The Thesis Committee for Rahul Nandakumar
certifies that this is the approved version of the following thesis:

Feature Engineering for Supervised Learning

SUPERVISING COMMITTEE:

Raghu Bollapragada, Supervisor

Deepayan Chakrabarti, Co-supervisor

Feature Engineering for Supervised Learning

by

Rahul Nandakumar

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science

The University of Texas at Austin

August 2024

Dedication

Dedicated to my mother and father.

Epigraph

What starts here changes the world.

—University of Texas at Austin

Acknowledgments

I would like to express my deepest gratitude to all those who have supported and guided me throughout the process of writing this thesis.

First and foremost, I wish to thank my advisor, Dr. Deepayan Chakrabarti, for his unwavering support, expert guidance, and invaluable insights. His patience, encouragement, and feedback were instrumental in the successful completion of this research. I am profoundly grateful for his mentorship and dedication to my academic growth.

I would also like to extend my gratitude to Dr. Raghu Bollapragada, for his thoughtful comments, constructive criticisms, and suggestions, which significantly enhanced the quality of this work.

I am thankful to University of Texas at Austin and the Operations Research and Industrial Engineering program for providing the resources and a conducive environment for my research. I would like to acknowledge the financial support received from a McCombs Excellence Grant (2024) which made this research possible.

A special thanks to my colleagues and friends in the ORIE department for their camaraderie, moral support, and for providing a stimulating and enjoyable environment in which to learn and grow.

Abstract

Feature Engineering for Supervised Learning

Rahul Nandakumar, M.S.E.
The University of Texas at Austin, 2024

SUPERVISORS: Raghu Bollapragada, Deepayan Chakrabarti

The main objective of this study is to investigate the possibility of generating extremely effective features when dealing with multicluster, multidimensional datasets with low information features. We present the design and development of a general-purpose, interpretable, and rectifiable continual learning algorithm. We introduce Patch, an algorithm that identifies ideal binary features, termed "one-way classifiers". A one-way classifier predicts the class accurately when the feature is active and remains non-specific when the feature is inactive. Patch finds these classifiers in a parallelizable manner, optimizing for reliability and preventing overfitting. This concept is extended to residuals through our PatchAugment algorithm, which improves the performance of any baseline model by focusing on poorly predicted data points.

Our formalization of manufactured features as one-way classifiers distinguishes our work from traditional embedding-based feature construction methods. Unlike embeddings that require all components to work in unison, each feature created by Patch is independently informative for a subset of the data, making it highly interpretable and verifiable.

The contributions of this thesis are twofold: first, we introduce a robust algorithm for automatic feature construction; second, we develop a novel continual

learning approach that leverages these features to enhance model performance. Experiments on multiple real-world datasets demonstrate significant improvements in AUC, validating the efficacy of our methods.

The structure of this thesis is as follows: Chapter 2 surveys related work. Chapter 3 formulates the feature construction problem and presents our model. Chapter 4 details our algorithms for feature construction and continual learning. Chapter 5 provides empirical validation, and Chapter 6 concludes the thesis.

Table of Contents

List of Tables 10
List of Figures 11
Chapter 1: Introduction 12
1.1 Contributions and Organization 13
Chapter 2: Literature Review 15
Chapter 3: Problem Formulation 18
3.1 Formal Problem Statement 18
3.2 Proposed Method 22
Chapter 4: Algorithm 26
Chapter 5: Experiments 32
Chapter 6: Conclusion 34
6.1 Key Contributions 34
6.2 Implications for Future Research 35
Appendix A: Detailed Experimental Results 36
Glossary 41
Works Cited 42
Vita 44

List of Tables

5.1 *Dataset characteristics.* 33

A.1 PatchAugment *on CIFAR-10:* The table shows the AUC scores obtained by Gradient Boosting and a Linear SVM, with and without PatchAugment. 36

A.2 PatchAugment *on Street View House Numbers:* The table shows the AUC scores obtained by Gradient Boosting and a Linear SVM, with and without PatchAugment. 36

A.3 PatchAugment *on CelebA:* The table shows the AUC scores obtained by Gradient Boosting and a Linear SVM, with and without PatchAugment. 39

A.4 PatchAugment *on Protein:* The table shows the AUC scores obtained by Gradient Boosting and a Linear SVM, with and without PatchAugment. 39

List of Figures

A.1 PatchAugment*'s accuracy versus number of manufactured features on CIFAR-10:* Each plot shows the AUC for a binary one-versus-the-rest classifier, using PatchAugment to augment Gradient Boosting and a linear SVM. PatchAugment yields a consistent and significant improvement in AUC. 37

A.2 PatchAugment*'s accuracy versus number of manufactured features on Street View House Numbers:* Each plot shows the AUC for a binary one-versus-the-rest classifier, using PatchAugment to augment Gradient Boosting and a linear SVM. PatchAugment yields a consistent and significant improvement in AUC. 38

A.3 PatchAugment*'s accuracy versus number of manufactured features on CelebA For Class Smiling and Class Pointy Nose:* Each plot shows the AUC for a binary one-versus-the-rest classifier, using PatchAugment to augment Gradient Boosting and a linear SVM. PatchAugment yields a consistent and significant improvement in AUC. 39

A.4 PatchAugment*'s accuracy versus number of manufactured features on Protein For Class 0,1,2:* Each plot shows the AUC for a binary one-versus-the-rest classifier, using PatchAugment to augment Gradient Boosting and a linear SVM. PatchAugment yields a consistent and significant improvement in AUC. 40

Chapter 1: Introduction

AI applications must continuously adapt to changing data distributions. The simplest method for this involves periodically refreshing the entire pipeline, not just by retraining machine learning (ML) models on new data, but also by updating the operating points of these models, integrating ML scores with business logic, identifying edge cases, and fine-tuning the guardrails accordingly. This comprehensive process is labor-intensive and expensive, making frequent updates impractical. This challenge is where continual learning steps in, aiming to update models seamlessly as new data arrives.

One straightforward approach to continual learning is fine-tuning existing models with a few backpropagation steps. Recent research has focused on speeding up this process and reducing the number of parameter updates. However, this method is limited to models trained via backpropagation, thus excluding many widely used models like gradient-boosted decision trees. Additionally, fine-tuning can introduce unpredictable and opaque changes in the model's decision boundary, potentially creating new edge cases and increasing business risks. If data corruption is detected post-fine-tuning, reversing its effects can be challenging, emphasizing the need for a more robust solution:

Design a general-purpose, interpretable, and rectifiable continual learning algorithm.

Our approach involves continuously modeling the residuals of a periodically trained baseline model. This flexibility allows for the use of any baseline model, including decision trees and SVMs, as demonstrated empirically in the Experiments Chapter. As new data arrives, we compute the residuals—the discrepancies between actual class labels and predicted probabilities. These residuals are then modeled using a set of *manufactured features*, each representing an interpretable function of the raw features. Our algorithm constructs these features in parallel, ensuring that each

feature's quality can be independently verified. This independence allows for reassessment and elimination of features in the event of data corruption, thus maintaining the resilience of our residuals model.

The feature manufacturing algorithm is central to our method. We identify the ideal binary feature as a "one-way classifier." A one-way classifier reliably predicts the class of a data point when the feature is active but remains uncertain when it is inactive. Essentially, a manufactured binary feature delineates regions of the raw feature space where data points predominantly belong to a single class. Our algorithm, termed PATCH, efficiently discovers such one-way classifiers through a parallelizable approach. This concept is further extended to residuals in an algorithm we call PATCHAUGMENT.

Our approach to feature manufacturing as one-way classifiers marks a significant departure from traditional methods. Previous approaches often view manufactured features as complex combinations of raw features, where each component's utility hinges on the entire feature set. For instance, components of embeddings in neural networks are intricate combinations of raw features and are useful only when aggregated. In contrast, each feature produced by PATCH functions as an independent one-way classifier, verifiable in isolation. Each is specifically designed to be highly informative for certain data subsets. Thus, while existing methods create features that are broadly informative, PATCH generates features that provide high informativeness for specific data subsets.

1.1 Contributions and Organization

Our research contributes significantly to both automatic feature construction and continual learning.

Formulation. We demonstrate that feature construction can be framed as a one-way classification problem. Leveraging well-established classification techniques allows us

to apply classification tools to feature construction. Our formulation ensures that each feature is interpretable as a simple classifier, with its accuracy independently verifiable.

Algorithm for Feature Manufacturing. We introduce PATCH, an algorithm designed to identify one-way classifiers relevant to small data subsets. PATCH employs robust optimization to ensure reliable classifiers that mitigate overfitting. The algorithm operates through a series of parallelizable iterations, enabling the simultaneous generation of numerous features.

Model for Manufactured Features. Given that real-world datasets lack ground truth for manufactured features, we develop a model where the ideal combinations of raw features are known. Experiments on data simulated with this model confirm that features generated by PATCH effectively capture the ideal feature combinations across various scenarios.

Algorithm for Continual Learning. We present PATCHAUGMENT, an algorithm for continual learning compatible with any baseline model, including boosted trees. PATCHAUGMENT identifies features that improve classification of residuals—data points where the baseline model underperforms. Empirical results on 3 real-world datasets show up to 9% improvement in AUC compared to boosted trees trained on the entire dataset. This demonstrates that PATCHAUGMENT enhances model performance even when the full dataset is available upfront.

In summary, our approach offers a pioneering solution to the complexities of continual learning, ensuring robustness, interpretability, and adaptability in the face of ever-changing data landscapes. By focusing on one-way classifiers and parallelizable feature construction, we pave the way for more resilient and effective AI systems.

Chapter 2: Literature Review

Feature construction is a crucial step in machine learning where we either generate an exhaustive set of new features, or modify existing ones by iteratively adding new features that would improve the prediction accuracy. We understand that the former deals with increased computational complexity, and a greater risk of overfitting, whereas the latter is prone to information loss, and while extracted features are interpretable in a mathematical sense, they may not have direct, intuitive interpretations in the context of the problem domain. In the larger sense, the whole process maybe time intensive.

In their work, Markovitch and Rosenstein (2002) propose the FICUS algorithm , which is an iterative approach to apply beam search over a defined feature space and constructs new features by applying constructor functions to the members of its current feature set. The utility of these features are evaluated based on metrics like information gain and mutual information.

The Cognito algorithm proposed by Khurana et al. (2016). considered a greedy exploitation strategy to maximize the accuracy using various features constructed. The selection is based on determining the best-performing models through incremental data allocation and on estimating the model performances based on upper bounds.

Fan et al. (2019). in their work proposed a framework to use a divide-and-conquer approach instead of exhaustive enumeration. The features created using this approach are assessed locally within specific sets of examples where errors are high and the features made so far aren't predicting well. While the above mentioned frameworks are theoretically plausible, there doesn't seem to exist an open source library to use these approaches for practical data science use cases.

Python libraries like *scikit-learn*, *featuretools*, and *tsfresh* exist, but have drawbacks with not being able to deal with large datasets, or being targeted towards a

specific ML task. To overcome this, Horn et al. (2020). created a Python library called autofeat, which provides the AutoFeatRegression model. This model automatically generates and selects additional non-linear input features given the original data and then trains a linear regression model with these features. In a sense, this model demonstrates how a high prediction accuracy can be attained while maintaining a transparent model that produces verifiable outcomes as a foundation for business decisions made by non-statisticians.

One of the primary challenges in continual learning is catastrophic forgetting, where a model trained sequentially on multiple tasks tends to forget previously learned information when trained on new tasks. Techniques such as rehearsal (using a memory buffer of past examples), regularization (penalizing changes to important weights), and architectural approaches (dynamically expanding the network) are often employed to mitigate this issue. We also find that effectively managing memory resources is critical in continual learning.

When learning new features, we want to allow knowledge of previous tasks to be protected. Elastic weight consolidation, proposed by Kirkpatrick et al. (2017). addresses the significant problem continual learning poses for neural networks. A quadratic penalty was applied to the difference between the parameters for the new task and those for the old task. This penalty was weighted using a diagonal matrix, with weights proportional to the diagonal elements of the Fisher information matrix evaluated for the old parameters on the old task. This approach has been shown to support continual learning in challenging reinforcement learning scenarios, such as Atari 2600 games.

In their work in Progress & compress: A scalable framework for continual learning Schwarz et al. (2018), the "progress" component involves using the EWC to preserve knowledge from previously learned tasks by applying regularization to important model parameters, while the "compress" component employs a generative model to summarize and store essential information from past tasks. This framework

enables scalable and efficient continual learning by mitigating the negative impact of new tasks on previously acquired knowledge.

Memory Aware Synapse (MAS) proposed by Aljundi et al. (2018) identifies and prioritizes the importance of model parameters for previously learned tasks by calculating the sensitivity of the loss function to changes in these parameters. This approach uses regularization to protect critical weights from modification when training on new tasks, thereby preserving valuable knowledge. The MAS method enhances the model's ability to maintain performance on old tasks while accommodating new ones, offering a robust solution for continual learning scenarios where preserving past knowledge is crucial.

Lopez-Paz and Ranzato (2017) proposed a Gradient based episodic memory (GEM) for continual learning which stores a subset of examples from previous tasks and leverages these examples during training on new tasks to constrain the updates to the model's parameters. This approach ensures that the gradient updates for new tasks do not adversely affect the performance on previously learned tasks, effectively preserving past knowledge while enabling the model to adapt to new data. GEM offers a practical solution for continual learning by balancing the integration of new information with the retention of previous learning.

Chapter 3: Problem Formulation

3.1 Formal Problem Statement

Our continual learning algorithm relies on the development of manufactured features. Therefore, we concentrate on the task of generating interpretable features from a set of raw features. To illustrate our approach, we start with a motivating example. Subsequently, we formalize the feature construction problem. Lastly, we introduce a model that allows for the identification of optimal manufactured features, providing a benchmark for evaluating any feature construction algorithm.

Example 1 (Vehicles and Their Types). Consider a dataset of images of various vehicles. The raw features of each image are represented by the pixel values. Several classification tasks can be performed on this dataset. For instance, class labels might differentiate between vehicles based on their type (e.g., cars vs. trucks), fuel type (e.g., electric vs. gasoline), or size (e.g., compact vs. large).

The ideal manufactured features are those that are useful across any of these classification tasks.

Intuitively, these features might correspond to vehicle categories such as body style or fuel efficiency. Each vehicle category can be classified into different types such as compact, sedan, or SUV, and into different fuel types like electric or hybrid. Thus, each class (e.g., cars or electric vehicles) can be seen as a union of a subset of these categories. A human can often identify a vehicle's category or fuel type from its raw features (the image pixels). Our goal is to replicate this identification process using the class labels for a specific classification task.

Generalizing Example 1, suppose the data comprises various types of entities. The ideal manufactured features would be binary indicators f_k, where each f_k equals 1 if a data point belongs to type k, and 0 otherwise. However, the training data only

provides raw features and class labels for each point, and the number of types may be unknown. Thus, the problem can be articulated as follows:

Problem 1 (Manufactured Features). Develop binary features that correspond to the underlying data types.

A seemingly direct approach might be to use clustering techniques. However, clustering methods are generally effective for low-dimensional problems with a limited number of clusters. In contrast, we are dealing with datasets that have thousands of raw features and an unknown number of data types. Additionally, clustering methods typically do not leverage class labels, which can provide valuable information. Data points with different class labels are unlikely to belong to the same type. Therefore, incorporating class labels can enhance performance beyond what standard clustering algorithms can achieve.

Consider a data point $\boldsymbol{x}$ that belongs to type k, meaning $f_k(\boldsymbol{x}) = 1$. Given this type, we can accurately predict its class label. For instance, an vehicle identified as a *SUV* can be classified as *electric*. Conversely, if we know that $f_k(\boldsymbol{x}) = 0$, this alone is insufficient to ascertain the class of $\boldsymbol{x}$. For example, knowing only that an vehicle is *not* a SUV does not inform us whether it is electric. Additional information is required for class prediction. Therefore, the feature f_k serves as a *one-way classifier*, as defined below.

Definition 1 (One-way classifier). Consider dataset with points $\boldsymbol{x}_j \in \mathbb{R}^d$ and labels $y_j \in \{0, 1\}$. A one-way classifier is a function $g : \mathbb{R}^d \to \{0, 1\}$ such that when $g(\boldsymbol{x}_j) = 1$, then $y_j = 1$. However, when $g(\boldsymbol{x}_j) = 0$, we cannot predict y_j.

Problem 2 (One-way classifiers). Given a dataset $(\boldsymbol{x}_j, y_j) \in \mathbb{R}^d \times \{0, 1\}$ with $j \in [n]$, find all one-way classifiers.

Since ideal manufactured features are equivalent to one-way classifiers, Problem 1 can be reduced to Problem 2. To further simplify, we focus on constructing a single one-way classifier at a time.

Problem 3 (One-Way Classifier Around a Seed). Given a seed point $(\boldsymbol{x}^s, y^s)$ of some type k (where $f_k(\boldsymbol{x}^s) = 1$, with k unknown a priori), identify a one-way classifier that recognizes all points of type k. Specifically, the classifier should differentiate between the sets $\mathcal{S}_0$ and $\mathcal{S}_1$ defined as:

$$\mathcal{S}_1 := \{\boldsymbol{x} \mid f_k(\boldsymbol{x}) = 1\}, \qquad \mathcal{S}_0 := \{\boldsymbol{x} \mid f_k(\boldsymbol{x}) = 0\}, \tag{3.1}$$

where the dependency of $\mathcal{S}_1$ and $\mathcal{S}_0$ on k is implied by the context.

If we solve Problem 3 for multiple seed points, this enables the construction of all one-way classifiers. Consequently, Problem 3 is sufficient to address Problem 2. Additionally, each instance of Problem 3 can be solved concurrently.

To tackle Problem 3, it is necessary to obtain sufficient samples from $\mathcal{S}_0$ and $\mathcal{S}_1$. However, our information about these sets is incomplete. Specifically, we know that $\mathcal{S}_0$ includes all points whose class differs from y^s, while $\mathcal{S}_1$ only includes the seed point $\boldsymbol{x}^s$. Formally,

$$\mathcal{S}_0 \supseteq \mathcal{H}_0 := \{\boldsymbol{x}_j \mid y_j \neq y^s\}, \qquad \mathcal{S}_1 \supset \mathcal{H}_1 := \{\boldsymbol{x}^s\}. \tag{3.2}$$

To construct an effective one-way classifier, additional samples from $\mathcal{S}_1$ are needed. A straightforward approach would be to expand $\mathcal{H}_1$ by including other points near $\boldsymbol{x}^s$ based on pairwise distance or angle. However, this method often fails in high-dimensional spaces.

We introduce a data generation model that allows us to identify the ideal manufactured features. This model serves as a testbed for algorithms addressing Problem 2. Additionally, our analysis of this model illustrates why straightforward methods for expanding $\mathcal{H}_1$ are inadequate, providing motivation for the proposed method discussed in the next Chapter

In our model, data points are distributed across K different types on a d-dimensional unit sphere $\mathcal{S}^{d-1}$, where both d and K are assumed to be large. Each

type $k \in [K]$ is associated with a location on the sphere, denoted as $\boldsymbol{z}_k \in \mathcal{S}^{d-1}$, and a class label $h_k \in \{0, 1\}$.

To generate the j-th data point, we first sample its type $\theta_j \in [K]$ from a multinomial distribution. Next, we draw a point from a Gaussian distribution $\mathcal{N}(\boldsymbol{z}_{\theta_j}, \tau^2 I_d)$ and project it onto $\mathcal{S}^{d-1}$ to obtain $\boldsymbol{x}_j$. The class label of $\boldsymbol{x}_j$ is then assigned based on its type: $y_j = h_{\theta_j}$.

The details of the model are specified below.

Definition 2. The model $\mathcal{M}(n, d, K, \tau^2, p)$ generates n data points with features $\boldsymbol{x}_j \in \mathcal{S}^{d-1}$ and class labels $y_j \in \{0, 1\}$ (where $j \in [n]$) as follows:

$$
\begin{aligned}
\boldsymbol{z}_k &\sim \text{Haar}(\mathcal{S}^{d-1}) \quad \text{for all } k \in [K] \\
h_k &\sim \text{Bernoulli}(p) \quad \text{for all } k \in [K] \\
\theta_j &\sim \text{Multinomial}(1/K, \ldots, 1/K) \quad \text{for all } j \in [n] \\
\tilde{\boldsymbol{x}}_j \mid \{\theta_i\}_{i\in[n]}, \{\boldsymbol{z}_k\}_{k\in[K]} &\sim \mathcal{N}(\boldsymbol{z}_{\theta_j}, \tau^2 I_d) \\
\boldsymbol{x}_j &= \frac{\tilde{\boldsymbol{x}}_j}{\|\tilde{\boldsymbol{x}}_j\|_2} \\
y_j \mid \{\theta_i\}_{i\in[n]} &= h_{\theta_j}
\end{aligned}
\tag{3.3}
$$

In this model, the class label y_j of point j is solely dependent on its type θ_j. Therefore, *the ideal features must capture the information about* $\{\theta_j\}$. Specifically, the ground truth binary features f_k for $k \in [K]$ are defined such that $f_k(\boldsymbol{x}_j) = 1$ if and only if $\theta_j = k$.

When $\theta_j = k$, the raw features $\boldsymbol{x}_j$ are distributed around the central point $\boldsymbol{z}_k$. Consequently, the ideal feature f_k functions as a classifier that distinguishes $\boldsymbol{z}_k$ from all other centers $\{\boldsymbol{z}_\ell\}_{\ell\neq k}$.

Consider Problem 3 within the framework of the model described above. To solve this problem, we need to expand $\hat{\mathcal{S}}_1$ by including other points that are likely to belong to the same type as the seed.

3.2 Proposed Method

We established that the ideal features are one-way classifiers. To train such a classifier using a seed $\boldsymbol{x}^s$, it is essential to expand the training set with additional points of the same type as the seed. However, relying on pairwise computations to identify these points is ineffective. Therefore, an alternative approach is necessary.

Under our model (Definition 2), points of the same type are drawn from the same distribution and thus should reside within a local neighborhood, provided the distribution is not overly diffuse. We determine this local neighborhood using a robust classifier, which leverages the overall data distribution and avoids the limitations of pairwise metrics. We iteratively update the local neighborhoods and the robust classifiers to construct the features.

We summarize our approach and discuss the construction of local neighborhoods in detail. Subsequently, we present detailed algorithms for feature construction and continual learning. We begin by training a robust classifier $\mathcal{C}$ to distinguish the seed from all points whose class differs from that of the seed. This process defines a region $\mathcal{T}$ that exclusively contains points from the seed's class. Importantly, $\mathcal{T}$ is *localized* around the seed, making it highly likely that points within $\mathcal{T}$ share the same type as the seed (denoted as type k).

Under our model, the distribution of points of type k is centered around a location $\boldsymbol{z}_k$. Therefore, the principal direction of the points within $\mathcal{T}$ approximates $\boldsymbol{z}_k$. We use this direction to update the seed and iterate the process.

Each iteration shifts the seed towards the central location $\boldsymbol{z}_k$ for type k. As a result, $\mathcal{T}$ increasingly contains better samples of type-k points, which improves the seed for the subsequent iteration. Consequently, after a few iterations, the seed converges to $\boldsymbol{z}_k$. The resulting robust classifier $\mathcal{C}$ then serves as the ideal one-way classifier.

The classifier $\mathcal{C}$ must distinguish between $\mathcal{H}_1 = \{\boldsymbol{x}^s\}$ and the set $\mathcal{H}_0$ of all points from a different class (as defined in Eq. 3.2). Since $\mathcal{H}_1$ contains only a single

point, the problem is underspecified. To address this, we consider a robust version of the problem by replacing $\mathcal{H}_1$ with a distribution centered around $\boldsymbol{x}^s$.

Specifically, we define the distribution

$$\mathcal{D}^+ := \text{Project}_{\mathcal{S}^{d-1}}(\mathcal{N}(\boldsymbol{x}^s, \sigma^2 I_d)), \tag{3.4}$$

which is formed by adding Gaussian noise to $\boldsymbol{x}^s$ and projecting the resulting points onto the unit sphere.

The goal is for $\mathcal{C}$ to discriminate between the distribution $\mathcal{D}^+$ (positive class) and the point set $\mathcal{H}_0$ (negative class).

Intuitively, $\mathcal{C}$ will identify a neighborhood $\mathcal{T}$ around $\boldsymbol{x}^s$ that contains only points of class y^s. The parameter σ in $\mathcal{D}^+$ controls the degree of localization of $\mathcal{T}$. When σ is small, $\mathcal{T}$ is closer to $\boldsymbol{x}^s$ and covers a smaller region of $\mathcal{S}^{d-1}$. Thus, points in $\mathcal{T}$ are more likely to belong to the seed's type. However, if σ is too small, $\mathcal{T}$ may include too few points. Cross-validation is used to select the optimal σ that achieves a good balance.

To train $\mathcal{C}$, we need to minimize the empirical loss over $\mathcal{H}_0$ plus the expected loss over $\mathcal{D}^+$. The formula for the expected loss over $\mathcal{D}^+$ is not straightforward. We first show that in high-dimensional settings, $\mathcal{D}^+$ can be approximated by a simpler distribution $\mathcal{D}'$. We then derive a closed-form formula for the expected loss over $\mathcal{D}'$.

We now demonstrate how to eliminate the projection operator in $\mathcal{D}^+$ (Eq. 3.4). Consider a classifier $\mathcal{C}$ parameterized by $(\beta, \boldsymbol{c}) \in \mathbb{R} \times \mathbb{R}^d$, where $\mathcal{C}(\boldsymbol{x}) = \mathbb{1}_{\beta + \boldsymbol{c}^T \boldsymbol{x} > 0}$.

The local neighborhood $\mathcal{T}$ is then given by

$$\mathcal{T} = \{\boldsymbol{x} \in \mathcal{S}^{d-1} \mid \mathcal{C}(\boldsymbol{x}) = \mathcal{C}(\boldsymbol{x}^s)\}.$$

To determine whether a point $\boldsymbol{x}$ belongs to $\mathcal{T}$, we need only compute the sign of $\boldsymbol{c}^T \boldsymbol{x} + \beta$. Since the sign is invariant to scaling, we can restrict $\boldsymbol{c}$ to be of unit norm.

Recall that the classifier $\mathcal{C}(\boldsymbol{x})$ is expected to output one for $\boldsymbol{x} \sim \mathcal{D}^+$ and zero for the point set $\mathcal{H}_0$. Hence, the expected misclassification loss of $\mathcal{C}$ in $\mathcal{D}^+$ is given by

$$E_{\boldsymbol{x}\sim\mathcal{D}^+}\mathcal{C}(\boldsymbol{x}) = E_{\boldsymbol{x}\sim\mathcal{D}^+}[\mathbb{1}_{\beta+\boldsymbol{c}^T\boldsymbol{x}>0}].$$

Now, suppose the classifier's loss on a point $\boldsymbol{x}$ is a function of $\boldsymbol{c}^T\boldsymbol{x}$. For instance, the misclassification loss is such a function, as $\mathcal{C}(\boldsymbol{x}) = \mathbb{1}_{\beta+\boldsymbol{c}^T\boldsymbol{x}>0}$. Theorem **??** shows that, in high-dimensional settings, the expected loss under $\mathcal{D}'$ and $\mathcal{D}^+$ are approximately the same. Since $\mathcal{D}'$ is easier to analyze because it is Gaussian, we can use it in place of $\mathcal{D}^+$ for training the classifier. Specifically, in $\mathcal{D}'$, the mean $\boldsymbol{x}^s$ and the standard deviation σ from $\mathcal{D}^+$ are scaled by a factor of $1/\sqrt{1+\sigma^2 d}$.

Loss Minimization over $\mathcal{D}'$: To classify $\mathcal{H}_0$ against $\mathcal{D}'$, we need to compute the classifier's loss. We use a weighted hinge loss, where the loss on a point with features $\boldsymbol{x} \in \mathcal{S}^{d-1}$, class $y \in \{0,1\}$, and weight w is given by

$$\ell(y, \boldsymbol{x}, w \mid y^s, \beta, \boldsymbol{c}) := w \cdot \max\left(0, 1 - (2 \cdot \mathbb{1}_{y=y^s} - 1) \cdot (\beta + \boldsymbol{c}^T\boldsymbol{x})\right). \tag{3.5}$$

The weight w can be set to 1 if the points are unweighted. The overall loss is given by

$$\frac{1}{|\mathcal{H}_0|} \sum_{(\boldsymbol{x},w)\in\mathcal{H}_0} \ell(1-y^s, \boldsymbol{x}, w) + E_{\boldsymbol{x}\sim\mathcal{D}'}\ell(y^s, \boldsymbol{x}, 1), \tag{3.6}$$

where we extend $\mathcal{H}_0$ to include weights for data points.

The expected loss over $\mathcal{D}'$ can be approximated by the empirical loss over samples drawn from $\mathcal{D}'$. However, sampling introduces variability and increases computational burden. Instead, we can directly compute the second term of Eq. 3.6 as follows.

Theorem 3.1 (Adapting Theorem 1 of Chakrabarti and Fauber (2022))**.** *Define*

$$s = 1 - \beta - \frac{\boldsymbol{c}^T\boldsymbol{x}^s}{\sqrt{1+\sigma^2 d}}, \tag{3.7}$$

$$t = \frac{\|\boldsymbol{c}\|_2 \cdot \sigma}{\sqrt{1+\sigma^2 d}}. \tag{3.8}$$

Then, we have

$$E_{\boldsymbol{x}\sim\mathcal{D}'}\ell(y^s, \boldsymbol{x}, 1 \mid y^s, \beta, \boldsymbol{c}) = s \cdot \Phi\left(\frac{s}{t}\right) + t \cdot \phi\left(\frac{s}{t}\right), \tag{3.9}$$

where $\phi(\cdot)$ and $\Phi(\cdot)$ denote the probability density function (pdf) and cumulative distribution function (cdf) of the standard normal distribution, respectively.

Thus, the expected loss over $\mathcal{D}'$ has a closed-form expression. Furthermore, the classifier's overall loss is convex Chakrabarti and Fauber (2022). which means it can be optimized by standard methods.

Chapter 4: Algorithm

We present two applications of our algorithm. First, we discuss how to generate features for a given dataset. Then, we address the problem of creating features to enhance the accuracy of a pre-existing classifier.

Algorithm 1 outlines the details of our approach. We process multiple seeds $\{\boldsymbol{x}_i\}$ in parallel. All seeds share the same class y_s and therefore have the same $\mathcal{H}_0$.

In Step 5, we construct classifiers by optimizing Eq. 3.6 using Theorem **??**. For each seed $\boldsymbol{x}_i$, this yields a classifier $\mathcal{C}_i$ with weight vector $\boldsymbol{c}_i$ and intercept β_i.

We use the points selected by $\mathcal{C}_i$ to update the seed for the next iteration (Step 11). After the final iteration, we return the classifiers $\{\mathcal{C}_i\}$. These classifiers can then be used to generate features for any test point.

To improve the accuracy of PATCH, we introduce two additional steps. First, $\mathcal{C}_i$ might not select enough points, meaning that $|\{j \mid y_j = y_s \text{ and } \mathcal{C}_i(\boldsymbol{x}_j) = 1\}|$ could be small. This limitation can affect the seed update accuracy (Step 11). To address this, we adjust the intercept of $\mathcal{C}_i$ while keeping $\boldsymbol{c}_i$ fixed (Step 16). This adjustment maximizes the number of selected points while ensuring they primarily belong to the class y_s.

Second, we evaluate whether the classifiers are distinct from each other (Step 8). If the classifiers become too similar (i.e., if they select overlapping subsets of points), we terminate the iterations early to prevent redundant classifiers.

PATCH can also be applied to enhance the accuracy of an existing classifier $\mathcal{H}$.

We assume that $\mathcal{H}$ has been calibrated and $p_j(0) := \mathcal{H}(\boldsymbol{x}_j)$ represents the estimated probability that y_j is 1. Given a one-way classifier $\mathcal{C}$, we can adjust the prediction to $p_j(\gamma) := (\mathcal{H} + \gamma\mathcal{C})(\boldsymbol{x}_j) := \mathcal{H}(\boldsymbol{x}_j) + \gamma\mathcal{C}(\boldsymbol{x}_j)$ for some $\gamma \in \mathbb{R}$. Our objective is to minimize the prediction loss, which we choose to be the squared error

Algorithm 1 Automatically generate features as one-way classifiers

function PATCH(class $y^{(s)} \in \{0,1\}$, training data $P = \{\boldsymbol{x}_j, y_j, w_j\}$, spread σ, max iterations m, repetition threshold θ, hurdle h)
  $\{\boldsymbol{x}_i^{(s)}\} \leftarrow$ sample of high-weight points from class $y^{(s)}$ ▷ Seeds
  $\hat{\mathcal{S}}_0 \leftarrow \{(\boldsymbol{x}_j, w_j) \mid y_j \neq y^{(s)}\}$ ▷ For large $|\hat{\mathcal{S}}_0|$, we only use a sample
  for counter $= 1, \ldots, m$ **do**
    $\mathcal{C}_i \leftarrow$ classify $\text{Project}_{S^{d-1}}(\mathcal{N}(\boldsymbol{x}_i^{(s)}, \sigma^2 I_d))$ versus $\hat{\mathcal{S}}_0$, for all seeds i
    $\mathcal{C}_i \leftarrow$ ADJUST$(\mathcal{C}_i, y^{(s)}, P, h)$ for all seeds i
    $Y_{ij} \leftarrow \mathcal{C}_i(\boldsymbol{x}_j)$ for all seeds i and training points j ▷ $Y_{ij} \in \{0,1\}$
    if top singular value of matrix $Y > \theta$ **then**
      break ▷ Too many overlapping features
    else
      $\boldsymbol{x}_i^{(s)} \leftarrow$ top principal component of $\{w_j \cdot \boldsymbol{x}_j \mid y_j = y^{(s)}$ and $\mathcal{C}_i(\boldsymbol{x}_j) = 1\}$, for all seeds i
    end if
  end for
  return $\{\mathcal{C}_i\}$
end function
function ADJUST$(\mathcal{C}, y^{(s)}, P,$ hurdle $h)$
  $\boldsymbol{c} \leftarrow$ feature weights from $\mathcal{C}$
  $J \leftarrow$ sort $\{j \in P\}$ in descending order of $\boldsymbol{x}_j^T \boldsymbol{c}$
  $r_{J_k} \leftarrow \sum_{\ell \leq k} w_{J_\ell} \cdot (2 \cdot \mathbb{1}_{y_{J_\ell} = y^{(s)}} - 1)$ for any $J_k \in J$
  $k^\star \leftarrow \max(50, \max_k \{r_{J_k} > h\})$
  $\beta^\star \leftarrow -\boldsymbol{x}_{J_k^\star}^T \boldsymbol{c}$
  return classifier with intercept $\beta^\star$ and weights $\boldsymbol{c}$
end function

Algorithm 2 Generate features that augment an existing classifier

function PATCHAUGMENT(training points $P = \{\boldsymbol{x}_j, y_j\}$, existing classifier $\mathcal{H}$, PATCH parameters (σ, m, θ, h))
  $P_{\text{train}}, P_{\text{valid}} \leftarrow$ split P into training and validation sets
  while AUC on P_{valid} is improving **do**
    $\mathcal{H} \leftarrow$ DE-TREND$(\mathcal{H}, P_{\text{train}}, P_{\text{valid}})$ ▷ Identify global patterns
    $w_j \leftarrow |y_j - \mathcal{H}(\boldsymbol{x}_j)|$ ▷ Set weights based on $\mathcal{H}$'s errors
    Select $y^{(s)} \in \{0, 1\}$ randomly
    $\{\mathcal{C}_i\} \leftarrow$ PATCH$(y^{(s)}, \{\boldsymbol{x}_j\}, \{w_j\}, \sigma, m, \theta)$ ▷ Identify local patterns
    $\mathcal{C}'_i \leftarrow$ SMOOTHC$(\mathcal{C}_i, y^{(s)}, P_{\text{train}})$ for each $\mathcal{C}_i$ ▷ Binary to real-valued features
    $\mathcal{H} \leftarrow$ UPDATE$(\mathcal{H}, \{\mathcal{C}'_i\}, P_{\text{valid}})$
  end while
  return $\mathcal{H}$
end function
function DE-TREND$(\mathcal{H}, P_{\text{train}}, P_{\text{valid}})$
  $\beta, \boldsymbol{c} \leftarrow$ Elastic Net regression to predict $\{y_j - \mathcal{H}(\boldsymbol{x}_j)\}$ using $\{\boldsymbol{x}_j\}$
  $\mathcal{C}_{\text{trend}} \leftarrow$ classifier with feature weights $\boldsymbol{c}$ and intercept β
  $\mathcal{C}'_{\text{trend}} \leftarrow$ SMOOTHC$(\mathcal{C}_{\text{trend}}, 1, P_{\text{train}})$
  $\mathcal{H} \leftarrow$ UPDATE$(\mathcal{H}, \mathcal{C}'_{\text{trend}}, P_{\text{valid}})$
  return $\mathcal{H}$
end function
function SMOOTHC$(\mathcal{C}, y^{(s)}, P_{\text{train}})$
  $\beta, \boldsymbol{c} \leftarrow$ intercept and feature weights from $\mathcal{C}$
  $J \leftarrow$ sort $\{j \in P_{\text{train}} \mid \beta + \boldsymbol{x}_j^T \boldsymbol{c} > 0\}$ in descending order of $\boldsymbol{x}_j^T \boldsymbol{c}$
  $r_{J_k} \leftarrow \left(\sum_{\ell \le k} w_{J_\ell} \cdot (2 \cdot \mathbb{1}_{y_{J_\ell} = y^{(s)}} - 1)\right) / \left(\sum_{\ell \le k} 1\right)$ for any $J_k \in J$
  $hi \leftarrow \arg\max_{k \ge \min(50, |J|)} r_{J_k}$
  $lo \leftarrow$ last element of J
  $\mathcal{C}'(\boldsymbol{x}) := \left(\max\left(0, \min\left(1, \frac{(\boldsymbol{x} - \boldsymbol{x}_{\boldsymbol{lo}})^T \boldsymbol{c}}{(\boldsymbol{x}_{hi} - \boldsymbol{x}_{lo})^T \boldsymbol{c}}\right)\right)\right) \cdot (r_{hi} - r_{lo})$ ▷ $\mathcal{C}' : \mathcal{S}^{d-1} \to \mathbb{R}$
  return $\mathcal{C}'$
end function
function UPDATE$(\mathcal{H}, \{\mathcal{C}'_i\}, P_{\text{valid}})$
  $\gamma^\star \leftarrow \arg\max_\gamma AUC(\{(\mathcal{H} + \gamma \cdot \sum \mathcal{C}_i)(\boldsymbol{x}_j), y_j\}_{j \in P_{\text{valid}}})$
  return $\mathcal{H} + \gamma^\star \cdot \sum \mathcal{C}_i$
end function

$\ell_{SE}(p_j \mid y_j) := (p_j(\gamma) - y_j)^2$.

For small γ, the change in loss is approximately given by

$$\begin{aligned}\ell_{SE}(p_j(\gamma) \mid y_j) - \ell_{SE}(p_j(0) \mid y_j) &\approx \ell'_{SE}(p_j(0) \mid y_j) \cdot \gamma \cdot \mathcal{C}(\boldsymbol{x}_j) \\ &= -2\gamma \cdot (y_j - \mathcal{H}(\boldsymbol{x}_j)) \cdot \mathcal{C}(\boldsymbol{x}_j) \\ &= -2\gamma \cdot \underbrace{(2y_j - 1)}_{\in\{-1,1\}} \cdot \underbrace{|y_j - \mathcal{H}(\boldsymbol{x}_j)|}_{w_j} \cdot \mathcal{C}(\boldsymbol{x}_j).\end{aligned}$$

Here, $\mathcal{C}(\boldsymbol{x}_j) = 1$ when the feature is "on" for $\boldsymbol{x}_j$, and 0 otherwise. Thus, the loss only changes for points selected by $\mathcal{C}$. If $\gamma > 0$, the loss decreases for points belonging to the positive class and increases otherwise. Therefore, to construct $\mathcal{C}$ using PATCH, we should focus on points with high w_j, and choose $\gamma > 0$ if the seed $y_s = 1$, or $\gamma < 0$ if $y_s = 0$.

Algorithm 2 outlines the details of this approach. We first calculate the weights w_j for each data point (step 5) and use them to build the one-way classifiers $\{\mathcal{C}_i\}$ (step 7).

To update $\mathcal{H}$, we need to determine a γ_i for each $\mathcal{C}_i$. Directly searching for a γ_i for each classifier may lead to overfitting. Instead, we combine all $\mathcal{C}_i$ into a single feature and find the optimal γ for this combined feature. Specifically, we first convert each binary feature $\mathcal{C}_i$ into a real-valued feature $\mathcal{C}'_i$ (step 20). Intuitively, a data point $\boldsymbol{x}$ is more likely to belong to the seed's class the farther it is from the decision boundary. $\mathcal{C}'_i$ represents this probability via a piecewise linear function. We then use $\sum_i \mathcal{C}'_i$ as a single new feature and find the optimal $\gamma^\star$ for this feature (steps 9 and 30). Finally, we update $\mathcal{H}$ using the optimal $\gamma^\star$.

Algorithm 2 also includes an additional optimization. While PATCH identifies local patterns around seed points, it might miss global patterns that affect the entire dataset. Since we use squared error as our loss function, we use ridge regression to capture these global patterns (step 13). These global patterns are then used to update $\mathcal{H}$ in the same manner as the local patterns.

Matching the desired properties. A feature generation algorithm should satisfy the following properties:

- *Parallelizable:* The algorithm should be capable of generating multiple features simultaneously. To achieve this, the measurement of a feature's utility should be independent of other features being generated in parallel.

- *Checkable:* To validate the accuracy of the generated features, they should be compared against ground truth features. Ideally, these ground truth features should be "optimal" combinations of the raw features available in the dataset. Although there is no universally accepted notion of optimality for this problem, a checkable feature generation algorithm should be tested against a synthetic data distribution with well-defined ground truth features, which provides intuitive benchmarks for evaluating the algorithm's performance. This also implies that the generated features should be interpretable within this data distribution.

- *Appropriate for high-dimensional problems:* The algorithm should be designed to generate useful features through non-linear combinations of existing features, especially when dealing with high-dimensional datasets where raw features are numerous, such as individual pixels in an image.

- *Scalable:* In practical scenarios, the number of features required is often unknown. Therefore, the algorithm should not necessitate prior knowledge of the number of features and should be scalable to handle large numbers of features efficiently.

Both PATCH and PATCHAUGMENT are designed to search for features in *parallel*. There is no restriction on the number of seeds or the number of features, making our approach *scalable* to large-scale feature generation problems. Moreover, our algorithms are tailored for *high-dimensional* settings, adhering to the model specified in Eq. 3.3. Under this model, the ground-truth features correspond to the hidden

types of the data points. Consequently, our algorithms are *checkable*, as we can assess the accuracy of feature generation using this model, as demonstrated in the experiments. In summary, our algorithms meet all the desired properties for effective feature generation.

Chapter 5: Experiments

We conduct experiments to address two key questions:

1. **Accuracy of Features Generated by Patch:** We aim to verify whether the features generated by PATCH accurately capture the underlying data structure and are aligned with the ground truth features.

2. **Effectiveness of PatchAugment in Enhancing Existing Classifiers:** We assess whether PATCHAUGMENT improves the performance of pre-existing classifiers by incorporating the features it generates.

To answer these questions, we perform experiments on both simulated and real-world datasets. Specifically, we use 4 real-world datasets to validate the effectiveness of our methods.

We performed the experiments by first splitting the data into training and testing sets. Specifically, the dataset was divided such that 70% of the data was used for training and 30% for testing. To ensure the robustness of our model, we further split the training data into actual training and validation subsets. This was achieved by creating a validation set with a fraction of the original training data, where the fraction size was determined dynamically to balance the dataset.

For each class, a baseline model was initially trained using Gradient Boosting and Support Vector Machine. This step involved evaluating the model performance on the training data. Subsequently, we applied PATCHAUGMENT, specifically on the validation subset. PATCHAUGMENT was tested with various hyperparameter settings to optimize the model's performance. Each combination of hyperparameters was evaluated by training the model and assessing its performance on the validation set. The metric used for evaluation was the Area Under the Curve (AUC) score.

After obtaining results from the initial tuning, we selected the best hyperparameter values based on their AUC performance on the validation set. The best values were determined by sorting the validation results and choosing the parameters that yielded the highest AUC scores.

With the best hyperparameters identified, we conducted a final set of experiments. This involved retraining the model with the optimal parameters and evaluating its performance on the test set.

Dataset	**Type**	**#classes**	**#raw features**	**#training points**
CIFAR-10	Image	10	1024	35000
CELEB-A	Image	2	4096	141819
SVHN	Image	10	3072	69502
Protein	Sparse matrix	3	357	12436

Table 5.1: *Dataset characteristics.*

Our experiments confirm that PATCH accurately generates features that align with the ground truth, and PATCHAUGMENT effectively boosts the performance of existing classifiers. Detailed results and analysis are provided below.

Chapter 6: Conclusion

We present a novel approach to feature engineering for supervised learning, focusing on the development and validation of the PATCH algorithm and its extension, PATCHAUGMENT. The primary contributions of this work are twofold:

1. **Automatic Feature Construction:** We introduced PATCH, an algorithm that constructs highly interpretable and verifiable features termed "one-way classifiers." These features are designed to predict the class accurately when active, enhancing the model's interpretability and reliability.

2. **Continual Learning Enhancement:** Through the PATCHAUGMENT algorithm, we extended the utility of Patch to residual learning. This approach iteratively improves baseline models by focusing on poorly predicted data points, thereby boosting overall performance and resilience.

6.1 Key Contributions

We demonstrate the effectiveness of these methods through extensive experimentation on multiple real-world datasets, showing significant improvements in AUC scores. The main contributions can be summarized as follows:

1. Robust Algorithm for Feature Manufacturing: PATCH identifies reliable one-way classifiers using a parallelizable and robust optimization approach. This method prevents overfitting and ensures each feature's quality is independently verifiable.

2. Innovative Approach to Residual Learning: PATCHAUGMENT leverages manufactured features to model residuals, maintaining the resilience and accuracy of the predictive model even in the presence of data corruption.

6.2 Implications for Future Research

The findings of this thesis open several avenues for future research:

1. Scalability and Efficiency: Further exploration into the scalability of PATCH and PATCHAUGMENT for larger and more complex datasets could enhance their applicability in real-world scenarios.

2. Integration with Other Learning Algorithms: Investigating the integration of PATCH with various other machine learning algorithms could provide deeper insights into its versatility and robustness.

3. Extended Applications: Applying the concepts of one-way classifiers and residual learning to other domains, such as natural language processing or image recognition, could yield promising results and broaden the scope of this research.

By framing feature construction as a one-way classification problem and introducing robust algorithms to implement this approach, we have shown that it is possible to create highly informative and interpretable features that significantly enhance model performance. The success of PATCH and PATCHAUGMENT in various experimental settings underscores the potential of these methods to transform how features are engineered and utilized in machine learning models. Future research building on these foundations can further refine and expand these techniques, contributing to the advancement of the field.

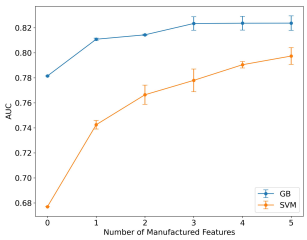
Appendix A: Detailed Experimental Results

Class	GB	GB+PatchAugment	SVM	SVM+PatchAugment
Airplane	0.7814	0.8237 ± 0.0057	0.6769	0.7974 ± 0.0067
Automobile	0.8316	0.87146 ± 0.0020	0.7523	0.8421 ± 0.0146
Bird	0.7257	0.7740 ± 0.0018	0.6362	0.7578 ± 0.0083
Cat	0.6987	0.7453 ± 0.0022	0.6033	0.7247 ± 0.0010
Deer	0.7046	0.7393 ± 0.0031	0.6514	0.73435 ± 0.0030
Dog	0.7428	0.7955 ± 0.0024	0.6658	0.7627 ± 0.01183
Frog	0.7607	0.80015 ± 0.0040	0.6548	0.7856 ± 0.0086
Horse	0.7818	0.8177 ± 0.0096	0.6373	0.7925 ± 0.0023
Ship	0.8232	0.8552 ± 0.0011	0.7510	0.8289 ± 0.0040
Truck	0.8467	0.8646 ± 0.0024	0.7893	0.8506 ± 0.0029

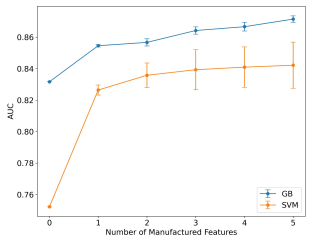
Table A.1: PatchAugment *on CIFAR-10:* The table shows the AUC scores obtained by Gradient Boosting and a Linear SVM, with and without PatchAugment.

Class	GB	GB+PatchAugment	SVM	SVM+PatchAugment
1	0.8977	0.9398 ± 0.0018	0.6280	0.9406 ± 0.0016
2	0.8837	0.9407 ± 0.0005	0.6041	0.9113 ± 0.0065
3	0.8151	0.8862 ± 0.0029	0.5600	0.8560 ± 0.0168
4	0.9031	0.9469 ± 0.0006	0.6124	0.9372 ± 0.0024
5	0.8428	0.9130 ± 0.0027	0.5804	0.8832 ± 0.0087
6	0.8274	0.8834 ± 0.0026	0.5790	0.8116 ± 0.0054
7	0.8661	0.9340 ± 0.0016	0.6036	0.9290 ± 0.0015
8	0.7873	0.8575 ± 0.0010	0.5899	0.8109 ± 0.0021
9	0.8182	0.9002 ± 0.0026	0.5924	0.8847 ± 0.0140
10	0.8407	0.9056 ± 0.0031	0.6474	0.9054 ± 0.0039

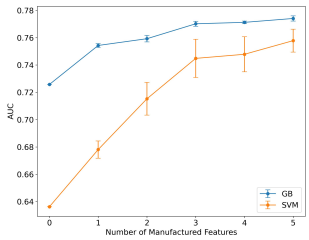
Table A.2: PatchAugment *on Street View House Numbers:* The table shows the AUC scores obtained by Gradient Boosting and a Linear SVM, with and without PatchAugment.



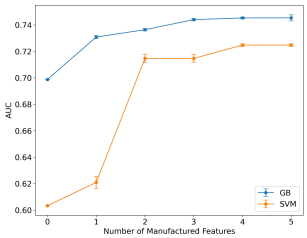
(a) Class Airplane



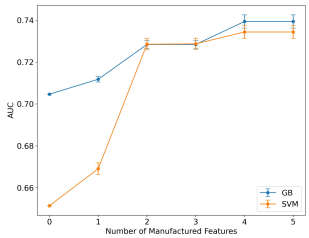
(b) Class Automobile



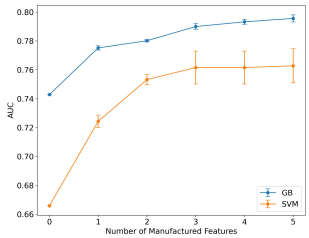
(c) Class Bird



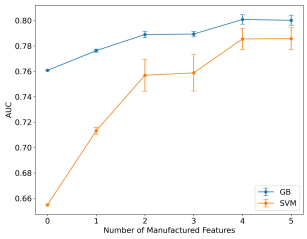
(d) Class Cat



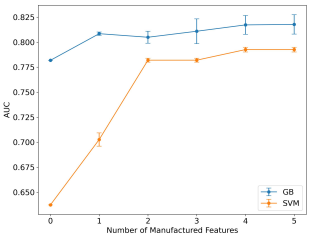
(e) Class Deer



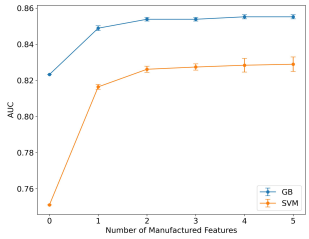
(f) Class Dog



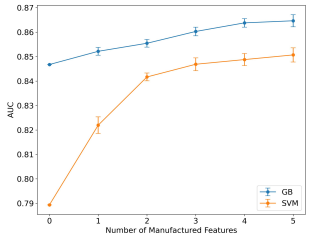
(g) Class Frog



(h) Class Horse

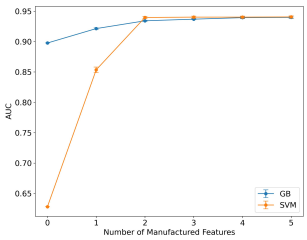


(i) Class Ship

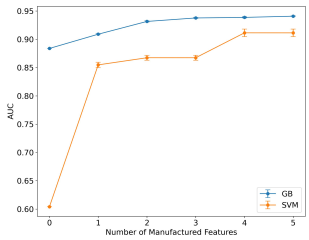


(j) Class Trunk

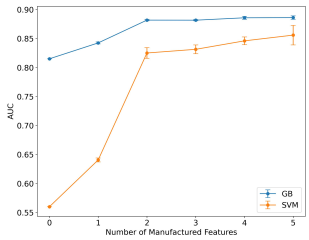
Figure A.1: PatchAugment*'s accuracy versus number of manufactured features on CIFAR-10:* Each plot shows the AUC for a binary one-versus-the-rest classifier, using PatchAugment to augment Gradient Boosting and a linear SVM. PatchAugment yields a consistent and significant improvement in AUC.



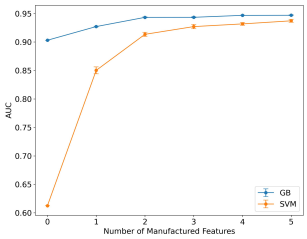
(a) Class 1



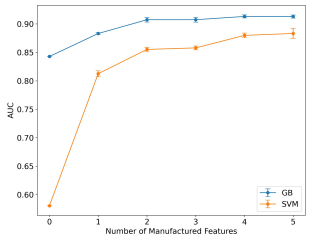
(b) Class 2



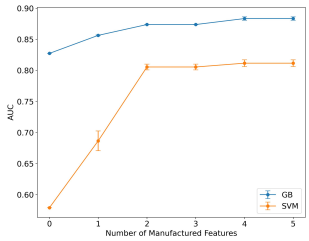
(c) Class 3



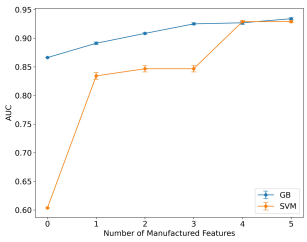
(d) Class 4



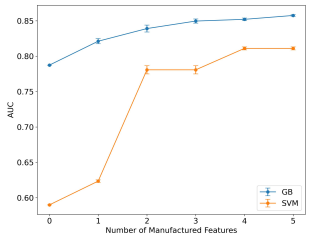
(e) Class 5



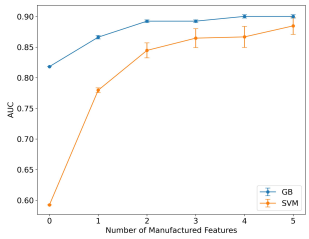
(f) Class 6



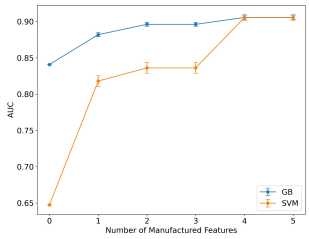
(g) Class 7



(h) Class 8



(i) Class 9



(j) Class 10

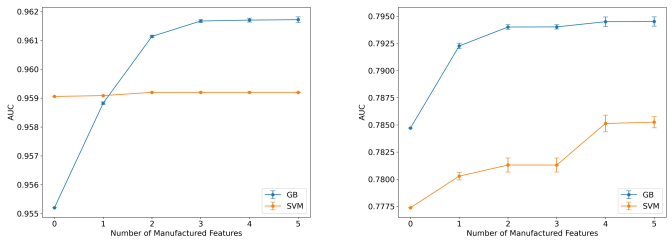
Figure A.2: PatchAugment*'s accuracy versus number of manufactured features on Street View House Numbers:* Each plot shows the AUC for a binary one-versus-the-rest classifier, using PatchAugment to augment Gradient Boosting and a linear SVM. PatchAugment yields a consistent and significant improvement in AUC.

Class	GB	GB+PatchAugment	SVM	SVM+PatchAugment
Smiling	0.9552	0.9617 ± 0.0001	0.9590	0.9591 ± 0.0000
Pointy Nose	0.7847	0.7945 ± 0.00042	0.7773	0.7852 ± 0.0005

Table A.3: PatchAugment *on CelebA:* The table shows the AUC scores obtained by Gradient Boosting and a Linear SVM, with and without PatchAugment.

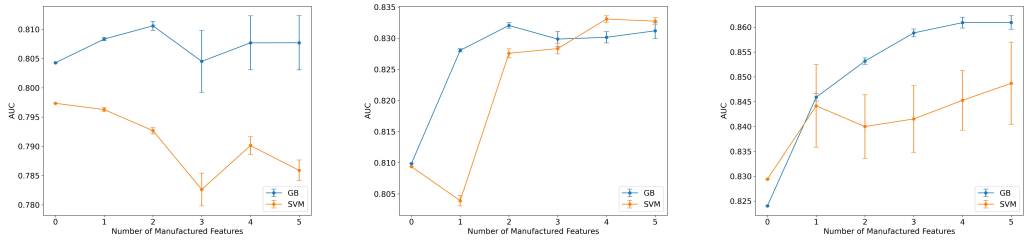
Class	GB	GB+PatchAugment	SVM	SVM+PatchAugment
0	0.8042	0.8076 ± 0.0046	0.7973	0.7858 ± 0.0017
1	0.8098	0.8311 ± 0.0012	0.8093	0.8327 ± 0.0005
2	0.8240	0.8609 ± 0.0013	0.8294	0.8486 ± 0.0082

Table A.4: PatchAugment *on Protein:* The table shows the AUC scores obtained by Gradient Boosting and a Linear SVM, with and without PatchAugment.



(a) Class Smiling (b) Class Pointy Nose

Figure A.3: PatchAugment*'s accuracy versus number of manufactured features on CelebA For Class Smiling and Class Pointy Nose:* Each plot shows the AUC for a binary one-versus-the-rest classifier, using PatchAugment to augment Gradient Boosting and a linear SVM. PatchAugment yields a consistent and significant improvement in AUC.



(a) Class 0 (b) Class 1 (c) Class 2

Figure A.4: PatchAugment*'s accuracy versus number of manufactured features on Protein For Class 0,1,2:* Each plot shows the AUC for a binary one-versus-the-rest classifier, using PatchAugment to augment Gradient Boosting and a linear SVM. PatchAugment yields a consistent and significant improvement in AUC.

Glossary

Rahul Nandakumar A hard-working graduate student, nearing the end of this chapter of your education!

Master of Science Rahul Nandakumar is about to earn this. Congratulations!

Works Cited

Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European conference on computer vision (ECCV)*, pages 139–154, 2018.

Deepayan Chakrabarti and Benjamin Fauber. Robust High-Dimensional Classification From Few Positive Examples. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1952–1958, July 2022.

Cheng Fan, Yongjun Sun, Yang Zhao, Mengjie Song, and Jiayuan Wang. Deep learning-based feature engineering methods for improved building energy prediction. *Applied energy*, 240:35–45, 2019.

Franziska Horn, Robert Pack, and Michael Rieger. The autofeat python library for automated feature engineering and selection. In *Machine Learning and Knowledge Discovery in Databases: International Workshops of ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part I*, pages 111–120. Springer, 2020.

Udayan Khurana, Deepak Turaga, Horst Samulowitz, and Srinivasan Parthasrathy. Cognito: Automated feature engineering for supervised learning. In *2016 IEEE 16th international conference on data mining workshops (ICDMW)*, pages 1304–1307. IEEE, 2016.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

Shaul Markovitch and Dan Rosenstein. Feature generation using general constructor functions. *Machine Learning*, 49:59–98, 2002.

Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *International conference on machine learning*, pages 4528–4537. PMLR, 2018.

Vita

Rahul Nandakumar is going to graduate soon!

Address: rahul.nandakumar@utexas.edu

This thesis was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.