I_2} + \underbrace{|g(X') - \eta(X')|}_{I_1(X')}.$$

For I_2 note that the fact that g is L-Lipschitz implies $I_2(X, X') \leq L\|X - X'\|$.

For $I_1(X), I_1(X')$ we start by defining $\texttt{GOOD}_t := \{x \in \mathcal{X} \colon |\eta(x) - g(x)| \leq t\}$. Note that Markov's inequality yields

$$\mathbb{P}(X \notin \texttt{GOOD}_t) = \mathbb{P}\left(|\eta(X) - g(X)|^2 \geq t^2\right) \leq \frac{\mathcal{L}_{g,X}(id)}{t^2}.$$

Therefore,

$$\begin{aligned}\mathbb{P}(|\eta(X) - \eta(X')| \leq \varepsilon + L\|X - X'\|) &\geq \mathbb{P}(X, X' \in \texttt{GOOD}_{\varepsilon/2}) \\ &\geq \left(1 - \frac{4\mathcal{L}_{g,X}(id)}{\varepsilon^2}\right)^2 \geq 1 - \frac{8\mathcal{L}_{g,X}(id)}{\varepsilon^2},\end{aligned}$$

concluding the proof. □

A.2 Class-Dependent Label Noise in Ease.ML/Snoopy

In this Section we provide the proof of Theorem 3.6.1 and provide further discussion on the main result.

Recall that we suppose that the noise in the labels is class dependent, rather than instance dependent. In other words, for a noisy random variable $\widetilde{Y}$, we assume

$$t_{\widetilde{y},y} := \mathbb{P}(Y_\rho = \widetilde{y}\,|Y = y, X = x) = \mathbb{P}(Y_\rho = \widetilde{y}\,|Y = y)$$

corresponding to the transition matrix T in Wei et al. (2022). Moreover, we assume $y_x = \arg\max_{y\in\mathcal{Y}} p(Y = y|x) = \arg\max_{y\in\mathcal{Y}} p_\rho(Y_\rho = y|x)$.

Proof of Theorem 3.6.1: We have

$$\begin{aligned}
R^*_{X,Y_\rho} &= \mathbb{E}_X[1 - \mathbb{P}(Y_\rho = y_x|x)] \\
&= 1 - \mathbb{E}_X\left[\sum_{y\in\mathcal{Y}} \mathbb{P}(Y_\rho = y_x, Y = y|X = x)\right] \\
&= 1 - \mathbb{E}_X\left[\sum_{y\in\mathcal{Y}} \mathbb{P}(Y_\rho = y_x|Y = y, X = x)\mathbb{P}(Y = y|X = x)\right] \\
&= 1 - \mathbb{E}_X\left[\sum_{y\in\mathcal{Y}} t_{y_x,y}p(y|x)\right] \\
&= 1 - \mathbb{E}_X[t_{y_x,y_x}p(y_x|x)] - \mathbb{E}_X\left[\sum_{y\neq y_x} t_{y_x,y}p(y|x)\right] \\
&= \mathbb{E}_X[1 - p(y_x|x)] + \mathbb{E}_X[\rho(y_x)p(y_x|x)] - \mathbb{E}_X\left[\sum_{y\neq y_x} t_{y_x,y}p(y|x)\right],
\end{aligned}$$

recalling that $\rho(y) = 1 - t_{y,y}$ is the flipping fraction with respect to y.

This implies that

$$R^*_{X,Y_\rho} = R^*_{X,Y} + \mathbb{E}_X[\rho(y_x)p(y_x|x)] - \mathbb{E}_X\left[\sum_{y\neq y_x} t_{y_x,y}p(y|x)\right], \tag{A.2.1}$$

concluding the proof. □

Getting lower and upper bounds on the BER with respect to Y_ρ To get lower and upper bounds from (3.6.3), note that

$$\min_y \rho(y) \le \rho(y_x) \le \max_y \rho(y) \quad \Longleftrightarrow \quad \min_y (1 - t_{y,y}) \le \rho(y_x) \le \max_y (1 - t_{y,y}), \tag{A.2.2}$$

and, for $y \neq y_x$,

$$- \max_{y',y'':y'\neq y''} t_{y',y''} \le -t_{y_x,y} \le - \min_{y',y'':y'\neq y''} t_{y',y''}. \tag{A.2.3}$$

Comparing with Figure 3 in Wei et al. (2022), (A.2.2) corresponds to the smallest and largest distance from 1 on the diagonal, and (A.2.3) corresponds to the smallest and largest non-diagonal elements.

Plugging in the two lower bounds of (A.2.2) and (A.2.3) into (3.6.3), we get

$$\begin{aligned} R^*_{X,Y_\rho} &\ge R^*_{X,Y} + \min_y (1 - t_{y,y}) \mathbb{E}_X[p(y_x|x)] - \max_{y',y'' :\, y'\neq y''} t_{y',y''} \mathbb{E}_X \left[\sum_{y\neq y_x} p(y|x) \right] \\ &= R^*_{X,Y} + \min_y (1 - t_{y,y}) \mathbb{E}_X[p(y_x|x)] - \max_{y',y'' :\, y'\neq y''} t_{y',y''} \mathbb{E}_X \left[1 - p(y_x|x) \right] \qquad \text{(A.2.4)} \\ &= R^*_{X,Y} + (1 - R^*_{X,Y}) \min_y (1 - t_{y,y}) - R^*_{X,Y} \max_{y',y'' :\, y'\neq y''} t_{y',y''}. \qquad \text{(A.2.5)} \end{aligned}$$

Plugging in the two upper bounds yields

$$R^*_{X,Y_\rho} \le R^*_{X,Y} + (1 - R^*_{X,Y}) \max_{y\in\mathcal{Y}} (1 - t_{y,y}) - R^*_{X,Y} \min_{y',y'' :\, y'\neq y''} t_{y',y''}. \tag{A.2.6}$$

Merging (A.2.4) and (A.2.6) yields

$$\begin{aligned} R^*_{X,Y_\rho} \in \Bigg[& R^*_{X,Y} + (1 - R^*_{X,Y}) \min_{y\in\mathcal{Y}} (1 - t_{y,y}) - R^*_{X,Y} \max_{y',y'' :\, y'\neq y''} t_{y',y''}, \qquad \text{(A.2.7)} \\ & R^*_{X,Y} + (1 - R^*_{X,Y}) \max_{y\in\mathcal{Y}} (1 - t_{y,y}) - R^*_{X,Y} \min_{y',y'' :\, y'\neq y''} t_{y',y''} \Bigg]. \qquad \text{(A.2.8)} \end{aligned}$$

Using $0 \le R^*_{X,Y} \le s_{X,Y}$, where $s_{X,Y}$ denotes the error of state-of-the-art model, we get

$$R^*_{X,Y_\rho} \in \left[(1 - s_{X,Y}) \min_{y\in\mathcal{Y}} (1 - t_{y,y}) - s_{X,Y} \max_{y',y'' :\, y'\neq y''} t_{y',y''}, \quad s_{X,Y} + \max_{y\in\mathcal{Y}} (1 - t_{y,y}) \right]. \tag{A.2.9}$$

These represent valid lower and upper bounds,whereas in experimental section we also plot the following approximation of the RHS of (3.6.3):

$$R_{X,Y_{\bar\rho}} = s_{X,Y} + \mathbb{E}_Y \rho(y)(1 - s_{X,Y}), \tag{A.2.10}$$

representing average distance from 1 of diagonal elements, instead of simply taking minimum and maximum of the distances.

Examples We now provide two examples of how one could use Theorem 3.6.1, noting that the first example yields Lemma 3.4.1.

- *(Uniform flipping)* In this scenario we assume that for each class we flip ρ fraction of examples to any other class (including the original one) uniformly at random. In other words,

$$\rho(y) = \rho \cdot \left(1 - \frac{1}{C}\right), \forall y, \quad t_{y',y''} = \frac{\rho}{C}, \forall y' \neq y''.$$

  In this case Equation 3.6.3 becomes

$$\begin{aligned} R^*_{X,Y_\rho} &= R^*_{X,Y} + \rho\left(1 - \frac{1}{C}\right)\mathbb{E}_X[p(y_x|x)] - \frac{\rho}{C}\mathbb{E}_X\left[\sum_{y \neq y_x} p(y|x)\right] \\ &= R^*_{X,Y} + \rho\left(1 - \frac{1}{C}\right)\mathbb{E}_X[p(y_x|x)] - \frac{\rho}{C}\left(1 - \mathbb{E}_X\left[p(y_x|x)\right]\right) \\ &= R^*_{X,Y} + \rho\mathbb{E}_X[p(y_x|x)] - \frac{\rho}{C} \\ &= R^*_{X,Y} + \rho\left(1 - \frac{1}{C} - R^*_{X,Y}\right). \end{aligned}$$

  We see that this is exactly Lemma 3.4.1.

- *(Pairwise flipping)* In this scenario we assume that for each y there exists a single class y'_x such that $p(y'_x|x) = 1 - p(y_x|x)$. In that case, we flip ρ fraction of samples to y'_x. That means that $\rho(y_x) = 1 - t_{y_x,y'_x} = \rho$, and all other $t_{y_x,y} = 0$, for $y \neq y'_x, y_x$. Then Equation 3.6.3 becomes

$$\begin{aligned} R^*_{X,Y_\rho} &= R^*_{X,Y} + \rho\mathbb{E}_X[p(y_x|x)] - \rho\mathbb{E}_X\left[p(y'_x|x)\right] \\ &= R^*_{X,Y} + \rho\left(2\mathbb{E}_X[p(y_x|x)] - 1\right) \\ &= R^*_{X,Y} + \rho\left(1 - 2R^*_{X,Y}\right). \end{aligned}$$

Bounds for CIFAR-N Inserting the values from Figure 3 in Wei et al. (2022) into Equation A.2.9 yields:

$$\begin{aligned} \text{CIFAR10-N Aggregated} &: \quad R_{X,Y_\rho} \in [0.03 - 0.10 \cdot s_{X,Y}, \quad 0.17 + s_{X,Y}] \\ \text{CIFAR10-N Random} &: \quad R_{X,Y_\rho} \in [0.10 - 0.23 \cdot s_{X,Y}, \quad 0.26 + s_{X,Y}] \\ \text{CIFAR100-N Noisy} &: \quad R_{X,Y_\rho} \in [0.083 - 0.312 \cdot s_{X,Y}, \quad 0.854 + s_{X,Y}]. \end{aligned}$$

A.3 Justifications on Taking the Minimum

Figures A.1- A.4 illustrate the quantities defined in Section 3.6.2.2.

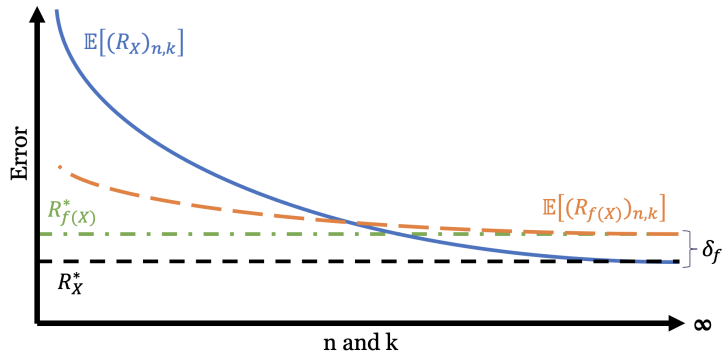


Figure A.1: Transformation bias δ_f for a fixed transformation f.

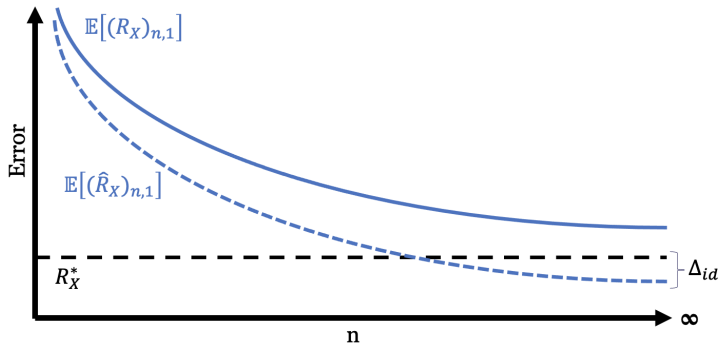


Figure A.2: Asymptotic tightness Δ_{id} on the raw data (identify transformation).

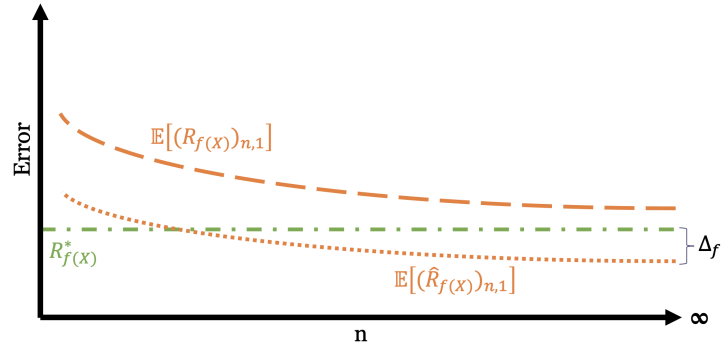


Figure A.3: Asymptotic tightness Δ_f for a fixed transformation f.

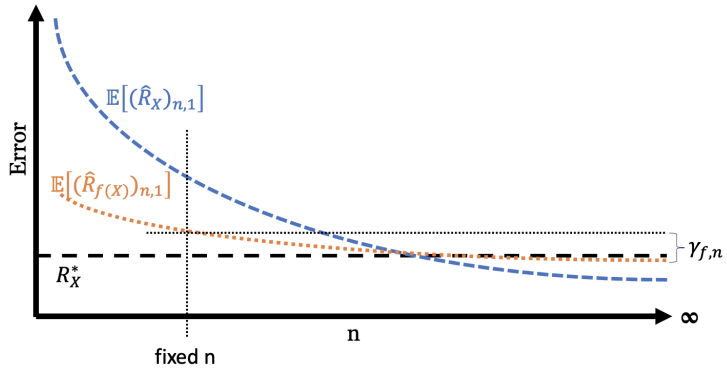


Figure A.4: N-sample gap (of the estimator) $\gamma_{f,n}$ and a single transformation f.

A.3.1 Proof of Sufficient Conditions

We next proof that the conditions of Cases 1 and 2 are sufficient for the claims maid in the main body of this thesis.

Case 1: When is $\widehat{R}$ optimal? We prove that Condition 3.6.7 implies $\widehat{R}_{f(X),n} \geq R^*_X$, for all $f \in \mathcal{F}$.

Proof: For any transformation f,

$$\begin{aligned}\widehat{R}_{f(X),n} &= \widehat{R}_{f(X),\infty} + \gamma_{f,n} \\ &= R^*_{f(X)} - \Delta_f + \gamma_{f,n} \\ &= R^*_X + \delta_f + \gamma_{f,n} - \Delta_f \\ &\geq R^*_X,\end{aligned}$$

with the last inequality coming from Condition 3.6.7. □

Case 2: When is $\widehat{R}$ at least as good as the suggested estimator by Cover and Hart (1967) on the raw features? We prove Condition 3.6.8 implies $\widehat{R}_{f(X),n} \geq \widehat{R}_{X,\infty}$, for all $f \in \mathcal{F}$.

Proof: For any transformation f,

$$\begin{aligned}\widehat{R}_{f(X),n} &= \widehat{R}_{f(X),\infty} + \gamma_{f,n} \\ &= R^*_X + \delta_f + \gamma_{f,n} - \Delta_f \\ &\geq R^*_X - \Delta_{id} \\ &= \widehat{R}_{X,\infty},\end{aligned}$$

where the inequality comes from Condition 3.6.8. □

Chapter B

Supplementary Material – Experimental Details and Additional Results

B.1 Detailed Experimental Settings

B.1.1 Datasets

We use the standard splits provided by the datasets, as given in Table B.1. We collected all the datasets but YELP from the Tensorflow Datasets collection.[1], whereas YELP can be downloaded from `https://www.yelp.com/dataset`. Table B.2 highlights the details for noisy CIFAR versions.

B.1.2 Feature Transformations

We provide the list of all tested feature transformations, together with their dimensionality, for the vision datasets and text classification datasets in Tables B.3 and B.4, respectively. We were not able to export the BOW (and hence neither the BOW-TFIDF nor the PCA transformed) feature representations for YELP due to the large amount of samples and their high dimensionality. Additionally, calculating the NCA representations did not successfully terminate for any of the

[1] `https://www.tensorflow.org/datasets/`

Table B.1: Dataset statistics for computer vision datasets in the upper part, and natural language processing datasets in the lower part.

Name	Dimension	Classes	Train Samples	Test Samples	SOTA %
MNIST	784	10	60K	10K	0.16 (Byerly et al., 2020)
CIFAR10	3072	10	50K	10K	0.63 (Kolesnikov et al., 2019)
CIFAR100	3072	100	50K	10K	6.49 (Kolesnikov et al., 2019)
IMDB	104083	2	25K	25K	3.79 (Yang et al., 2019)
SST2	14583	2	67K	872	3.2 (Yang et al., 2019)
YELP	175710	5	500K	50K	27.80 (Yang et al., 2019)

Table B.2: CIFAR-N datasets statistics. The variable $t_{i,j}$ refer to the value on the i-th row and j-th column of the noise transition matrix t (c.f., Section A.2).

Dataset	Noise	$\max_i t_{i,i}$	$\min_i t_{i,i}$	$\max_{i \neq j} t_{i,j}$
CIFAR10-Aggre	9%	17%	3%	10%
CIFAR10-Random1	17%	26%	10%	23%
CIFAR10-Random2	18%	26%	10%	23%
CIFAR10-Random3	18%	26%	10%	23%
CIFAR100-Noisy	40%	85%	8%	31%

text classification datasets, as this method does not scale to high dimensional and large-sample-size inputs. All reported transformations are publicly available through either the scikit-learn toolkit.[2], TensorFlow Hub[3] or PyTorch Hub[4]

[2] `https://scikit-learn.org/stable/`
[3] `https://tfhub.dev/`
[4] `https://pytorch.org/hub/`

Table B.3: Feature transformations for images as features.

Transformation	Source	MNIST	CIFAR10	CIFAR100
Identity - Raw	-	✓	✓	✓
PCA (d=32)	scikit-learn	✓	✓	✓
PCA (d=64)	scikit-learn	✓	✓	✓
PCA (d=128)	scikit-learn	✓	✓	✓
NCA (d=64)	scikit-learn	✓	✓	✓
AlexNet(d=4096)	PyTorch-Hub	✓	✓	✓
GoogleNet (d=1024)	PyTorch-Hub	✓	✓	✓
VGG16 (d=4096)	PyTorch-Hub	✓	✓	✓
VGG19 (d=4096)	PyTorch-Hub	✓	✓	✓
ResNet50-V2 (d=2048)	TF-Hub	✓	✓	✓
ResNet101-V2 (d=2048)	TF-Hub	✓	✓	✓
ResNet152-V2 (d=2048)	TF-Hub	✓	✓	✓
InceptionV3 (d=2048)	TF-Hub	✓	✓	✓
EfficientNet-B0 (d=1280)	TF-Hub	✓	✓	✓
EfficientNet-B1 (d=1280)	TF-Hub	✓	✓	✓
EfficientNet-B2 (d=1408)	TF-Hub	✓	✓	✓
EfficientNet-B3 (d=1536)	TF-Hub	✓	✓	✓
EfficientNet-B4 (d=1792)	TF-Hub	✓	✓	✓
EfficientNet-B5 (d=2048)	TF-Hub	✓	✓	✓
EfficientNet-B6 (d=2304)	TF-Hub	✓	✓	✓
EfficientNet-B7 (d=2560)	TF-Hub	✓	✓	✓

Table B.4: Feature transformations for natural language as features.

Transformation	Source	IMDB	SST2	YELP
Identiy - BOW	-	✓	✓	✗
BOW-TFIDF	scikit-learn	✓	✓	✗
PCA (d=8)	scikit-learn	✓	✓	✗
PCA (d=16)	scikit-learn	✓	✓	✗
PCA (d=32)	scikit-learn	✓	✓	✗
PCA (d=64)	scikit-learn	✓	✓	✗
PCA (d=128)	scikit-learn	✓	✓	✗
ELMO (d=1024)	TF-Hub	✓	✓	✓
NNLM-EN (d=50)	TF-Hub	✓	✓	✓
NNLM-EN-WITH-NORMALIZATION (d=50)	TF-Hub	✓	✓	✓
NNLM-EN (d=128)	TF-Hub	✓	✓	✓
NNLM-EN–WITH-NORMALIZATION (d=128)	TF-Hub	✓	✓	✓
Universal Sentence Encoder (USE) (d=512)	TF-Hub	✓	✓	✓
BERT-Base (d=678)	PyTorch-Hub	✓	✓	✓

B.1.3 Model Details SHiFT

Table B.5: All models are avaibale with "https://tfhub.dev/" as prefix (part 1/3).

Model	Inference Cost (ms)	Mini-Batch Size
google/cropnet/feature_vector/cassava_disease_V1/1	11	224
google/cropnet/feature_vector/cassava_disease_V1/1	11	224
google/cropnet/feature_vector/concat/1	12	224
google/cropnet/feature_vector/imagenet/1	12	224
google/imagenet/efficientnet_v2_imagenet1k_b0/feature_vector/2	11	224
google/imagenet/efficientnet_v2_imagenet1k_b1/feature_vector/2	14	240
google/imagenet/efficientnet_v2_imagenet1k_b2/feature_vector/2	14	260
google/imagenet/efficientnet_v2_imagenet1k_b3/feature_vector/2	17	300
google/imagenet/efficientnet_v2_imagenet1k_l/feature_vector/2	85	480
google/imagenet/efficientnet_v2_imagenet1k_m/feature_vector/2	55	480
google/imagenet/efficientnet_v2_imagenet1k_s/feature_vector/2	23	384
google/imagenet/efficientnet_v2_imagenet21k_b0/feature_vector/2	11	224
google/imagenet/efficientnet_v2_imagenet21k_b1/feature_vector/2	11	240
google/imagenet/efficientnet_v2_imagenet21k_b2/feature_vector/2	12	260
google/imagenet/efficientnet_v2_imagenet21k_b3/feature_vector/2	14	300
google/imagenet/efficientnet_v2_imagenet21k_ft1k_b0/feature_vector/2	11	224
google/imagenet/efficientnet_v2_imagenet21k_ft1k_b1/feature_vector/2	12	240
google/imagenet/efficientnet_v2_imagenet21k_ft1k_b2/feature_vector/2	13	260
google/imagenet/efficientnet_v2_imagenet21k_ft1k_b3/feature_vector/2	14	300
google/imagenet/efficientnet_v2_imagenet21k_ft1k_l/feature_vector/2	73	480
google/imagenet/efficientnet_v2_imagenet21k_ft1k_m/feature_vector/2	33	480
google/imagenet/efficientnet_v2_imagenet21k_ft1k_xl/feature_vector/2	74	512
google/imagenet/efficientnet_v2_imagenet21k_l/feature_vector/2	63	480
google/imagenet/efficientnet_v2_imagenet21k_m/feature_vector/2	33	480
google/imagenet/efficientnet_v2_imagenet21k_s/feature_vector/2	21	384
google/imagenet/inception_resnet_v2/feature_vector/4	26	224
google/imagenet/inception_v1/feature_vector/4	13	224
google/imagenet/inception_v2/feature_vector/4	11	224
google/imagenet/inception_v3/feature_vector/4	14	224
google/imagenet/inception_v3/feature_vector/5	18	299
google/imagenet/mobilenet_v1_025_128/feature_vector/5	6	128
google/imagenet/mobilenet_v1_025_160/feature_vector/5	6	160
google/imagenet/mobilenet_v1_025_192/feature_vector/5	6	192
google/imagenet/mobilenet_v1_025_224/feature_vector/5	6	224
google/imagenet/mobilenet_v1_050_128/feature_vector/5	6	128
google/imagenet/mobilenet_v1_050_160/feature_vector/5	8	160
google/imagenet/mobilenet_v1_050_192/feature_vector/5	6	192
google/imagenet/mobilenet_v1_050_224/feature_vector/5	6	224
google/imagenet/mobilenet_v1_075_128/feature_vector/5	7	128
google/imagenet/mobilenet_v1_075_160/feature_vector/5	6	160

Table B.6: All models are avaibale with "https://tfhub.dev/" as prefix (part 2/3).

Model	Inference Cost (ms)	Mini-Batch Size
google/imagenet/mobilenet_v1_075_192/feature_vector/5	6	192
google/imagenet/mobilenet_v1_075_224/feature_vector/5	6	224
google/imagenet/mobilenet_v1_100_128/feature_vector/5	6	128
google/imagenet/mobilenet_v1_100_160/feature_vector/5	7	160
google/imagenet/mobilenet_v1_100_192/feature_vector/5	6	192
google/imagenet/mobilenet_v1_100_224/feature_vector/4	7	224
google/imagenet/mobilenet_v2_035_128/feature_vector/5	8	128
google/imagenet/mobilenet_v2_035_160/feature_vector/5	8	160
google/imagenet/mobilenet_v2_035_192/feature_vector/5	9	192
google/imagenet/mobilenet_v2_035_224/feature_vector/5	8	224
google/imagenet/mobilenet_v2_035_96/feature_vector/5	9	96
google/imagenet/mobilenet_v2_050_128/feature_vector/5	9	128
google/imagenet/mobilenet_v2_050_160/feature_vector/5	8	160
google/imagenet/mobilenet_v2_050_192/feature_vector/5	8	192
google/imagenet/mobilenet_v2_050_224/feature_vector/5	9	224
google/imagenet/mobilenet_v2_050_96/feature_vector/5	8	96
google/imagenet/mobilenet_v2_075_128/feature_vector/5	9	128
google/imagenet/mobilenet_v2_075_160/feature_vector/5	11	160
google/imagenet/mobilenet_v2_075_192/feature_vector/5	11	192
google/imagenet/mobilenet_v2_075_224/feature_vector/5	10	224
google/imagenet/mobilenet_v2_075_96/feature_vector/5	11	96
google/imagenet/mobilenet_v2_100_128/feature_vector/5	9	128
google/imagenet/mobilenet_v2_100_160/feature_vector/5	9	160
google/imagenet/mobilenet_v2_100_192/feature_vector/5	11	192
google/imagenet/mobilenet_v2_100_224/feature_vector/4	9	224
google/imagenet/mobilenet_v2_100_96/feature_vector/5	8	96
google/imagenet/mobilenet_v2_130_224/feature_vector/5	9	224
google/imagenet/mobilenet_v2_140_224/feature_vector/5	11	224
google/imagenet/mobilenet_v3_large_075_224/feature_vector/5	11	224
google/imagenet/mobilenet_v3_large_100_224/feature_vector/5	10	224
google/imagenet/mobilenet_v3_small_075_224/feature_vector/5	9	224
google/imagenet/mobilenet_v3_small_100_224/feature_vector/5	9	224

Table B.7: All models are avaibale with "https://tfhub.dev/" as prefix (part 3/3).

Model	Inference Cost (ms)	Mini-Batch Size
google/imagenet/nasnet_mobile/feature_vector/4	19	224
google/imagenet/resnet_v1_101/feature_vector/4	16	224
google/imagenet/resnet_v1_152/feature_vector/4	22	224
google/imagenet/resnet_v1_50/feature_vector/4	13	224
google/imagenet/resnet_v2_101/feature_vector/4	16	224
google/imagenet/resnet_v2_152/feature_vector/4	22	224
google/imagenet/resnet_v2_50/feature_vector/4	13	224
tensorflow/efficientnet/b0/feature-vector/1	12	224
tensorflow/efficientnet/b1/feature-vector/1	15	240
tensorflow/efficientnet/b2/feature-vector/1	17	260
tensorflow/efficientnet/b3/feature-vector/1	21	300
tensorflow/efficientnet/b4/feature-vector/1	32	380
tensorflow/efficientnet/b5/feature-vector/1	44	456
tensorflow/efficientnet/b6/feature-vector/1	79	528
tensorflow/efficientnet/b7/feature-vector/1	151	600
vtab/exemplar/1	13	224
vtab/jigsaw/1	25	224
vtab/relative-patch-location/1	21	224
vtab/rotation/1	12	224
vtab/semi-exemplar-10/1	12	224
vtab/semi-rotation-10/1	12	224
vtab/sup-100/1	12	224
vtab/sup-exemplar-100/1	11	224
vtab/sup-rotation-100/1	12	224
vtab/uncond-biggan/1	17	128
vtab/vae/1	10	128
vtab/wae-gan/1	10	128
vtab/wae-mmd/1	10	128
vtab/wae-ukl/1	10	128

B.2 Additional FeeBee Results

B.2.1 No Transformation / Raw

To see the impact of choosing the best feature transformation per method, we report the results on the raw features (i.e., pixel values) for the vision datasets in Figure B.1 and Table B.8, nothing that NLP datasets do not come with any natural *raw* representations. For the method achieving good lower and upper bounds (i.e., kNN and GHP), we see that for both CIFAR variants, they constantly overestimate both the upper and lower bound. For the other methods, selecting the best hyper-parameters leads to non-informative variants.

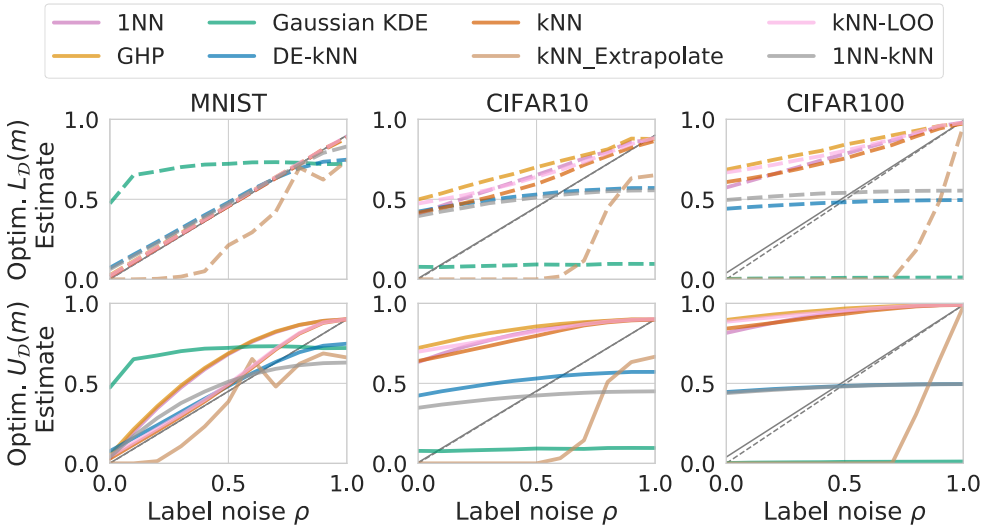


Figure B.1: Computer vision datasets on raw features.

Figure B.2: Evolution of the BER for selecting hyper-parameters only on the raw pixels that minimize **(top row, each)** $L_{\mathcal{D}}(m)$ and **(bottom row, each)** $U_{\mathcal{D}}(m)$.

Table B.8: $L_{\mathcal{D}}(m)$ and $U_{\mathcal{D}}(m)$: The optimal values per method on raw pixel values.

	DE-kNN	KDE	GHP	1NN-kNN	1NN	kNN	kNN-LOO	kNN_Ext
MNIST	0.12	0.64	0.03	0.07	0.02	0.02	0.03	0.50
CIFAR10	0.47	0.83	0.55	0.45	0.46	0.39	0.46	0.69
CIFAR100	0.49	0.98	0.66	0.49	0.54	0.53	0.61	0.71
MNIST	0.12	0.64	0.34	0.22	0.32	0.09	0.11	0.43
CIFAR10	0.47	0.83	0.85	0.48	0.79	0.75	0.80	0.66
CIFAR100	0.49	0.98	0.89	0.49	0.84	0.84	0.88	0.65

B.2.2 Optimal Lower- and Upper-Bound Estimates

In this section we provide additional visualizations to support the findings in Section 3.4.2, completing Figure 3.3. We observe that 1NN, kNN, kNN-LOO and GHP closely follow the evolution of the BER and that for them the main contributions to $L_{\mathcal{D}}(m)$ and $U_{\mathcal{D}}(m)$ come from $L_{\mathcal{D},\triangleright}(m)$ and $U_{\mathcal{D},\triangleright}(m)$. This will be improved as more transformations become available, thus reducing the bias. For DE-kNN, KDE and kNN-Extrapolate we clearly see difficulties in following the evolution of the BER, often visualized in the bottom-right corner, resulting in large $L_{\mathcal{D},\triangleleft}(m)$ and/or $U_{\mathcal{D},\triangleleft}(m)$.

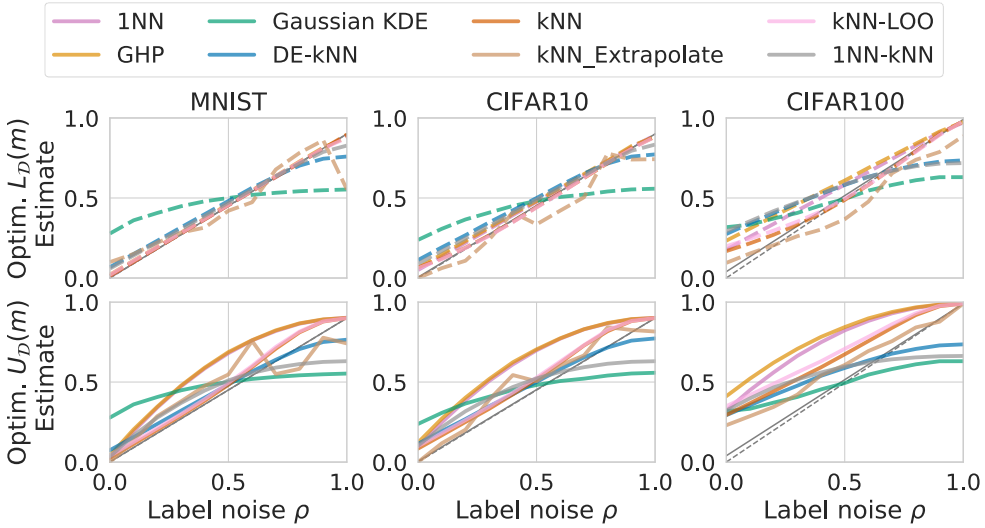
B.2.3 Further Example Plots

In this section we present example plots produced by `FeeBee` in Figure B.4. For simplicity, we focus only on two datasets and two methods. In order to have comparable lower and upper bounds, we fix the transformation in each figure.

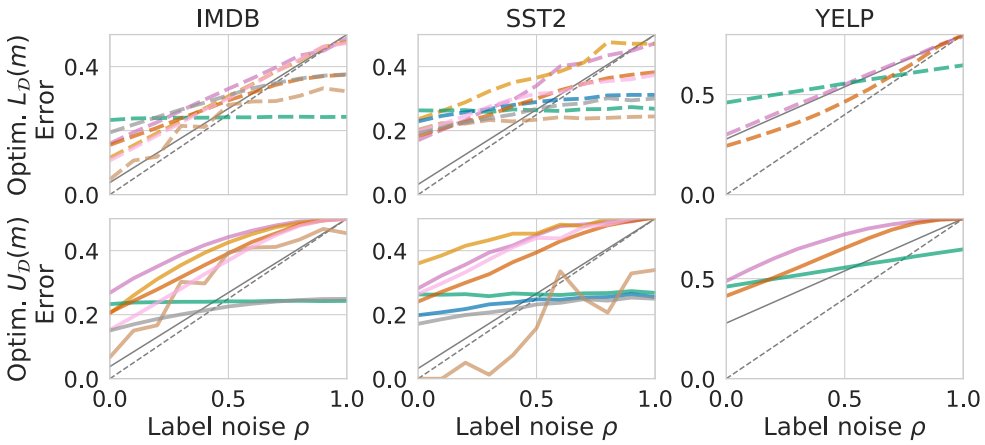
We observe that 1NN performs better than kNN-Extrapolate for the lower bound, clearly following the evolution of the BER. We also observe that for both the lower bound is always below upper bound.

B.2.4 Complete Tables

In this section we provide Tables B.9 – B.20 with optimal values of scores $L_{\mathcal{D}}(m)$ and $U_{\mathcal{D}}(m)$, for every dataset and every method, reporting the hyper-parameters and transformation that yield that score, together with the upper ($L_{\mathcal{D},\triangleright}$, $U_{\mathcal{D},\triangleright}$) and lower areas ($L_{\mathcal{D},\triangleleft}$, $U_{\mathcal{D},\triangleleft}$) that contribute to the scores.

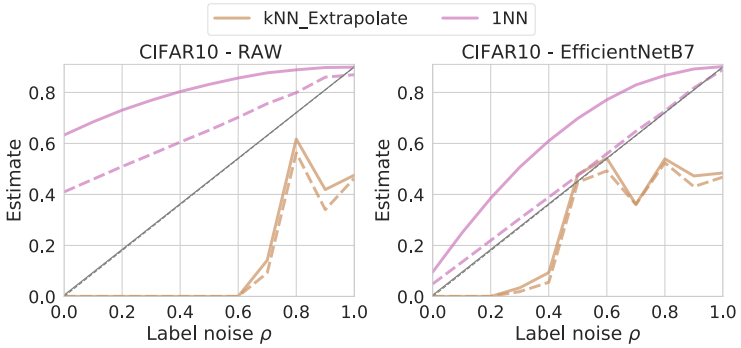


(a) Computer vision datasets.

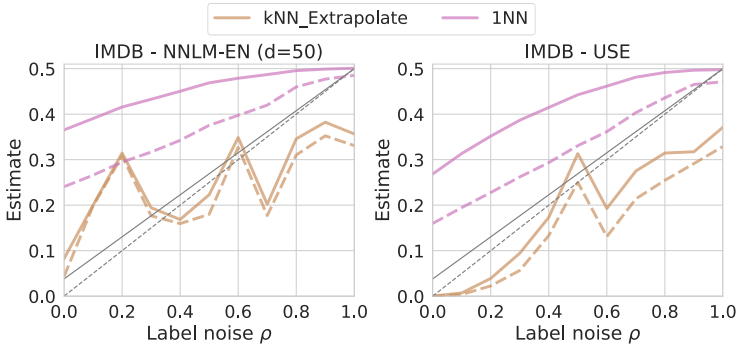


(b) NLP datasets.

Figure B.3: Evolution of the BER for selecting hyper-parameters and transformations that minimize **(top row, each)** $L_{\mathcal{D}}(m)$ and **(bottom row, each)** $U_{\mathcal{D}}(m)$.



(a)



(b)

Figure B.4: Example plots of $L_{\mathcal{D}}(m)$ and $U_{\mathcal{D}}(m)$ for **(a)** CIFAR10 over raw and EfficientNetB7, and **(b)** IMDB over NNLM-EN ($d = 50$) and USE.

Table B.9: MNIST - Optimal $L_{\mathcal{D}}(m)$

Method	Variant	Transformation	$L_{\mathcal{D}}(m)$	$L_{\mathcal{D},\triangleleft}(m)$	$L_{\mathcal{D},\triangleright}(m)$
kNN	dist=cosine, k=2	NCA (d=64)	0.02	0.00	0.01
1NN	dist=cosine	NCA (d=64)	0.02	0.01	0.01
kNN-LOO	dist=cosine, k=2	NCA (d=64)	0.03	0.01	0.02
GHP	default	PCA (d=32)	0.03	0.00	0.02
1NN-kNN	dist=cosine, k=21	PCA (d=32)	0.07	0.02	0.05
DE-kNN	dist=squared_l2, k=15	PCA (d=32)	0.11	0.05	0.06
kNN_Extrapolate	dist=cosine, k=10	PCA (d=32)	0.28	0.15	0.13
Gaussian KDE	B=0.05	VGG16	0.41	0.18	0.23

Table B.10: CIFAR10 - Optimal $L_{\mathcal{D}}(m)$

Method	Variant	Transformation	$L_{\mathcal{D}}(m)$	$L_{\mathcal{D},\triangleleft}(m)$	$L_{\mathcal{D},\triangleright}(m)$
kNN-LOO	dist=squared_l2, k=3	EfficientNet-B3	0.03	0.01	0.02
kNN	dist=squared_l2, k=3	EfficientNet-B2	0.03	0.01	0.02
1NN	dist=squared_l2	EfficientNet-B7	0.05	0.00	0.05
GHP	default	EfficientNet-B4	0.07	0.00	0.07
1NN-kNN	dist=squared_l2, k=22	EfficientNet-B7	0.10	0.02	0.08
DE-kNN	dist=squared_l2, k=18	EfficientNet-B7	0.14	0.04	0.11
kNN_Extrapolate	dist=squared_l2, k=3	PCA (d=32)	0.34	0.23	0.11
Gaussian KDE	B=0.1	ResNet152-V2	0.36	0.19	0.18

Table B.11: CIFAR100 - Optimal $L_{\mathcal{D}}(m)$

Method	Variant	Transformation	$L_{\mathcal{D}}(m)$	$L_{\mathcal{D},\triangleleft}(m)$	$L_{\mathcal{D},\triangleright}(m)$
kNN	dist=cosine, k=5	EfficientNet-B5	0.06	0.01	0.05
kNN-LOO	dist=cosine, k=6	EfficientNet-B6	0.07	0.01	0.07
1NN	dist=cosine	EfficientNet-B7	0.14	0.00	0.14
GHP	default	EfficientNet-B7	0.20	0.00	0.20
kNN_Extrapolate	dist=squared_l2, k=1	PCA (d=32)	0.22	0.17	0.05
DE-kNN	dist=squared_l2, k=4	EfficientNet-B7	0.27	0.10	0.18
1NN-kNN	dist=squared_l2, k=12	EfficientNet-B7	0.29	0.10	0.19
Gaussian KDE	B=0.1	VGG19	0.31	0.18	0.14

Table B.12: IMDB - Optimal $L_{\mathcal{D}}(m)$

Method	Variant	Transformation	$L_{\mathcal{D}}(m)$	$L_{\mathcal{D},\triangleleft}(m)$	$L_{\mathcal{D},\triangleright}(m)$
kNN-LOO	dist=squared_l2, k=1	USE	0.15	0.01	0.14
GHP	default	USE	0.16	0.01	0.16
kNN	dist=cosine, k=9	USE	0.25	0.09	0.16
1NN	dist=squared_l2	USE	0.25	0.01	0.24
kNN_Extrapolate	dist=squared_l2, k=6	PCA (d=16)	0.25	0.19	0.06
1NN-kNN	dist=cosine, k=4	PCA (d=32)	0.31	0.09	0.22
Gaussian KDE	B=0.0025	PCA (d=8)	0.49	0.29	0.20

Table B.13: SST2 - Optimal $L_{\mathcal{D}}(m)$

Method	Variant	Transformation	$L_{\mathcal{D}}(m)$	$L_{\mathcal{D},\triangleleft}(m)$	$L_{\mathcal{D},\triangleright}(m)$
kNN	dist=cosine, k=10	BOW-TFIDF	0.29	0.08	0.20
1NN	dist=cosine	ELMO	0.32	0.02	0.30
kNN-LOO	dist=cosine, k=7	USE	0.34	0.10	0.24
1NN-kNN	dist=cosine, k=3	ELMO	0.38	0.20	0.18
DE-kNN	dist=squared_l2, k=4	NNLM-EN-NORM	0.42	0.17	0.25
GHP	default	USE	0.44	0.01	0.42
kNN_Extrapolate	dist=squared_l2, k=2	PCA (d=8)	0.47	0.29	0.18
Gaussian KDE	B=0.0025	PCA (d=16)	0.49	0.24	0.25

Table B.14: YELP - Optimal $L_{\mathcal{D}}(m)$

Method	Variant	Transformation	$L_{\mathcal{D}}(m)$	$L_{\mathcal{D},\triangleleft}(m)$	$L_{\mathcal{D},\triangleright}(m)$
kNN	dist=squared_l2, k=9	USE	0.00	0.00	0.00
1NN	dist=squared_l2	USE	0.03	0.00	0.03
Gaussian KDE	B=0.05	NNLM-EN (d=50)	0.20	0.06	0.14

Table B.15: MNIST - Optimal $U_{\mathcal{D}}(m)$

Method	Variant	Transformation	$U_{\mathcal{D}}(m)$	$U_{\mathcal{D},\triangleleft}(m)$	$U_{\mathcal{D},\triangleright}(m)$
kNN	dist=cosine, k=10	NCA (d=64)	0.09	0.00	0.09
kNN-LOO	dist=cosine, k=10	NCA (d=64)	0.11	0.00	0.11
DE-kNN	dist=squared_l2, k=16	PCA (d=32)	0.11	0.04	0.07
1NN-kNN	dist=cosine, k=3	NCA (d=64)	0.22	0.12	0.09
1NN	dist=cosine	NCA (d=64)	0.32	0.00	0.32
GHP	default	PCA (d=32)	0.33	0.00	0.33
kNN_Extrapolate	dist=cosine, k=10	PCA (d=128)	0.37	0.16	0.21
Gaussian KDE	B=0.05	VGG16	0.41	0.18	0.23

Table B.16: CIFAR10 - Optimal $U_{\mathcal{D}}(m)$

Method	Variant	Transformation	$U_{\mathcal{D}}(m)$	$U_{\mathcal{D},\triangleleft}(m)$	$U_{\mathcal{D},\triangleright}(m)$
DE-kNN	dist=squared_l2, k=18	EfficientNet-B7	0.15	0.04	0.11
kNN	dist=cosine, k=10	EfficientNet-B7	0.15	0.00	0.15
kNN-LOO	dist=cosine, k=10	EfficientNet-B4	0.17	0.00	0.17
kNN_Extrapolate	dist=squared_l2, k=3	PCA (d=32)	0.20	0.04	0.16
1NN-kNN	dist=squared_l2, k=3	EfficientNet-B7	0.26	0.12	0.14
Gaussian KDE	B=0.1	ResNet152-V2	0.36	0.19	0.18
1NN	dist=cosine	EfficientNet-B7	0.37	0.00	0.37
GHP	default	EfficientNet-B4	0.39	0.00	0.39

Table B.17: CIFAR100 - Optimal $U_{\mathcal{D}}(m)$

Method	Variant	Transformation	$U_{\mathcal{D}}(m)$	$U_{\mathcal{D},\triangleleft}(m)$	$U_{\mathcal{D},\triangleright}(m)$
kNN_Extrapolate	dist=cosine, k=7	PCA (d=32)	0.21	0.02	0.19
DE-kNN	dist=squared_l2, k=4	EfficientNet-B6	0.28	0.09	0.19
Gaussian KDE	B=0.1	VGG19	0.31	0.18	0.14
kNN	dist=cosine, k=10	EfficientNet-B7	0.31	0.00	0.31
1NN-kNN	dist=squared_l2, k=3	EfficientNet-B7	0.36	0.14	0.22
kNN-LOO	dist=cosine, k=10	EfficientNet-B6	0.37	0.00	0.37
1NN	dist=cosine	EfficientNet-B7	0.49	0.00	0.49
GHP	default	EfficientNet-B7	0.55	0.00	0.55

Table B.18: IMDB - Optimal $U_{\mathcal{D}}(m)$

Method	Variant	Transformation	$U_{\mathcal{D}}(m)$	$U_{\mathcal{D},\triangleleft}(m)$	$U_{\mathcal{D},\triangleright}(m)$
kNN_Extrapolate	dist=squared_l2, k=6	PCA (d=16)	0.28	0.03	0.25
kNN-LOO	dist=squared_l2, k=9	USE	0.34	0.00	0.34
1NN-kNN	dist=cosine, k=2	PCA (d=32)	0.39	0.29	0.10
kNN	dist=squared_l2, k=9	USE	0.43	0.00	0.43
Gaussian KDE	B=0.0025	PCA (d=8)	0.49	0.29	0.20
GHP	default	USE	0.51	0.00	0.51
1NN	dist=squared_l2	USE	0.60	0.00	0.60

Table B.19: SST2 - Optimal $U_{\mathcal{D}}(m)$

Method	Variant	Transformation	$U_{\mathcal{D}}(m)$	$U_{\mathcal{D},\triangleleft}(m)$	$U_{\mathcal{D},\triangleright}(m)$
1NN-kNN	dist=cosine, k=2	NNLM-EN (d=128)	0.42	0.29	0.13
DE-kNN	dist=squared_l2, k=2	USE	0.43	0.27	0.17
kNN	dist=cosine, k=10	BOW-TFIDF	0.48	0.00	0.48
Gaussian KDE	B=0.0025	PCA (d=16)	0.49	0.24	0.25
kNN-LOO	dist=cosine, k=10	USE	0.57	0.00	0.57
kNN_Extrapolate	dist=squared_l2, k=2	USE	0.61	0.49	0.12
1NN	dist=cosine	ELMO	0.63	0.00	0.63
GHP	default	ELMO	0.73	0.00	0.73

Table B.20: YELP - Optimal $U_{\mathcal{D}}(m)$

Method	Variant	Transformation	$U_{\mathcal{D}}(m)$	$U_{\mathcal{D},\triangleleft}(m)$	$U_{\mathcal{D},\triangleright}(m)$
Gaussian KDE	B=0.05	NNLM-EN (d=50)	0.20	0.06	0.14
kNN	dist=cosine, k=10	USE	0.25	0.00	0.24
1NN	dist=squared_l2	USE	0.38	0.00	0.38

B.3 Successive-Halving with Tangents

In Section 3.6.3 we illustrate the successive-halving algorithm together with the improvement that uses tangent predictions to avoid unnecessary calculations on transformations that will certainly not proceed to the next step. In this section we provide an illustration of the algorithm in Figure 3.13 along with pseudocodes for both variants of the successive-halving algorithm in Algorithm 1. Switching from one variant to the other is simply done through the `use_tangent` flag, which either calls the function `Pulls_with_tangent_breaks`, or avoids this and performs the original successive-halving algorithm. As the remaining transformations after each step are the same in both variants, all theoretical guarantees of successive-halving can be transferred to our extension.

The `Pulls_with_tangent_breaks` function simply uses the tangent, which we approximate by a line through the two last known points of the convergence curve, to predict the smallest error that a feature transformation can achieve at the end of the current step. Here we assume that the convergence curves are convex, which holds on average Snapp et al. (1991). Therefore, it is a slightly more aggressive variant of the successive-halving algorithm, but we did not observe failures in practice since the tangent is usually a very crude lower bound.

Algorithm 1 Successive-Halving with Tangents

Input: Flag use_tangent, budget B, arms $1, \ldots, n$ with $l_{i,k}$ denoting the k-th loss from the i-th arm

Initialize: $S_0 = [n]$, predictions$[i] = 0$, for all $i \in [n]$

for $k = 0, 1, \ldots, \lceil \log_2(n) \rceil - 1$ **do**

$L = |S_k|$;

$r_k = \lfloor \frac{B}{L \cdot \lceil \log_2(n) \rceil} \rfloor$

$R_k = \sum_{j=0}^{k} r_j$

Pull r_k times each arm $i = 1, \ldots, \lfloor L/2 \rfloor$

if not use_tangent **then**

Pull r_k times each arm $i = \lfloor L/2 \rfloor + 1, \ldots, L$

else

threshold = $\max_{i=1,\ldots,\lfloor L/2 \rfloor} l_{i,R_k}$

S_k = Pulls_with_tangent_breaks(S_k, r_k, R_k, predictions[], threshold)

end if

Let σ_k be a permutation on S_k such that $l_{\sigma_k(1),R_k} \leq \ldots \leq l_{\sigma_k(|S_k|),R_k}$

$S_{k+1} = \{\sigma_k(1), \ldots, \sigma_k(\lfloor L/2 \rfloor)\}$.

end for

Output: Singleton element of $S_{\lceil \log_2(n) \rceil}$

Algorithm 2 Pulls_with_tangent_breaks

Input: S, r, R, predictions[], threshold

for $i \in S$ **do**

for $j = 1, \ldots, r$ **do**

if predictions[i]>threshold **then**

remove i from S

break

else

Pull arm i once

Update predictions$[i]$ using tangent value at R

end if

end for

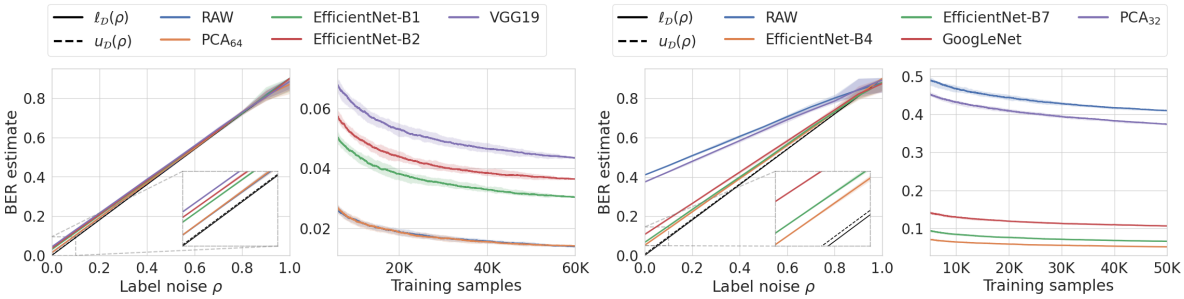
end for

Return S

B.4 Additional Ease.ML/Snoopy Results

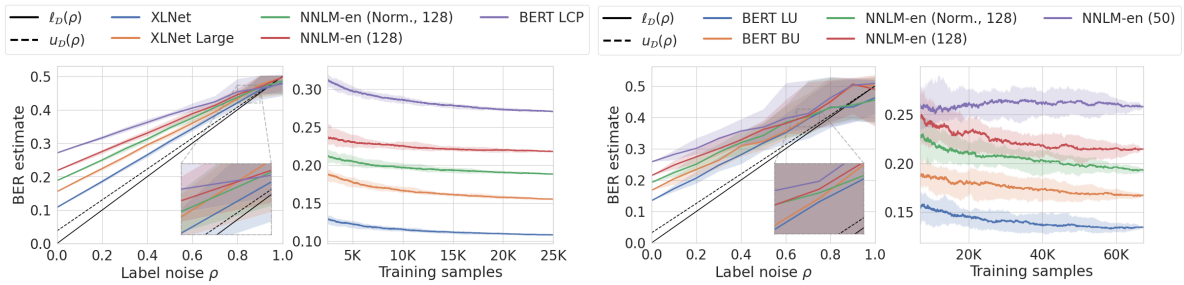
B.4.1 Convergence Plots

We provide the convergence plots and the evolution of the BER for all datasets and various embeddings (including the best performing one) in Figures B.5-B.7.



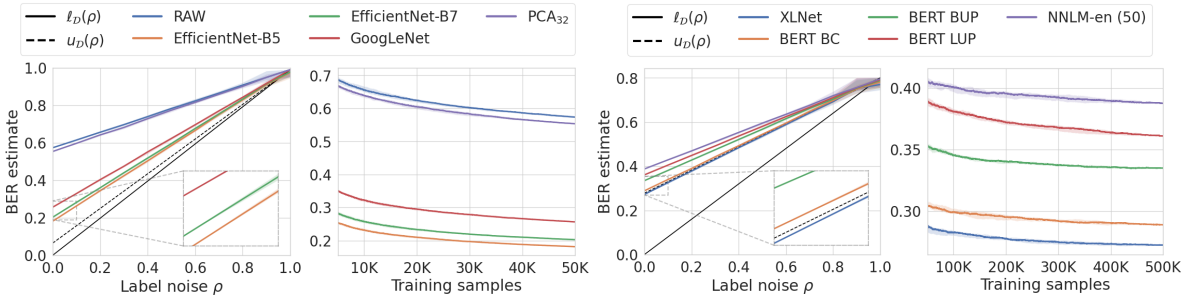
(a) MNIST **(b)** CIFAR10

Figure B.5: Evaluation and convergence of 1NN estimator for different feature transformations. **(Left each)** All the training points for different amount of label noise. **(Right each)** Zero label noise and increasing number of training samples.



(a) IMDB **(b)** SST2

Figure B.6: Evaluation and convergence of 1NN estimator for different feature transformations. **(Left each)** All the training points for different amount of label noise. **(Right each)** Zero label noise and increasing number of training samples.

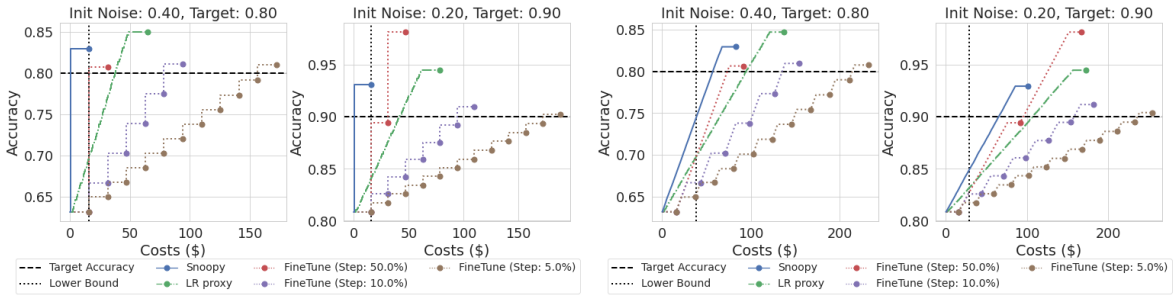


(a) CIFAR100 **(b)** YELP

Figure B.7: Evaluation and convergence of 1NN estimator for different feature transformations. **(Left each)** All the training points for different amount of label noise. **(Right each)** Zero label noise and increasing number of training samples.

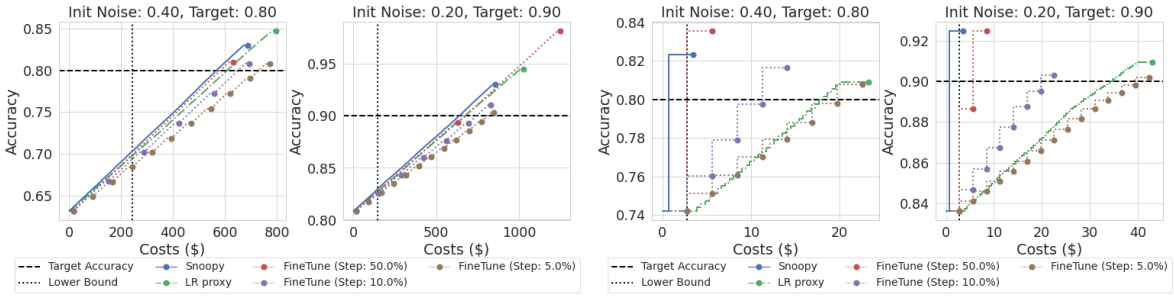
B.4.2 Additional End-To-End Figures

We report the figures for the end-to-end use-case for the omitted datasets next. We did not run the experiments for MNIST, where fine-tuning a large model performed worse than running the AutoML system. Nonetheless, following the insight of Section 3.6.4 and Figure 3.14, we expect no changes in the insights.



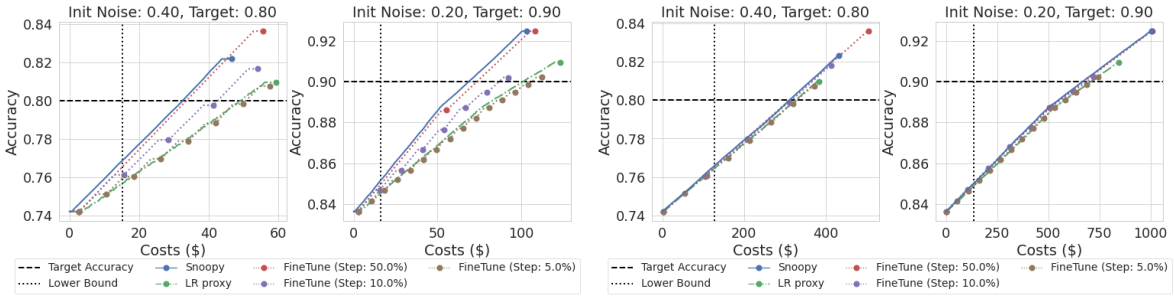
(a) CIFAR10 - Free labels. **(b)** CIFAR10 - Cheap labels.

Figure B.8: End-to-end use case: (a) CIFAR10 - free labels, and (b) CIFAR10 - cheap labels.



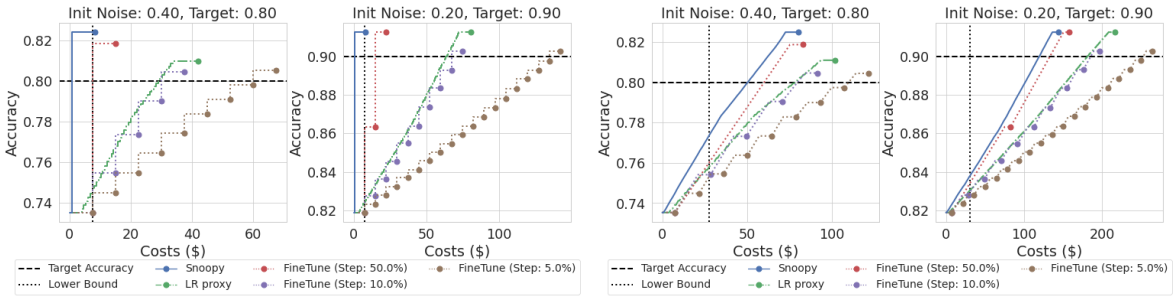
(a) CIFAR10 - Expensive labels. **(b)** IMDB - Free labels.

Figure B.9: End-to-end use case: (a) CIFAR10 - expensive labels, and (b) IMDB - free labels.



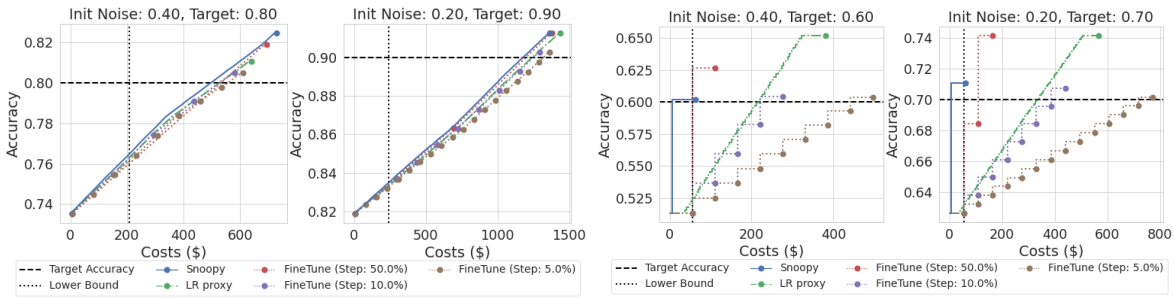
(a) IMDB - Cheap labels. **(b)** IMDB - Expensive labels.

Figure B.10: End-to-end use case: (a) IMDB - cheap labels, and (b) IMDB - expensive labels.



(a) SST2 - Free labels. **(b)** SST2 - Cheap labels.

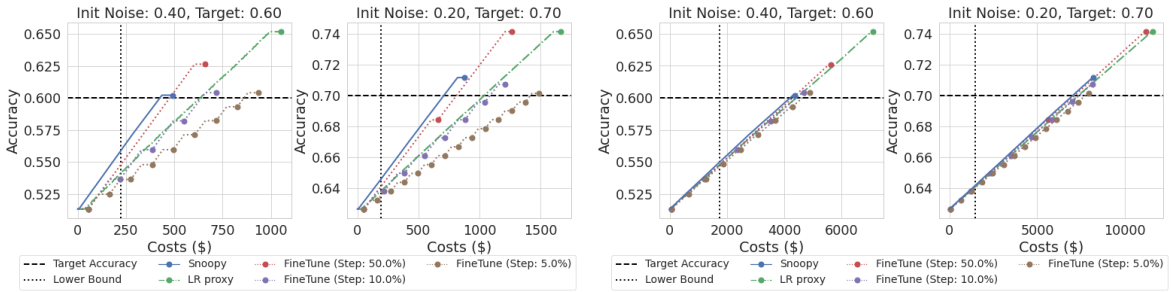
Figure B.11: End-to-end use case: (a) SST2 - free labels, and (b) SST2 - cheap labels.



(a) SST2 - Expensive labels.

(b) YELP - Free labels.

Figure B.12: End-to-end use case: (a) SST2 - expensive labels, and (b) YELP - free labels.



(a) YELP - Cheap labels.

(b) YELP - Expensive labels.

Figure B.13: End-to-end use case: (a) YELP - cheap labels, and (b) YELP - expensive labels.

Bibliography

Alessandro Achille, Michael Lam, Rahul Tewari, Avinash Ravichandran, Subhransu Maji, Charless C Fowlkes, Stefano Soatto, and Pietro Perona. 2019. Task2vec: Task embedding for meta-learning. In *IEEE International Conference on Computer Vision*, pages 6430–6439.

Leonel Aguilar Melgar, David Dao, Shaoduo Gan, Nezihe M Gürel, Nora Hollenstein, Jiawei Jiang, Bojan Karlaš, Thomas Lemmin, Tian Li, Yang Li, et al. 2021. Ease.ml: A lifecycle management system for machine learning. In *Conference on Innovative Data Systems Research*. CIDR.

András Antos, Luc Devroye, and Laszlo Gyorfi. 1999. Lower bounds for bayes error estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(7):643–645.

Jean-Yves Audibert and Alexandre B Tsybakov. 2007. Fast learning rates for plug-in classifiers. *The Annals of statistics*, 35(2):608–633.

Hossein Azizpour, Ali Sharif Razavian, Josephine Sullivan, Atsuto Maki, and Stefan Carlsson. 2015. Factors of transferability for a generic convnet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(9):1790–1802.

Alexei Baevski, Wei-Ning Hsu, Qiantong Xu, Arun Babu, Jiatao Gu, and Michael Auli. 2022. Data2vec: A general framework for self-supervised learning in speech, vision and language. *arXiv preprint arXiv:2202.03555*.

Dara Bahri, Heinrich Jiang, and Maya Gupta. 2020. Deep k-nn for noisy labels. In *International Conference on Machine Learning*, pages 540–550. PMLR.

Yajie Bao, Yang Li, Shao-Lun Huang, Lin Zhang, Lizhong Zheng, Amir Zamir, and Leonidas Guibas. 2019. An information-theoretic approach to transferability in task transfer learning. In *IEEE International Conference on Image Processing*, pages 2309–2313. IEEE.

Carlo Batini, Cinzia Cappiello, Chiara Francalanci, and Andrea Maurino. 2009. Methodologies for data quality assessment and improvement. *ACM Computing Surveys*, 41(3).

Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, et al. 2017. Tfx: A tensorflow-based production-scale machine learning platform. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1387–1395.

Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. 2009. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer.

James Bergstra, Dan Yamins, and David D Cox. 2013. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Python in Science Conference*, volume 13, page 20. Citeseer.

Visar Berisha, Alan Wisler, Alfred O Hero, and Andreas Spanias. 2015. Empirically estimable classification bounds based on a nonparametric divergence measure. *IEEE Transactions on Signal Processing*, 64(3):580–591.

Avrim Blum and Moritz Hardt. 2015. The ladder: A reliable leaderboard for machine learning competitions. In *International Conference on Machine Learning*, pages 1006–1014.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

Leon Bottou. 2016. The infinite MNIST dataset. `https://leon.bottou.org/projects/infimnist`.

Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. 2013. *Concentration inequalities: A nonasymptotic theory of independence*. Oxford university press.

Glenn W Brier et al. 1950. Verification of forecasts expressed in terms of probability. *Monthly weather review*, 78(1):1–3.

Ljubomir J Buturovic and Milan Z Markovic. 1992. Improving k-nearest neighbor bayes error estimates. In *IAPR International Conference on Pattern Recognition. Vol. II. Conference B: Pattern Recognition Methodology and Systems*, volume 1, pages 470–471. IEEE Computer Society.

Adam Byerly, Tatiana Kalganova, and Ian Dear. 2020. A branching and merging convolutional network with homogeneous filter capsules. *arXiv preprint arXiv:2001.09136*.

Tianshi Cao, Sasha Alexandre Doubov, David Acuna, and Sanja Fidler. 2021. Scalable neural data server: A data recommender for transfer learning. In *Advances in Neural Information Processing Systems*, volume 34.

Kamalika Chaudhuri and Sanjoy Dasgupta. 2014. Rates of convergence for nearest neighbor classification. In *Advances in Neural Information Processing Systems*, pages 3437–3445.

Zhiyuan Chen and Bing Liu. 2018. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 12(3):1–207.

Ira Cohen and Moises Goldszmidt. 2004. Properties and benefits of calibrated classifiers. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 125–136. Springer.

Cody Coleman, Christopher Yeh, Stephen Mussmann, Baharan Mirzasoleiman, Peter Bailis, Percy Liang, Jure Leskovec, and Matei Zaharia. 2019. Selection via proxy: Efficient data selection for deep learning. In *International Conference on Learning Representations*.

Gao Cong, Wenfei Fan, Floris Geerts, Xibei Jia, and Shuai Ma. 2007. Improving data quality: Consistency and accuracy. In *Proceedings of the VLDB Endowment*, volume 7, pages 315–326.

Thomas Cover. 1968. Rates of convergence for nearest neighbor procedures. In *Hawaii International Conference on Systems Sciences*, volume 415.

Thomas Cover and Peter Hart. 1967. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27.

Thomas Cover and Joy Thomas. 2006. *Elements of information theory*. Wiley-Interscience.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. Ieee.

Zhun Deng, Linjun Zhang, Kailas Vodrahalli, Kenji Kawaguchi, and James Y Zou. 2021. Adversarial training helps transfer learning via better representations. In *Advances in Neural Information Processing Systems*, volume 34.

Aditya Deshpande, Alessandro Achille, Avinash Ravichandran, Hao Li, Luca Zancato, Charless Fowlkes, Rahul Bhotika, Stefano Soatto, and Pietro Perona. 2021. A linearized framework and a new benchmark for model selection for fine-tuning. *arXiv preprint arXiv:2102.00084*.

Pierre A Devijver. 1985. A multiclass, k-nn approach to bayes risk estimation. *Pattern recognition letters*, 3(1):1–6.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Luc Devroye. 1981. On the asymptotic probability of error in nonparametric discrimination. *The Annals of Statistics*, 9(6):1320–1327.

Maik Döring, László Györfi, and Harro Walk. 2017. Rate of convergence of k-nearest-neighbor classification rule. *The Journal of Machine Learning Research*, 18(1):8485–8500.

Paul M Duvall, Steve Matyas, and Andrew Glover. 2007. *Continuous integration: improving software quality and reducing risk*. Pearson Education.

Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Roth. 2015. The reusable holdout: Preserving validity in adaptive data analysis. *Science*, 349(6248):636–638.

Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407.

Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. 2016. Devops. *IEEE Software*, 33(3):94–100.

Wenfei Fan. 2015. Data quality: From theory to practice. *ACM SIGMOD Record*, 44(3):7–18.

William B Frakes and Kyo Kang. 2005. Software reuse research: Status and future. *IEEE Transactions on Software Engineering*, 31(7):529–536.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2001. *The Elements of Statistical Learning*, volume 1. Springer series in statistics New York.

Jozsef Fritz. 1975. Distribution-free exponential error bound for nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 21(5):552–557.

Keinosuke Fukunaga and L Hostetler. 1975. K-nearest-neighbor bayes-risk estimation. *IEEE Transactions on Information Theory*, 21(3):285–293.

Keinosuke Fukunaga and Donald M Hummels. 1987. Bayes error estimation using parzen and k-nn procedures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 634–643.

Keinosuke Fukunaga and David Kessell. 1973. Nonparametric bayes error estimation using unclassified samples. *IEEE Transactions on Information Theory*, 19(4):434–440.

Sébastien Gadat, Thierry Klein, and Clément Marteau. 2016. Classification in general finite dimensional spaces with the k-nearest neighbor rule. *The Annals of Statistics*, 44(3):982–1009.

Ujwal Gadiraju, Ricardo Kawase, Stefan Dietze, and Gianluca Demartini. 2015. Understanding malicious behavior in crowdsourcing platforms: The case of online surveys. In *ACM Conference on Human Factors in Computing Systems*, pages 1631–1640.

Amirata Ghorbani and James Zou. 2019. Data shapley: Equitable valuation of data for machine learning. In *International Conference on Machine Learning*, pages 2242–2251. PMLR.

Antonio Ginart, Martin Zhang, and James Zou. 2022. Mldemon: Deployment monitoring for machine learning systems. In *International Conference on Artificial Intelligence and Statistics*.

Jacob Goldberger, Geoffrey E Hinton, Sam T Roweis, and Russ R Salakhutdinov. 2005. Neighbourhood components analysis. In *Advances in Neural Information Processing Systems*, pages 513–520.

Stefan Grafberger, Shubha Guha, Julia Stoyanovich, and Sebastian Schelter. 2021. Mlinspect: A data distribution debugger for machine learning pipelines. In *International Conference on Management of Data*, pages 2736–2739.

Jack Greenfield and Keith Short. 2003. Software factories: assembling applications with patterns, models, frameworks and tools. In *ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 16–27.

László Györfi. 1981. The rate of convergence of k_n-nn regression estimates and classification rules. *IEEE Transactions on Information Theory*, 27(3):362–364.

László Györfi, Michael Kohler, Adam Krzyżak, and Harro Walk. 2002. *A distribution-free theory of nonparametric regression*, volume 1. Springer.

Steve Hanneke. 2007. A bound on the label complexity of agnostic active learning. In *International Conference on Machine learning*, pages 353–360.

Tatsunori Hashimoto. 2021. Model performance scaling with multiple data sources. In *International Conference on Machine Learning*, pages 4107–4116. PMLR.

Kaiming He, Ross Girshick, and Piotr Dollár. 2019. Rethinking imagenet pre-training. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Minyoung Huh, Pulkit Agrawal, and Alexei A Efros. 2016. What makes ImageNet good for transfer learning? *arXiv preprint arXiv:1608.08614*.

Ihab F Ilyas and Xu Chu. 2019. *Data cleaning*. Morgan & Claypool.

Takashi Ishida, Ikko Yamane, Nontawat Charoenphakdee, Gang Niu, and Masashi Sugiyama. 2022. Is the performance of my deep network too good to be true? a direct approach to estimating the bayes error in binary classification. *arXiv preprint arXiv:2202.00395*.

Kevin Jamieson and Ameet Talwalkar. 2016. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial Intelligence and Statistics*, pages 240–248.

Ruoxi Jia, David Dao, Boxin Wang, Frances Ann Hubis, Nezihe Merve Gurel, Bo Li, Ce Zhang, Costas Spanos, and Dawn Song. 2019a. Efficient task-specific data valuation for nearest neighbor algorithms. In *Proceedings of the VLDB Endowment*, volume 12, pages 1610–1623. VLDB Endowment.

Ruoxi Jia, Xuehui Sun, Jiacen Xu, Ce Zhang, Bo Li, and Dawn Song. 2019b. An empirical and comparative analysis of data valuation with scalable algorithms. *arXiv preprint arXiv:1911.07128*.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*.

Karen Sparck Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*.

Matti Kääriäinen and John Langford. 2005. A comparison of tight generalization error bounds. In *International Conference on Machine learning*, pages 409–416. ACM.

Juraj Kacur, Radoslav Vargic, and Pavol Mulinka. 2011. Speaker identification by k-nearest neighbors: Application of pca and lda prior to knn. In *International Conference on Systems, Signals and Image Processing*, pages 1–4. IEEE.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Bojan Karlaš, Matteo Interlandi, Cedric Renggli, Wentao Wu, Ce Zhang, Deepak Mukunthu Iyappan Babu, Jordan Edwards, Chris Lauren, Andy Xu, and Markus Weimer. 2020a. Building continuous integration services for machine learning. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2407–2415.

Bojan Karlaš, Peng Li, Renzhi Wu, Nezihe Merve Gürel, Xu Chu, Wentao Wu, and Ce Zhang. 2020b. Nearest neighbor classifiers over incomplete information: From certain answers to certain predictions. *arXiv preprint arXiv:2005.05117*.

Bojan Karlaš, Ji Liu, Wentao Wu, and Ce Zhang. 2018. Ease.ml in action: Towards multi-tenant declarative learning services. In *Proceedings of the VLDB Endowment*, volume 11, pages 2054–2057. VLDB Endowment.

Andrej Karpathy. 2017. Software 2.0. `https://medium.com/@karpathy/software-2-0-a64152b37c35`.

Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, pages 1885–1894. PMLR.

Michael Kohler and Adam Krzyzak. 2007. On the rate of convergence of local averaging plug-in classification rules under a margin condition. *IEEE Transactions on Information Theory*, 53(5):1735–1742.

Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. 2019. Large scale learning of general visual representations for transfer. *arXiv preprint arXiv:1912.11370*.

Simon Kornblith, Jonathon Shlens, and Quoc V Le. 2019. Do better Imagenet models transfer better? In *IEEE Conference on Computer Vision and Pattern Recognition*.

Tim Kraska. 2018. Northstar: an interactive data science system. In *Proceedings of the VLDB Endowment*, volume 11, pages 2150–2164. VLDB Endowment.

Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J Franklin, and Ken Goldberg. 2016. ActiveClean: Interactive data cleaning for statistical modeling. In *Proceedings of the VLDB Endowment*, volume 9. VLDB Endowment.

Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. *(Technical Report)*.

Sanjeev R Kulkarni and Steven E Posner. 1995. Rates of convergence of nearest neighbor estimation under arbitrary sampling. *IEEE Transactions on Information Theory*, 41(4):1028–1039.

Arun Kumar, Matthias Boehm, and Jun Yang. 2017. Data management in machine learning: Challenges, techniques, and systems. In *ACM International Conference on Management of Data*, pages 1717–1722.

John Langford. 2005. Tutorial on practical prediction theory for classification. *Journal of machine learning research*, 6(Mar):273–306.

Alvaro Fernando Lara. 2017. Continuous integration for ml projects. https://medium.com/onfido-tech/continuous-integration-for-ml-projects-e11bc1a4d34f.

Alvaro Fernando Lara. 2018. Continuous delivery for ml models. https://medium.com/onfido-tech/continuous-delivery-for-ml-models-c1f9283aa971.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature*, 521(7553):436–444.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816.

Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In *USENIX Symposium on Operating Systems Design and Implementation*, pages 583–598.

Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. 2019. CleanML: A Benchmark for Joint Data Cleaning and Machine Learning [Experiments and Analysis]. *arXiv preprint arXiv:1904.09483*.

Tian Li, Jie Zhong, Ji Liu, Wentao Wu, and Ce Zhang. 2018. Ease.ml: Towards multi-tenant resource sharing for machine learning workloads. In *Proceedings of the VLDB Endowment*, volume 11, pages 607–620. VLDB Endowment.

Yuliang Li, Xiaolan Wang, Zhengjie Miao, and Wang-Chiew Tan. 2021. Data augmentation for ml-driven data preparation and integration. In *Proceedings of the VLDB Endowment*, volume 14, pages 3182–3185. VLDB Endowment.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2018. Progressive neural architecture search. In *European Conference on Computer Vision*.

Ji Liu, Ce Zhang, et al. 2020. Distributed learning systems with first-order methods. *Foundations and Trends® in Databases*, 9(1):1–100.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2015. Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440.

Amiel Meiseles and Lior Rokach. 2020. Source model selection for deep learning in the time series domain. *IEEE Access*.

Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. 2016. MLlib: Machine learning in Apache Spark. *The Journal of Machine Learning Research*, 17(1):1235–1241.

Thomas Mensink, Jasper Uijlings, Alina Kuznetsova, Michael Gygli, and Vittorio Ferrari. 2021. Factors of influence for transfer learning across diverse appearance domains and task types. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Nov(01):1–1.

Supun Nakandala, Yuhao Zhang, and Arun Kumar. 2020. Cerebro: a data system for optimized deep learning model selection. In *Proceedings of the VLDB Endowment*, volume 13, pages 2159–2173. VLDB Endowment.

Preetam Nandy, Kinjal Basu, Shaunak Chatterjee, and Ye Tu. 2020. A/b testing in dense large-scale networks: design and inference. In *Advances in Neural Information Processing Systems*, volume 33, pages 2870–2880.

Cuong V Nguyen, Tal Hassner, Cedric Archambeau, and Matthias Seeger. 2020. Leep: A new measure to evaluate transferability of learned representations. In *International Conference on Machine Learning*.

Maria-Elena Nilsback and Andrew Zisserman. 2008. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729. IEEE.

Laurel Orr, Atindriyo Sanyal, Xiao Ling, Karan Goel, and Megan Leszczynski. 2021. Managing ml pipelines: feature stores and the coming wave of embedding ecosystems. In *Proceedings of the VLDB Endowment*, volume 14, pages 3178–3181. VLDB Endowment.

Sinno Jialin Pan and Qiang Yang. 2009. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*.

Nazila Panahi, Mahrokh G Shayesteh, Sara Mihandoost, and Behrooz Zali Varghahan. 2011. Recognition of different datasets using pca, lda, and various classifiers. In *International Conference on Application of Information and Communication Technologies (AICT)*, pages 1–5. IEEE.

Nicolas Papernot and Patrick McDaniel. 2018. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *arXiv preprint arXiv:1803.04765*.

Thu Pham-Gia, Noyan Turkkan, and Andriëtte Bekker. 2007. Bounds for the bayes error in classification: A bayesian approach using discriminant analysis. *Statistical Methods and Applications*, 16(1):7–26.

Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. 2018. Data lifecycle challenges in production machine learning: a survey. *ACM SIGMOD Record*, 47(2):17–28.

Neoklis Polyzotis, Martin Zinkevich, Sudip Roy, Eric Breck, and Steven Whang. 2019. Data validation for machine learning. In *Machine Learning and Systems*, volume 1.

Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and Sundaraja S Iyengar. 2018. A survey on deep learning: Algorithms, techniques, and applications. *ACM Computing Surveys (CSUR)*, 51(5):1–36.

Joan Puigcerver, Carlos Riquelme, Basil Mustafa, Cedric Renggli, André Susano Pinto, Sylvain Gelly, Daniel Keysers, and Neil Houlsby. 2021. Scalable transfer learning with expert models. In *International Conference on Learning Representations*.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506.

Alexander J Ratner, Stephen H Bach, Henry R Ehrenberg, and Chris Ré. 2017. Snorkel: Fast training set generation for information extraction. In *ACM International Conference on Management of Data*, pages 1683–1686.

Christopher Ré, Feng Niu, Pallavi Gudipati, and Charles Srisuwananukorn. 2020. Overton: A data system for monitoring and improving machine-learned products. In *Conference on Innovative Data Systems Research*. CIDR.

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. 2019. Do imagenet classifiers generalize to imagenet? In *International Conference on Machine Learning*, pages 5389–5400. PMLR.

Cedric Renggli, Frances Ann Hubis, Bojan Karlaš, Kevin Schawinski, Wentao Wu, and Ce Zhang. 2019a. Ease.ml/ci and ease.ml/meter in action: Towards data management for statistical generalization. In *Proceedings of the VLDB Endowment*, volume 12.

Cedric Renggli, Bojan Karlas, Bolin Ding, Feng Liu, Kevin Schawinski, Wentao Wu, and Ce Zhang. 2019b. Continuous integration of machine learning models with ease.ml/ci: Towards a rigorous yet practical treatment. In *SysML Conference*.

Cedric Renggli, André Susano Pinto, Luka Rimanic, Joan Puigcerver, Carlos Riquelme, Ce Zhang, and Mario Lucic. 2022a. Which model to transfer? finding the needle in the growing haystack. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Cedric Renggli, Luka Rimanic, Nezihe Merve Gürel, Bojan Karlaš, Wentao Wu, and Ce Zhang. 2021a. A data quality-driven view of mlops. *IEEE Data Engineering Bulletin*.

Cedric Renggli, Luka Rimanic, Nora Hollenstein, and Ce Zhang. 2021b. Evaluating bayes error estimators on read-world datasets with feebee. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks)*, volume 34.

Cedric Renggli, Luka Rimanic, Luka Kolar, Wentao Wu, and Ce Zhang. 2020. Ease.ml/snoopy in action: Towards automatic feasibility analysis for machine learning application development. In *Proceedings of the VLDB Endowment*, volume 13.

Cedric Renggli, Luka Rimanic, Luka Kolar, Wentao Wu, and Ce Zhang. 2022b. Ease.ml/snoopy: Towards automatic feasibility studies for ml via quantitative understanding of "data quality for ml". *arXiv preprint arXiv:2010.08410*.

Cedric Renggli, Xiaozhe Yao, Luka Kolar, Luka Rimanic, Ana Klimovic, and Ce Zhang. 2022c. Shift: An efficient, flexible search engine for transfer learning. *arXiv preprint arXiv:2204.01457*.

Luka Rimanic, Cedric Renggli, Bo Li, and Ce Zhang. 2020. On convergence of nearest neighbor classifiers over feature transformations. In *Advances in Neural Information Processing Systems*, volume 33.

Jonathan S. Rosenfeld, Amir Rosenfeld, Yonatan Belinkov, and Nir Shavit. 2020. A constructive prediction of the generalization error across scales. In *International Conference on Learning Representations*.

Sebastian Ruder, Matthew E Peters, Swabha Swayamdipta, and Thomas Wolf. 2019. Transfer learning in natural language processing. In *Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*, pages 15–18.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*.

Kevin Scaman and Aladin Virmaux. 2018. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In *Advances in Neural Information Processing Systems*, pages 3839–3848.

Sebastian Schelter. 2020. Amnesia – a selection of machine learning models that can forget user data very fast. In *Conference on Innovative Data Systems Research*.

Sebastian Schelter, Felix Biessmann, Tim Januschowski, David Salinas, Stephan Seufert, and Gyuri Szarvas. 2018a. On challenges in machine learning model management. *Data Engineering*, page 5.

Sebastian Schelter, Stefan Grafberger, Shubha Guha, Olivier Sprangers, Bojan Karlaš, and Ce Zhang. 2022. Screening native ml pipelines with "arguseyes". In *Conference on Innovative Data Systems Research*. CIDR.

Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Biessmann, and Andreas Grafberger. 2018b. Automating large-scale data quality verification. In *Proceedings of the VLDB Endowment*, volume 11, pages 1781–1794. VLDB Endowment.

S. Y. Sekeh, B. L. Oselio, and A. O. Hero. 2020. Learning to bound the multi-class Bayes error. *IEEE Transactions on Signal Processing*.

Shai Shalev-Shwartz and Shai Ben-David. 2014. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge university press.

Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. 2016. Understanding and improving convolutional neural networks via concatenated rectified linear units. In *International Conference on Machine Learning*, pages 2217–2225. PMLR.

Robert R Snapp, Demetri Psaltis, and Santosh S Venkatesh. 1991. Asymptotic slowing down of the nearest-neighbor classifier. In *Advances in Neural Information Processing Systems*, pages 932–938.

Robert R Snapp and Tong Xu. 1996. Estimating the bayes risk from sample data. In *Advances in Neural Information Processing Systems*, pages 232–238. MORGAN KAUFMANN PUBLISHERS.

Robert Stojnic. 2018a. Continuous integration for machine learning. https://medium.com/@rstojnic/continuous-integration-for-machine-learning-6893aa867002.

Robert Stojnic. 2018b. Continuous integration for machine learning. https://www.reddit.com/r/MachineLearning/comments/8bq5la/d_continuous_integration_for_machine_learning/.

Diane M Strong, Yang W Lee, and Richard Y Wang. 1997. Data quality in context. *Communications of the ACM*, 40(5).

David Q Sun, Hadas Kotek, Christopher Klein, Mayank Gupta, William Li, and Jason D Williams. 2020. Improving human-labeled data through dynamic automatic conflict resolution. *arXiv preprint arXiv:2012.04169*.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the Inception architecture for computer vision. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. 2018. A survey on deep transfer learning. In *International Conference on Artificial Neural Networks*.

Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR.

Bruce Thompson. 2005. Canonical correlation analysis. *Encyclopedia of statistics in behavioral science*.

Anh T Tran, Cuong V Nguyen, and Tal Hassner. 2019. Transferability and hardness of supervised classification tasks. In *IEEE International Conference on Computer Vision*, pages 1395–1405.

Dat Tran. 2017. Continuous integration for data science. http://engineering.pivotal.io/post/continuous-integration-for-data-science/.

Bruno Trstenjak, Sasa Mikac, and Dzenana Donko. 2014. Knn with tf-idf based framework for text categorization. *Procedia Engineering*, 69:1356–1364.

Alexander B Tsybakov. 2004. Optimal aggregation of classifiers in statistical learning. *The Annals of Statistics*, 32(1):135–166.

Hans Van Vliet, Hans Van Vliet, and JC Van Vliet. 2008. *Software engineering: principles and practice*, volume 13. John Wiley & Sons.

Vladimir N Vapnik and A Ya Chervonenkis. 2015. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of Complexity*. Springer.

Manasi Vartak, Harihar Subramanyam, Wei-En Lee, Srinidhi Viswanathan, Saadiyah Husnoo, Samuel Madden, and Matei Zaharia. 2016. Modeldb: a system for machine learning model management. In *Workshop on Human-In-the-Loop Data Analytics*, pages 1–3.

Terry J. Wagner. 1971. Convergence of the nearest neighbor rule. *IEEE Transactions on Information Theory*, 17(5):566–571.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355.

Richard Y Wang and Diane M Strong. 1996. Beyond accuracy: What data quality means to data consumers. *Journal of management information systems*, 12(4):5–33.

Zhiguo Wang, Wael Hamza, and Linfeng Song. 2017. *k*-nearest neighbor augmented neural networks for text classification. *arXiv preprint arXiv:1708.07863*.

Jiaheng Wei, Zhaowei Zhu, Hao Cheng, Tongliang Liu, Gang Niu, and Yang Liu. 2022. Learning with noisy labels revisited: A study using real-world human annotations. In *International Conference on Learning Representations*.

Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. 2016. A survey of transfer learning. *Journal of Big data*.

Zhirong Wu, Alexei A Efros, and Stella X Yu. 2018. Improving generalization via scalable neighborhood component analysis. In *European Conference on Computer Vision*, pages 685–701.

Doris Xin, Hui Miao, Aditya Parameswaran, and Neoklis Polyzotis. 2021. Production machine learning pipelines: Empirical analysis and optimization opportunities. In *International Conference on Management of Data*, pages 2639–2652.

Xi Yan, David Acuna, and Sanja Fidler. 2020. Neural data server: A large-scale search engine for transfer learning data. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3893–3902.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems*, pages 5754–5764.

Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, et al. 2018. Accelerating the machine learning lifecycle with mlflow. *IEEE Data Engineering Bulletin*, 41(4):39–45.

Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. 2018. Taskonomy: Disentangling task transfer learning. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Xiaohua Zhai, Joan Puigcerver, Alexander Kolesnikov, Pierre Ruyssen, Carlos Riquelme, Mario Lucic, Josip Djolonga, Andre Susano Pinto, Maxim Neumann, Alexey Dosovitskiy, et al. 2019a. A large-scale study of representation learning with the visual task adaptation benchmark. *arXiv preprint arXiv:1910.04867*.

Xiaohua Zhai, Joan Puigcerver, Alexander Kolesnikov, Pierre Ruyssen, Carlos Riquelme, Mario Lucic, Josip Djolonga, Andre Susano Pinto, Maxim Neumann, Alexey Dosovitskiy, et al. 2019b. The visual task adaptation benchmark. *arXiv preprint arXiv:1910.04867*.

Ce Zhang, Christopher Ré, Michael Cafarella, Christopher De Sa, Alex Ratner, Jaeho Shin, Feiran Wang, and Sen Wu. 2017. Deepdive: Declarative knowledge base construction. *Communications of the ACM*, 60(5):93–102.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*.