Ease.ml/snoopy in Action: Towards Automatic Feasibility Analysis for Machine Learning Application Development*

Cedric Renggli*,† Luka Rimanic*,† Luka Kolar† Wentao Wu‡ Ce Zhang†
† ETH Zurich ‡ Microsoft Research, Redmond
{cedric.renggli, luka.rimanic, ce.zhang}@inf.ethz.ch
kolarl@student.ethz.ch, wentao.wu@microsoft.com

ABSTRACT

We demonstrate `ease.ml/snoopy`, a data analytics system that performs *feasibility analysis* for machine learning (ML) applications *before* they are developed. Given a performance target of an ML application (e.g., accuracy above 0.95), `ease.ml/snoopy` provides a decisive answer to ML developers regarding whether the target is achievable or not. We formulate the feasibility analysis problem as an instance of Bayes error estimation. That is, for a data (distribution) on which the ML application should be performed, `ease.ml/snoopy` provides an estimate of the Bayes error – the *minimum error rate* that can be achieved by *any* classifier. It is well-known that estimating the Bayes error is a notoriously hard task. In `ease.ml/snoopy` we explore and employ estimators based on the combination of (1) nearest neighbor (NN) classifiers and (2) pre-trained feature transformations. To the best of our knowledge, this is the first work on Bayes error estimation that combines (1) and (2). In today's cost-driven business world, feasibility of an ML project is an ideal piece of information for ML application developers – `ease.ml/snoopy` plays the role of a reliable "*consultant*."

PVLDB Reference Format:
Cedric Renggli et al.. Ease.ml/snoopy in Action: Towards Automatic Feasibility Analysis for Machine Learning Application Development. *PVLDB*, 13(12): 2837-2840, 2020.
DOI: https://doi.org/10.14778/3415478.3415488

1. INTRODUCTION

Development of machine learning (ML) applications is similar to development of regular software: It is an engineering process that requires principled methodology to manage its lifecycle and control its quality. Unfortunately, unlike development of regular software, which is typically guided by modern software engineering principles that have been developed and refined for decades, development of ML applications currently lacks such guidelines. Plenty of work has been devoted to building *efficient* systems [12, 15], or *effective* tools [1, 9, 11, 13] to improve productivity of developers. However, the *control* over ML application development itself is still at its beginnings [7, 17, 18].

*The first two authors contributed equally to this work.

Proceedings of the VLDB Endowment, Vol. 13, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3415478.3415488

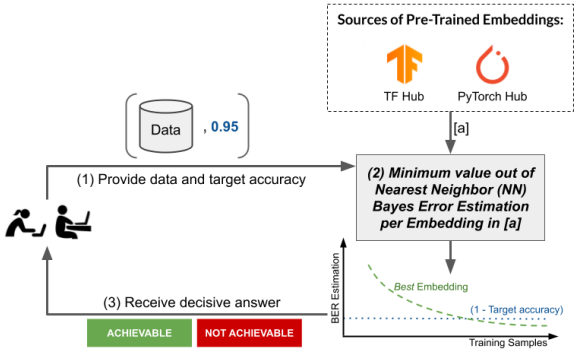


Figure 1: User interaction with `ease.ml/snoopy`

In this paper, we take a step further in narrowing down this gap by bringing classic software engineering perspectives into the development of ML applications. We focus on *feasibility analysis*, a crucial step in software engineering that studies the possibility of building a software system that meets user's requirements. We develop `ease.ml/snoopy`, a system that evaluates *whether the goal of a given ML application can be achieved*. `ease.ml/snoopy` offers such a service in an automated manner *without* training any model — it *snoops* at the data solely and therefore runs at a low cost in terms of computation and time.

More specifically, `ease.ml/snoopy` provides a decisive answer to whether there exists a model that, on a given dataset, achieves a better-than-required target *classification accuracy*. We formulate this feasibility analysis problem as an instance of estimating the *Bayes error*. It is well known that the Bayes error indicates the error rate of the *Bayes optimal classifier*, which is the *minimum* error rate achievable by *any* classifier. Unfortunately, since the underlying probability distribution is usually unknown for real-world datasets, one can only estimate the Bayes error. Estimating it accurately is a notoriously hard task despite decades of research [3, 5, 21, 22].

(Nearest Neighbor Estimator) The nearest neighbor (NN) classifier has been extensively studied in the literature as an estimator for the Bayes error, due to its analytic tractability equipped with nearly optimal performance when many samples are available. Roughly speaking, the NN algorithm classifies a feature vector x by consulting a sample dataset D of n data points, where it first finds the *nearest* neighbors of x (among the n points in D), and then simply assigns x to the label represented by the nearest neighbor. In `ease.ml/snoopy`, we adapt the NN estimator originally developed by Cover and Hart [3], which has been further analyzed by Snapp et al. [21]. The later work showed that their estimator suffers from the *curse of dimensionality*, a well-known difficulty in the ML world that happens when the inputs are high-dimensional. To surpass this issue, we piggyback the NN classier with a set of pre-trained em-

beddings and other feature transformations, in order to speed up the convergence of the NN classifier. This allows us to get a tighter estimate of the Bayes error.

(Evaluation of the Estimator) Evaluating our Bayes error estimator on realistic datasets is not a trivial task — most, if not all, previous work on Bayes error estimation works around this problem by using synthetic datasets with known Bayes errors [3, 5, 22]. To overcome this restriction, we have developed a novel methodology for evaluating our Bayes error estimator (Section 2.2).

(Demonstration Scenarios) We will focus on the following scenarios in our demonstration:

1. A demonstration of how manual feasibility study on a variety of real-world dataset would look like, in order to motivate the need for *automatic feasibility analysis*;
2. A demonstration on why existing AutoML systems are not well suited to perform feasibility study;
3. A demonstration of the functionality of `ease.ml/snoopy` and its evaluation methodology on the real world datasets;
4. Interactions with the audience to *test* the system with user's data and non-public pre-trained embeddings.

2. SYSTEM OVERVIEW

Building `ease.ml/snoopy`, and in particular its evaluation component, is a challenging task. In this section we give an overview of the technicalities involved in building such a novel system. Figure 2 illustrates the interface for `ease.ml/snoopy`. The user interactions for running the evaluation using `ease.ml/snoopy` are minimal, in the following sense: After uploading the dataset (features and labels for the test and training samples), one simply has to specify the desired target accuracy, and, if available, the best state-of-the-art. To speed up the evaluation process, one can manually filter the available feature transformations.

2.1 Functionality

`ease.ml/snoopy` performs a feasibility analysis for ML applications automatically. A user of `ease.ml/snoopy` only needs to provide the system with the dataset D that she wants to use for her ML application, as well as its target performance (e.g., the accuracy of a classification task). The system will decide whether this target is meaningful, i.e., whether the *best* classifier trained using D can meet the target. As we have discussed so far, `ease.ml/snoopy` achieves this goal by providing an estimated Bayes error of the given ML application on D.

Formally, let $\mathcal{X}$ be the feature space and $\mathcal{Y}$ be the label space, with $C = |\mathcal{Y}|$. Let X, Y be random variables on Ω taking values in $\mathcal{X}$ and $\mathcal{Y}$. We denote their joint distributions by $\mathcal{D} = p(X, Y)$. The *Bayes optimal classifier* is the classifier that achieves the lowest error rate among all possible classifiers from $\mathcal{X}$ to $\mathcal{Y}$. Its error rate $R^*_{X,Y}$ is then the *Bayes error rate*, and can be calculated as

$$R^*_{X,Y} = \mathbb{E}_X \left[1 - \max_{Y \in \mathcal{Y}} p(Y|X) \right].$$

The NN classifier exhibits nice analytic properties in relation to the Bayes error. Let h_n be an NN classifier with n data points in a set D sampled *i.i.d* from a fixed distribution $\mathcal{D}$, in short $D \sim \mathcal{D}$. We use the following simplified notation hereafter:

- $L_n = L_{\mathcal{D}}(h_n)$: The error of the NN classifier with n sample points in $D \sim \mathcal{D}$;
- $L_\infty = L_{\mathcal{D}}(h_\infty)$: The error of the NN classifier with *infinite* sample points in $D \sim \mathcal{D}$.

Figure 2: User interface for `ease.ml/snoopy`

The following observation by Cover and Hart [3] (under very mild assumptions on continuity of probability distributions) is the crucial motivation behind our estimator:

THEOREM 2.1. *For any given $D \sim \mathcal{D}$ with n points,*

$$\lim_{n \to \infty} L_n = L_\infty.$$

Moreover, L_∞ is bounded by

$$R^*_{X,Y} \leq L_\infty \leq R^*_{X,Y} \cdot \left(2 - \frac{C}{C-1} R^*_{X,Y} \right) \leq 2R^*_{X,Y}. \quad (2.1)$$

Intuitively, (1) the error rate of the NN classifier *converges* as we increase the sample size; and (2) the Bayes error can be *bounded from below* by *half* of the error of the NN classifier error with infinite samples. Of course, accurate computation of L_∞ is unfeasible in practice, since it requires infinite number of samples from $\mathcal{D}$. Nonetheless, the convergence property of the NN classifier ensures that we can estimate L_∞ reasonably well in the large-sample regime. The accuracy of the estimator we use in `ease.ml/snoopy` has been empirically validated in the literature on a range of synthetic low-dimensional datasets [3, 21]. A major drawback of the NN classifier lies in its slow convergence when dealing with high dimensional input [21]. To bypass this in `ease.ml/snoopy`, we run the classifier not directly on the raw input features, but on features produced by a deterministic feature transformation f. We remark that such a transformation cannot reduce the Bayes error, and hence will never give a too optimistic Bayes error estimator. Making use of this property, together with the previous theorem, we define our estimator as follows:

$$\widehat{R}_{X,Y} := \min_{f \in \mathcal{F}} \frac{L_{f,n}}{1 + \sqrt{1 - \frac{C}{C-1} L_{f,n}}},$$

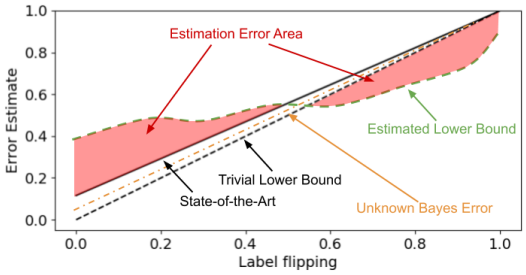


Figure 3: Evaluation Methodology

where $\mathcal{F}$ is a set of feature transformations, and $L_{f,n}$ is the error of the NN classifier with n samples after applying the transformation $f \in \mathcal{F}$ to all the samples. Notice that taking the mininum of $L_{f,n}/2$ would give a looser bound.

(Pre-Trained Feature Transformations) In order to have a good estimator using the previously described approach, we need to build a set of feature transformations $\mathcal{F}$ that speeds up the convergence of the NN classifier accuracy, whilst only slightly increasing the underlying Bayes error. In `ease.ml/snoopy` we follow a common approach in transfer learning [16], by making use of *pretrained embeddings* available in public repositories such as Tensorflow Hub[1] or PyTorch-Hub[2]. In order to unify the use of embeddings from both the deep learning frameworks and additional arbitrary feature transformations, we specify a simple interface using Numpy arrays:

```
new_features, new_labels <- apply(features, labels)
```

2.2 Evaluation Methodology

The goal of the evaluation is to assess the quality of our estimator on realistic datasets, without access to prior knowledge of the true Bayes error. We start by introducing some simple notation in order to describe our methodology.

Consider a dataset D with n i.i.d. samples from $\mathcal{D}$, for which we *know the state-of-the-art (SOTA) classification performance* s_D (see Table 1). Note that the latter is an upper bound of the Bayes error $R^*_{X,Y}$. Now assume that for a proportion of ρ samples we randomly change the label. Clearly, this would increase the Bayes error, and the following result quantifies it.

THEOREM 2.2. *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 $R^*_{X,Y_\rho} = R^*_{X,Y} + \rho(1 - 1/C - R^*_{X,Y})$.*

The random variable Y_ρ takes values in $\mathcal{Y}$, while ρ corresponds to the probability of randomly assigning the label from the original label to a random value in $\mathcal{Y}$. Theorem 2.2 implies that $1 - 1/C \geq R^*_{X,Y_\rho} \geq R^*_{X,Y}$, with equality when $\rho = 1$ or $\rho = 0$.

As a direct consequence of Theorem 2.2, using the SOTA as an upper bound for $R^*_{X,Y}$, and $R^*_{X,Y} \geq 0$ as the trivial lower bound, we can define the valid bounds on $R^*_{X,Y_\rho} \in [\ell_D(\rho), u_D(\rho)]$, with $u_D(\rho) = s_D + \rho(1 - 1/C - s_D)$ and $\ell_D(\rho) = \rho(1 - 1/C)$. For a fixed method m, we can estimate the lower bound $\ell_{D_\rho,m}(\rho)$ on a manipulated dataset D_ρ obtained by taking $\rho \cdot n$ samples out of D, and randomly changing their labels, whilst keeping the other $(1-\rho) \cdot n$ samples intact. To define the error of a given method m on the modified dataset D_ρ, we can estimate two areas: (i) the area where m is clearly underestimating the Bayes error lower bound, and (ii) the area where m is overestimating it. We illustrate these quantities in Figure 3. Notice that for every method m it holds that the area is equal to zero if and only if for all $\rho \in [0, 1]$, the lower-bound estimate lies between $u_D(\rho)$ and $\ell_D(\rho)$, in which case the method m is an *optimal* lower-bound estimate.

[1] `https://tfhub.dev/`

[2] `https://pytorch.org/hub`

Table 1: Datasets and the performance of SOTA classifiers.

NAME	Classes	TRAIN / TEST	SOTA %
MNIST	10	60K / 10K	0.17 [2]
CIFAR10	10	50K / 10K	0.7 [10]
CIFAR100	100	50K / 10K	6.4 [10]
IMDB	2	25K / 25K	3.79 [24]
CoLa	2	8.5K / 1K	22.8 [23]

3. DEMONSTRATION SCENARIOS

We present the details of the scenarios that we plan to demonstrate with `ease.ml/snoopy`. We will use two data modalities that are ubiquotous in modern machine learning. The first group consists of visual classification tasks, including CIFAR10, CIFAR100, and MNIST. The second group consists of standard text classification tasks, where we focus on IMDB and CoLa. Table 1 presents the details. The chosen tasks are well studied with many years of research. Hence, it is possible that its state-of-the-art models achieve accuracies that are *close* to the true unknown Bayes error. We note 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 term-frequency/inverse-document-frequency weighting [8].

3.1 Scenario 1: Missing Automation

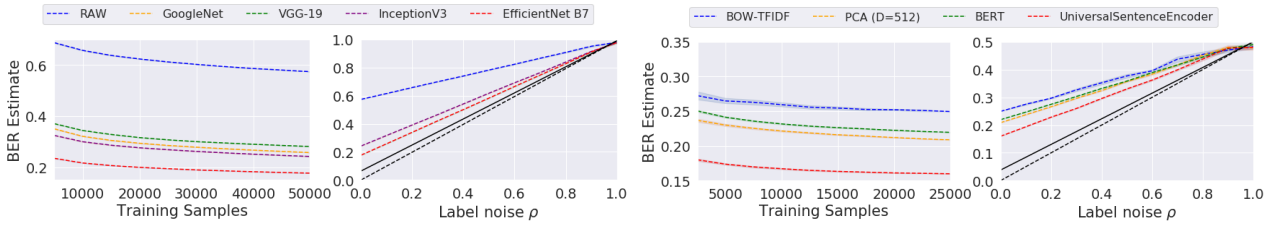
We will start by highlighting the importance of *automation* in feasibility study. We will increasingly add small fractions of label noise to the datasets. By making use of Theorem 2.2 and the SOTA values, we are able to tell in what range the Bayes error falls. Especially in the regime of little label noise, we will showcase that it is very hard and time-consuming to distinguish different fractions of noisy labels, by simply looking at handpicked samples or conducting manual statistical analysis without any structured approach. This will hopefully convince the audience of the motivation of automatic feasibility analysis.

3.2 Scenario 2: AutoML Systems

We will then proceed by making use of available AutoML systems to automatically train a model on a given dataset, and emphasize why such systems are not well suited to perform a thorough feasibility study. Following the same approach described in Scenario 1, we will show that the idea of running multiple instances with different manipulated datasets (corrupted with a known fraction of label noise) on AutoML systems has two major drawbacks. First, it may cost a lot of money and time for an AutoML system to output the best model and its accuracy. Second, if the model accuracy is below the desired target, the user either needs to *guess* whether there exists a better model not covered by the search space of the AutoML system, or trust the model without knowing the Bayes error. We hope that this will convince the audience that AutoML systems are not sufficient for automatic feasibility study, as they do not produce a lower bound on the Bayes error.

3.3 Scenario 3: Ease.ml/snoopy

We then showcase the functionalities of `ease.ml/snoopy` by making use of all the datasets in Table 1 with their respective SOTA values and a set of feature transformations. When selecting potential pre-trained embeddings, we look at multiple available public



(a) CIFAR100 (b) IMDB

Figure 4: (a) NN estimation on CIFAR100. (b) NN estimation on IMDB.

sources, such as TensorFlow Hub or PyTorch-Hub, as well as simple transformations such as PCA and NCA [6], with various target dimensions. Notice that particularly in the image domain, pretrained embeddings often assume a fixed-sized resolution, which might differ from the target image size. The adjustment in those cases is done using default resizing methods of either TensorFlow or PyTorch. Finally, including the *identity* transformation in our set of feature transformations allows us to quantify the difference between raw data representations and other transformations. In Figure 4 we highlight the difference in terms of convergence rates, for a fixed set of transformations and one dataset from each modality, with respect to our evaluation methodology.

3.4 Scenario 4: Interaction with Audience

In this final part, we would like to invite our audience to interact with the system, possibly using their own data and optionally known SOTA values. The input format of the data needs to be standardized to the previously described Numpy arrays. In order to support our audience, we plan to provide simple tools to either check the input format, or convert other formats such as Tensorflow Dataset[3], images stored in folders following the Torchvision conventions[4], or text files containing one sample per line. Finally, we plan to extend the list of feature transformations by specifying REST URLs following simple specifications given in advance.

4. RELATED WORK

We focused on a simple nearest neighbor classifier in order to estimate the Bayes error. There are various different approaches though. Given any density estimator based on finite data, one can explore two alternatives: (1) estimating the class posterior density for every single class [4], or (2) estimating the class prior and likelihood density for every class [5, 19], followed by deriving the maximum class posterior by applying the Bayes Theorem. Typically, these methods are not *practical* in the sense that there is no strategy to tune the hyper-parameters. More recent works aim to estimate the class posterior divergence in order to estimate the Bayes error [14, 20]. These methods are either only defined for binary classification problems, or are computationally not tractable for large sample size as they need to compute the minimum spanning tree (MST) over the fully connected graph of samples.

5. CONCLUSION

We have demonstrated `ease.ml/snoopy`, a new genre of data analytics system that provides automatic, a priori feasibility analysis for ML applications. `ease.ml/snoopy` helps ML developers understand how realistic their desired performance goals are, by providing an estimate over the Bayes error of ML applications on available datasets. `ease.ml/snoopy` fits into the broad view of controlling and managing lifecycles of ML application development, and we hope that our work provides interesting scenarios that could inspire future research in this fertile ground.

Acknowledgements CZ and the DS3Lab gratefully acknowledge the support from Swiss Data Science Center, the Swiss National Science Foundation (Project Number 200021_184628), Alibaba, eBay, Google Focused Research Awards, Oracle Labs, Zurich Insurance, Chinese Scholarship Council, and the Department of Computer Science at ETH Zurich.

[3] `https://www.tensorflow.org/datasets/`

[4] `https://pytorch.org/docs/stable/torchvision/datasets.html`

6. REFERENCES

[1] T. Bakogiannis, I. Giannakopoulos, D. Tsoumakos, and N. Koziris. Apollo: A dataset profiling and operator modeling system. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1869–1872, 2019.

[2] A. Byerly, T. Kalganova, and I. Dear. A branching and merging convolutional network with homogeneous filter capsules. *arXiv*, 2020.

[3] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[4] R. O. Duda et al. *Pattern classification*. John Wiley & Sons, 2012.

[5] K. Fukunaga and D. M. Hummels. Bayes error estimation using parzen and k-nn procedures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (5):634–643, 1987.

[6] J. Goldberger, G. E. Hinton, S. T. Roweis, and R. R. Salakhutdinov. Neighbourhood components analysis. In *NIPS*, pages 513–520, 2005.

[7] F. A. Hubis, W. Wu, and C. Zhang. Ease.ml/meter: Quantitative overfitting management for human-in-the-loop ml application development. *arXiv*, 2019.

[8] K. S. Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 1972.

[9] B. Karlaš, J. Liu, W. Wu, and C. Zhang. Ease.ml in action: Towards multi-tenant declarative learning services. *PVLDB*, 11(12):2054–2057, 2018.

[10] A. Kolesnikov et al. Large scale learning of general visual representations for transfer. *arXiv*, 2019.

[11] T. Kraska. Northstar: An interactive data science system. *PVLDB*, 11(12):2150–2164, 2018.

[12] M. Li et al. Scaling distributed machine learning with the parameter server. In *OSDI 14*, pages 583–598, 2014.

[13] T. Li, J. Zhong, J. Liu, W. Wu, and C. Zhang. Ease.ml: Towards multi-tenant resource sharing for machine learning workloads. *PVLDB*, 11(5):607–620, 2018.

[14] J. Lin. Divergence measures based on the Shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, 1991.

[15] X. Meng et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, 2016.

[16] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[17] C. Renggli et al. Continuous integration of machine learning models with ease.ml/ci: Towards a rigorous yet practical treatment. In *SysML*, 2019.

[18] C. Renggli et al. Ease.ml/ci and ease.ml/meter in action: towards data management for statistical generalization. *PVLDB*, 12(12):1962–1965, 2019.

[19] M. Rosenblatt. Remarks on some nonparametric estimates of a density function. *Annals of Mathematical Statistics*, 27(3):832–837, 1956.

[20] S. Y. Sekeh, B. Oselio, and A. O. Hero. Learning to bound the multi-class Bayes error. *arXiv*, 2018.

[21] R. R. Snapp et al. Asymptotic slowing down of the nearest-neighbor classifier. In *NIPS*, pages 932–938, 1991.

[22] R. R. Snapp and T. Xu. Estimating the bayes risk from sample data. In *NIPS*, pages 232–238, 1996.

[23] A. Warstadt et al. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641, 2019.

[24] Z. Yang et al. Xlnet: Generalized autoregressive pretraining for language understanding. In *NIPS*, pages 5754–5764, 2019.