OPTIMIZING A PREDICTION PIPELINE BY PREPENDING AN EFFICIENT
LOW-FIDELITY MODEL

A thesis presented to the faculty of
San Francisco State University
In partial fulfillment of
The Requirements for
The Degree

Master of Science
In
Computer Science

by

Jakob Dohrmann

San Francisco, California

December 2020

Copyright by
Jakob Dohrmann
2020

CERTIFICATION OF APPROVAL

I certify that I have read *OPTIMIZING A PREDICTION PIPELINE BY PREPENDING AN EFFICIENT LOW-FIDELITY MODEL* by Jakob Dohrmann and that in my opinion this work meets the criteria for approving a thesis submitted in partial fulfillment of the requirements for the degree: Master of Science in Computer Science at San Francisco State University.

Kazunori Okada
Professor of Computer Science

Hui Yang
Associate Professor of Computer Science

Arno Puder
Professor of Computer Science

OPTIMIZING A PREDICTION PIPELINE BY PREPENDING AN EFFICIENT LOW-FIDELITY MODEL

Jakob Dohrmann
San Francisco State University
2020

Anti-spam systems for email employ sophisticated machine learning (ML) models to combat evolving spammer strategies devised to thwart detection. Concurrently, high volume of spam demands high throughput, and accuracy and defensibility of verdicts are crucial to satisfy users. Addressing this multi-faceted optimization problem for balancing accuracy, throughput and cost that stand in opposition, we propose a pipeline framework for modeling spam detection in various application/business settings. We share a real-world labeled dataset consisting of low-fidelity features based on sender domain reputation (SDR), representing traffic flowing through such a pipeline. This thesis presents a method to prepend an efficient gradient boosting machine (GBM) model trained on SDR data to the existing pipeline by jointly optimizing these competing constraints along ROC curves. Furthermore, we extend recently proposed feature power measures to GBMs and apply them to analyze variable importance and to perform feature selection. We experimentally demonstrate effectiveness of proposed methods in various application scenarios over our dataset.

I certify that the Abstract is a correct representation of the content of this thesis.

Chair, Thesis Committee

Date

ACKNOWLEDGMENTS

I thank Dr. Kazunori Okada for being my advisor, for invaluable feedback, and for making it a joy to write this thesis. His flexibility in our engagement and his unwavering trust and encouragement were key to realizing this work.

I also thank Dr. Arno Puder and Dr. Hui Yang for always having an open door, their assistance during the graduation process, and for being on my committee.

Thanks to the staff, faculty and students at SFSU, particularly the Biocomputing Research Lab, for all they have taught me over the years and all the many ways in which they contributed to making this degree a transformative experience and a step towards Mastery in more than just Computer Science. Shout-out to Juris, Tom, and David, who remain friends and confidants.

I thank Dr. Yamilka for her council and support. I am glad to have met her at the time when it was most needed.

I appreciate my colleagues at work for putting up with odd hours and their encouragement. Special thanks goes to Adam and Lee for fruitful technical discussions and for lending their expertise.

Finally, I thank my wife Vica and my daughters Zoe and Elise. They stood with me and supported me through the long process accumulating in this degree. I am dedicated to forever express my gratitude towards my family in our shared life.

TABLE OF CONTENTS

1	Introduction	1
1.1	Spam Detection Pipeline Optimization	1
1.2	Proposed Approach and Contributions	3
1.3	Thesis Organization	6
2	Background and Prior Work	7
2.1	Spam Detection	7
2.2	Sender Domain Reputation	11
2.3	Gradient Boosting Machines	14
2.3.1	Basic Concepts	14
2.3.2	Decision Trees as Base-Learners	18
2.3.3	Differences Between GBM and RF	19
2.3.4	Simplified Notation for GBMs	20
2.3.5	Cost Efficient Gradient Boosting	21
2.4	Model Interpretation	22
2.4.1	Feature Importance Based on Gain and Split Count	23
2.4.2	Shapley values	24
2.4.3	SHAP and TreeSHAP	26
2.4.4	Feature Power	28
2.4.5	Comparison of Model Interpretation Methods	31

2.5	Feature Selection Based on Variable Rankings	33
3	Dataset	34
3.1	Data Duplication and Implications for Predictions	37
3.2	Dataset Size and Verdict Distribution	39
4	Theoretical Considerations	41
4.1	Modeling a Spam Detection Pipeline	41
4.2	Adding an Engine to the Pipeline	44
4.2.1	Excuse Into Cost Feasibility of Pipeline Modification	44
4.2.2	Feasibility Considerations for a New Engine	50
4.3	Expanding Feature Power	60
4.3.1	Feature Power for Regression Trees	60
4.3.2	Feature Power for GBM Regressors	64
4.3.3	Feature Power for GBM Classifiers	66
5	Experiments	71
5.1	Software Environment	71
5.2	Types of Predictions	72
5.3	Mean, Mode, and Aggregation Baselines	73
5.4	Baseline Predictions and Model Comparison	73
5.4.1	Comparison of GBM Implementations	75
5.4.2	Regression Versus Classification	80

5.4.3	Observed Lack of Overfitting in Deep Trees	83
5.5	Pipeline Optimization for Application Constraints	87
5.5.1	FP Avoidance	87
5.5.2	Reducing the Number of Features	94
5.6	Model Interpretation	99
5.6.1	Improving Ground Truth	99
5.6.2	Correlation of Model Interpretation Measures and Features . .	102
5.6.3	Feature Selection Based on Model Interpretation	106
6	Conclusion	114
6.1	Theoretical and Practical Contributions	114
6.2	Experimental Results	116
6.3	Future Work	119
	Appendix A: Example Data	122
	Appendix B: Mathematical Symbols	125
	Bibliography	127

LIST OF TABLES

Table		Page
5.1	Model comparison for the regression setting, ordered by validation score. Bold (maroon) numbers highlight best values in the set. LightGBM produces best scores, followed by XGBoost. Other GBMs and NeuralNet rank third and are statistically indistinguishable from each other at $p < 5\%$ (indicated by italics and shading).	76
5.2	Model comparison for binary classification in the aggressive setting, ordered by validation score. Baseline is given by prevalence of most common label as 0.5103. Bold (maroon) numbers highlight best values in the set. LightGBM produces best scores, followed by XGBoost. NeuralNet and DecisionTree rank third and are statistically indistinguishable at $p < 5\%$ (indicated by italics and shading).	77
5.3	Model comparison for binary classification in the conservative setting, ordered by validation score. Baseline is given by prevalence of most common label as 0.6642. Bold (maroon) numbers highlight best values in the set. LightGBM produces best validation score, XGBoost best training scores in cross-validation. The two models are statistically indistinguishable at $p < 5\%$ (indicated by italics and shading). Other GBMs rank seconds and are also statistically indistinguishable.	78

5.4	Model comparison for three-class setting, ordered by validation score. Scores reflect accuracy. Baseline is given by prevalence of most common label as 0.4897. Bold (maroon) numbers highlight best values in the set. LightGBM produces best scores, followed by other GBM implementations.	79
5.5	Comparison of several GBM implementations. Bold (maroon) numbers highlight best values in a setting. LightGBM consistently produces best CV and validation scores It is also significantly faster than competing implementations. Models with CV scores that are statistically indistinguishable from each other at $p < 5\%$ are highlighted with italics and shading.	81
5.6	Penalties $c(prediction, pipeline)$ for different types of (mis-)classifications and settings outlined in section 5.2. When the sole purpose of a new engine is to off-load the remainder of the pipeline, TNs produce no gain, FNs no harm (they are thus associated with low penalties). TPs are the desired outcome. FPs are to be avoided. Higher FP aversion in the conservative setting is reflected by a higher penalty. The three-class setting inherits and combines penalties from binary scenarios. Relative class weight is determined as the sum over three-class penalties: $w_- = 3 + 6 + 11 = 20$, $w_? = 4 + 2 + 7 = 13$, $w_+ = 5 + 3 + 1 = 9$. .	89

5.7	Assume a TP rate of 15% at the aggressive stance is required to justify introducing a new model. Let us further assume the cost of dealing with an FP is equivalent to the savings produced by 20 TP convictions (suggesting a value of $\lambda = 20$). The table rows contain $TCR(\lambda = 20)$ as well as FP and TP rates for GBMs trained in the aggressive classification and the regression formulation over a range of multiplier l for non-positive class-weights. Higher weight of non-positive training data reduces convictions. Grayed out rows represent models that do not satisfy the required TPR or result in $TCR < 1$ and can thus be eliminated. Bold, shaded values highlight the best model (highest feasible TCR), achieved in the regression setting trained with $l = 10$	92
5.8	Features relevant to examples in figure 5.9. Prevalence is simply the ratio of the feature being active in the dataset. Correlation and precision are computed against the pipeline spam verdict (aggressive formulation with threshold score 0.5).	104
6.1	Example of one-hot encoded data. This format is used in our experiments.	122
6.2	Mathematical symbols and their meaning in this document.	126

LIST OF FIGURES

Figure		Page
3.1	<p>$[+, -]$ represent (hidden) ground truth that is not known for our dataset. The tiled triangle covers positive pipeline verdicts (areas A,B are TPs, areas E,F are FPs). The grey oval shows positive SDR predictions. Area C contains novel TPs that we will have to penalize as FPs together with area D. Area F contains novel TNs that will be interpreted jointly with area A as FNs.</p>	35
3.2	<p>Distribution of pipeline verdicts and scores after stratification and de-duplication. Positive and suspect spam (+ and ?) together have similar volume as messages not detected as spam (−). Scores are concentrated at the upper and lower bound with noteworthy steps at 0.5 and 0.9, marking transition between verdicts. The spike around 0.7 indicates score pinning for (not further specified) conditions. . . .</p>	40

- 4.1 The full email volume V_1 comprising both legitimate messages (smileys) and spam (bold "S") enters the pipeline from the left. A first slice of engines, E_{pre} , with combined B_{pre}, P_{pre} filters out a subset of $V_1 B_{pre}$ presumed bad messages. A conviction is considered a final action and identified spam is not further processed. Reduced volume $V_{post} = (1 - B_{pre}) V_1$ is passed on to the proposed new engine E_{new} , which in turn filters out $V_{post} B_{new}$ messages over P_{new} . It emits further reduced volume $V_{new} = V_{post}^* = (1 - B_{new}) V_{post}$ for final processing by E_{post} 45
- 4.2 The left pie-chart shows convictions of the existing pipeline. Grey, homogeneous areas indicate mail that was allowed to pass the pipeline and was delivered (alleged legitimate). The several patterned areas highlight messages identified as spam: Dotted areas are convictions from E_{pre} , vertical stripes are associated with E_{post} . Right illustrates how a newly added engine E_{new} (horizontal stripes) influences the pipeline verdicts. $B_{new} = B_{new+} + B_{overlap}$ is composed of novel convictions and overlap with the existing verdicts. B_{post+} are additional conviction that E_{post} can issue due to features exposed by E_{new} 49

4.3	Utility distribution over ROC-space for different parameter settings, assuming equal size of the positive and negative class ($Neg = Pos$). 4.3a considers block rate only, 4.3b shows utility when only considering TCR at equal penalties for FPs and TPs. 4.3c and 4.3d show utility combining block rate and TCR and illustrate the effect of increasing FP penalty λ . We note that utility values have been truncated at -1 for better visualization but range down to -2.25 in 4.3d)	57
4.4	ROC-curve for a GBM regression model and the lower bounds on TPR induced by $\lambda = 5$ and $B_{min} = 5\%$. Only the upper-left quadrant, above the dotted and dashed lines, is considered worthwhile. The blue cross indicates the point on the ROC-curve inside of the worthwhile section, for which the utility function reaches its maximum.	59

4.5	Utility function for several parameter values: Colorful bold sections highlight worthwhile discrimination thresholds. Marks identify operating points as local maxima of the utility function limited to worthwhile thresholds. Lowering B_{min} extends the worthwhile range towards the right (higher decision thresholds). Decreasing λ extends the lower bound of the worthwhile range. For $\eta = 1$ (solid line), the operating point is always associated with maximum (worthwhile) conviction rate. As η decreases, λ controls the utility penalty incurred by FPs. Higher FP penalty requires higher discrimination thresholds (dashed line), while lower values result in lower thresholds (dotted line).	61
5.1	Comparison of existing pipeline verdicts and scores and model predictions in native classification and regression-classification approaches. Native classification pushes distribution towards the boundaries with more-than-original negative and significantly fewer suspect verdicts. Regression-classification approach shown in orange does the opposite and concentrates many samples in the suspect range.	82
5.2	Comparison of native classification and regression-classification models.	84
5.3	Mean average error in training and validation both decrease with tree depth. Overfitting would be indicated by increasing validation errors after a certain point. This is not observed here.	85

5.4	FP and TP rates for a range of multiplier l applied to base weights of negative and suspect classes as defined in table 5.6. Higher values of multiplier l result in models with lower FP and TP rates. Solid lines represent the FP and TP rate achieved with $l = 500$ (downward triangle) on the very left, and its development for lower values ending with $l = 0$ (no differentiated class weight, upward triangle) at the right for native classification GBMs. Dashed lines show the same for regression-classification models operating at original pipeline thresholds (spam at ≥ 0.9 for conservative and ≥ 0.5 for aggressive). Thin dotted lines show full ROC-curves for regression-classification models. Higher values of multiplier l are located at very steep sections of the ROC curve indicating a high TPR/FPR ratio (though at low conviction rates overall). Modifying the discrimination threshold even slightly leads to drastic changes in TPR. For lower values of multiplier l the ROC-curves are flatter and tuning the discrimination threshold has lesser effect on the TPR/FPR ratio. Overall, native classification models are generally more aggressive and produce both, higher TPR and FPR for any given value of multiplier l	90
-----	---	----

5.5	Heatmaps illustrate the utility landscape over a grid of values for λ , and l for select utility trade-off parameters η . Utility is computed for regression-classification in the aggressive setting. Assigning higher weight to positive classes at fixed FP penalty λ yields utility gains. Red dots mark highest utility for each value of λ	93
5.6	Increasing penalty γ leads to faster and cheaper models. Higher values of α reduce prediction speed but do not significantly limit the number of features used by the GBM (and thus do not reduce feature cost). Accuracy is negatively impacted by both types of penalties. . .	95
5.7	Utility landscapes by CEGB models with three different business settings. Red circles indicate the global maximum. Depending on the parameters of the utility function, CEGB can yield improvements for specific penalties (left). When that is not the case (middle), reduced cost might offset the loss of model quality (right). Distinct color pallet in the left panel is used to indicate different range of utility values compared to the other two.	97

5.8	Maximum utility (and corresponding TPR and FPR) is computed over a grid of parameter values η, λ for a model trained on the full 252 features and one that is limited to 125 features from the envelope phase. Models use regression-classification in the aggressive setting. Distinct color pallet in the upper panels is used to indicate different range of utility values. 5.8a shows raw utility of the model trained on full data for reference. 5.8b shows Δ utility computed as utility achieved by the envelope model, subtracted from corresponding values of the full model. Higher value indicates higher utility loss. When throughput is the main concern (high values of η) and FP penalty is low, the envelope model has the same utility as the full model, but at drastically reduced cost. A secondary region of low utility loss is found around $\eta = 0.6, \lambda = 4$. Depending on the cost savings, limiting the model to the envelope phase features might be worthwhile. Remaining graphs indicate the magnitude by which the FPR and TPR can be improved based on additional features.	100
5.9	Red dots indicate the feature was active, blue indicates inactivity. Details for relevant features are summarized in table 5.8.	103
5.10	Correlation between various feature importance measures computed over a LightGBM regression-classification model using the aggressive setting, and feature characteristics.	105

5.11 Characteristics for regression-classification models in aggressive setting trained with top-k features (x-axis) selected according to various ranking methods. Graphs on the left provide an overview. 5.11a shows R^2 scores for regression models. 5.11c shows utility for $\eta = 0.2, \lambda = 8$ on the same models used for regression-classification in the aggressive setting. Graphs on the right are zoomed in on the same regression-classification models with few features. 5.11b shows precision while 5.11d illustrates recall. Rankings from highly correlated TreeSHAP, split-, and gain importance produce similar models, with split gain generally trailing behind the other two. Ranking by feature prevalence seems competitive when the goal is a model with high true positive rate. Feature power (*pathPow*) produces highly precise models so is a good choice when FP prevention is key. Ranking according to spam precision produces highly precise models. These methods, while tempting for their simplicity, are susceptible to selecting highly correlated features which limits their practical use (see figure 5.12). . 109

5.12	For each method (x-axis), we investigate two noise-scenarios and two model variants. Graphs on top show results for regular GBMs, bottom for CEGB variants. Left panels illustrate a scenario where 45 increasingly noisy copies (noise level ranging from 1% to 45%) of only the top ranked feature for each variable importance method are introduced. Right shows results for more subtle noise: For top 5 features an exact logical negation is added. Then, 4 copies with noise ranging from 1% to 4% are created for both, top 5 features, and their negations. This again results in 45 dummy features in total. GBMs are retrained on the noisy dataset and features ranked again. y-axis indicates how many of the 45 noisy dummies get ranked among the top 50. Lower values indicate methods that are more resilient against the respective type of disturbance.	113
------	---	-----

Chapter 1

Introduction

1.1 Spam Detection Pipeline Optimization

Large organizations today see massive volume of email traffic which includes a significant fraction spam. While estimates vary, there is wide consensus that spam and other abusive email constitute the majority of email messages. For example, a report by the Messaging, Malware and Mobile Anti-Abuse Working Group (M3AAWG) states that abusive email accounted for between 87% and 90% of messages in their study [45]. Detecting and convicting spam plays an important role in managing overall productivity and limiting risk of significant operational disruption and financial liability that can be caused by phishing, fraud, and malware [9, 27].

This has motivated a multitude of anti-spam techniques, focusing on different parts of spam delivery, and aggregated via various system designs [12, 16, 29, 46]. The

most efficient method is realized by matching the sender IP against lists of IPs known to disseminate spam. While coarse, this method allows rejecting large swaths of messages, especially spam originating from botnets [31], at the cost of a single lookup per connection. Sender Domain Reputation (SDR), discussed in depth in section 2.2, remains efficient and offers additional granularity [61]. Spam detection based on full email content, see section 2.1 for details, facilitates the most complex detection mechanisms. Bayesian classifiers are among the first machine-learning approaches proposed for email [24] and remain competitive for text-based spam detection [19]. Support Vector Machines (SVM) have been successfully deployed to enhance Bayesian methods [21] and are also used independently [66]. Clustering techniques, for example using locality sensitive hashing (LSH), allow tracking of campaigns and facilitate propagating an established verdict to similar messages [2]. Most recently, deep learning techniques in context of anti-spam are actively investigated by the research community [40, 54].

High-fidelity solutions for spam detection using the full message content, when employed standalone, can become bottlenecks for throughput of email traffic due to their high computational cost [25]. Increased efficiency can be achieved, among other approaches, by combining several spam detection methods into a pipeline [53]. Fast approaches are executed first to convict certain types of easily detectable spam at very low computational cost. This preempts execution of computationally more costly methods. Models further down in the pipeline can then be trained on

increasingly challenging messages using ever more elaborate techniques.

Introducing a lightweight model is not free however. The new classifier has to be developed and maintained, features have to be created or acquired, and there is a risk of introducing FPs. When considering the addition of a new classifier to a detection pipeline, we have to ensure reasonable improvement on its overall and multifaceted effects such as pipeline throughput (efficiency), the quality of prediction (efficacy as a weighted form of accuracy or error rate), and cost-per-conviction. Desired effects often contradict each other; lower cost and complexity usually result in lower efficacy. An appropriate trade-off should reflect application or business-driven design goals.

In keeping cost low, one of the most obvious ways is to reduce the number of features that are being used. Selecting and training computationally efficient models and limiting model complexity is another. Lastly, depending on the average time it takes for a proposed addition to the pipeline to classify a message, the place at which it is inserted into the pipeline can be optimized.

1.2 Proposed Approach and Contributions

In this research, we develop and evaluate a method that can be used to improve an existing spam detection system in pipeline form by prepending a low-fidelity classifier. To define and quantify pipeline improvement, we formulate a utility func-

tion that can consider and apply differentiated weights to efficacy and efficiency constraints outlined above.

We also design and create a real-world dataset, consisting of features pertaining to domains used by a sender. These features can be acquired early on in the message transmission process and thus lend themselves to efficient detection.

In absence of independent ground truth for the email messages reflected in our dataset, the existing pipeline verdict established by high-fidelity models serves as the classification label that is to be approximated.

We train and evaluate several commonly used ML models to establish an overview and a baseline of how well existing pipeline verdicts can be approximated using low-fidelity features. We then focus on Gradient Boosting Machines (GBMs) [23] as they yield best predictions at competitive speed. We employ decision trees as base learners in GBMs for their inherent interpretability and because we wish to exploit existing pools of feature interpretation measures to further enhance our model’s explainability.

We proceed with mapping the effect of business constraints such as FP avoidance and efficiency and cost considerations on the speed and quality of GBM predictions. Effects are quantified using our utility function such that a practitioner can make informed decisions when deciding on the best solution given specific efficiency, efficacy, and cost requirements.

Furthermore, we investigated efficacy of various model interpretation methods for trained spam detection models. Specifically, we broaden applicability of feature-power measures [22] developed for Random Forest (RF) [14] classifiers to Gradient Boosting Machines (GBMs) [23] used in either a classification or regression setting. We conclude with exploring how model interpretation methods can be used to improve ground-truth and review their viability for feature selection.

Source code of methods used in this thesis and the dataset are made available to the research community.¹

¹<https://github.com/JakobDohrmann/PredictionPipelineMS2020>

1.3 Thesis Organization

Following this introduction capturing background and main goals, the remainder of this thesis is organized in the following chapters:

- Chapter 2 reviews the literature and provides introductions to the methods employed in this study.
- Chapter 3 describes a real-world dataset that serves as the basis for our experiments.
- Chapter 4 introduces the mathematical frameworks we propose and describes the various aspects of pipeline optimization strategies.
- Chapter 5 addresses a detailed investigation of how our theoretical considerations are applied to the dataset and presents the experimental results we have obtained.
- Chapter 6 discusses our findings, their significance and proposed areas of future study.

Chapter 2

Background and Prior Work

2.1 Spam Detection

Modern spam detection systems employ and combine various techniques to render a verdict, such as positive or suspect spam, on email messages in order to protect recipients from spam and email-based threats.

While implementation details are well-guarded in the industry, some general statements can be made about the nature of anti-spam pipelines. Increasing information about a message is available during the transmission process. In the beginning, only the sending IP is known. As the network connection is established, additional details about the sending system can be gathered (*i.e.*, using fingerprinting techniques). The SMTP conversation can provide additional insight into sending entities. The

message itself then contains headers and content that can be further analyzed. Finally, user interaction and feedback can be considered. Volumetric information can be applied at any stage to try and detect abnormal behavior. Clustering and believe propagation can increase the impact of a conviction. Engines operating on earlier stages of the message transmission have fewer features to work with (and generally operate faster) while later engines have more context, leading to more authoritative predictions at the penalty of increased computational cost. Earlier engines can pass their features, and additional insights they may have generate, on to later engines. Reversely, verdicts from later engines can be aggregated over entities available at earlier stages (such as IPs, domains or hashes) and back-propagated to facilitate reputation-based convictions more efficiently.

Widely used open-source solutions such as SpamAssassin [7] and Rspamd [58] evaluate a multitude of “rules” that reflect the presence of spam indicators. The term “rule” is used in a very generic sense here; examples of what a rule might indicate include a regular expression match on the email content, the presence of a URI in the email on a block-list, or the output of a machine learning model such as a Bayesian classifier [24].

These abstract rules are then associated with a spam score. In SpamAssassin, scores can be set manually [8], or assigned by a neural network trained with error back propagation (Perceptron) over a corpus of emails labeled as either ham (legitimate mail) or spam [5].

Finally, to determine whether a given message is spam, the scores of all rules that are triggered by the message are summed up. Two thresholds are commonly defined. If the lower is surpassed by the summed scores, the message will be considered suspect. If the higher is also reached, the message is considered spam. This approach can generate highly accurate results due to the very rich features (full email content and context), and the possibility to evaluate and combine an ensemble of independent classifiers into a final verdict.

The field of spam detection operates under application-specific constraints that are reflected in the evaluation of employed models. Email recipients behind an anti-spam system can often identify and manually delete false negatives (FNs, spam message that remained undetected). There are however extremely destructive email-based threats such as malware or phishing that are often well crafted or targeted and thus also hard to identify. These threats are generally lower volume with the majority of spam being a mere nuisance. On the other hand, false positives (FPs, benign messages being marked as spam) are usually much more cumbersome to detect and fix. Only when a recipient misses an expected message and sifts through deleted or quarantined messages, if they are accessible at all, can a falsely convicted message be salvaged. Business critical messages not reaching their recipients in time can lead to financial loss for a company. Perceived impact of FPs and FNs generally differs between environments and quality measures that can incorporate differentiated penalties for different types of errors are required [4].

Further, spammers are actively avoiding detection and are continuously working to counter any methods designed to thwart their efforts [26, 36, 41, 63, 64]. The implication of such an adversarial setting, and this might go against the intuition of many machine learning practitioners in other fields, is that no training set will ever cover the data a trained model is expected to classify. To maintain efficacy, it is paramount to identify artifacts spammers can't avoid (*e.g.* because they are critical to their business model or an immutable effect of their operations or sending infrastructure). Where that is not sufficient, anti-spam researchers are trained to foresee next steps spammers might take and ensure models are robust against likely mutations and avoidance strategies. Human intuition and domain knowledge then become critical components to the machine learning framework and the way they are trained and operated. To support such efforts, it is essential that researchers are able to interpret existing models and are able to understand correlations between features.

Lastly, when a message is wrongly missed or convicted, it is usually necessary to justify the incident to a message's recipient and its stakeholders. Operators of a spam detection pipeline are often asked to ensure a similar issue will not repeat in the future. To serve such requests, it is desirable to employ models that are easily interpreted and that allow targeted interventions, beyond just adding specific samples to the training set.

2.2 Sender Domain Reputation

Some of the most prevalent techniques to efficiently detect large amounts of spam are based on reputation systems [3, 20, 28, 48, 49, 57]. Reputation can be assigned to various entities in an email. Common reputation systems focus on sender IPs, domains, URIs or even attachments (see [62] for an overview).

Sender IP based reputation systems can be realized with a single lookup at TCP connection time before the email transmission even starts. They are thus extremely efficient. At the same time, the sender IP is a very coarse way of assigning reputation. A single IP may host many independent senders. Such shared IPs, commonly observed in cloud email, webmail, email service providers, cannot be blocked without adverse effects to all the other senders using the same IP. Increased adoption of IPv6 with its huge address space also call for supplemental approaches [34].

Domains (such as example.org) used by a sender, offer higher granularity and can be tracked across hosting IPs. Reputation carrying entities should be long-lived [51]. Domains are usually of higher value to the sender compared to the sending IP and thus less likely to be abandoned. Domains are costlier to obtain since they have to be registered. Even free or very cheap domains require some level of interaction while IPs, especially for spam sent via botnets with their huge and changing pool of infected machines, can be more easily exchanged. Domains may also carry intrinsic value such as brand-recognition or association with a legitimate entity. At the same

time, sender domains are available early on in the email transmission process and thus lend themselves to efficient detection in an anti-spam pipeline.

For the purpose of this project, we distinguish two phases during the email transmission process that allow extraction of sender domains and related features present in our dataset. Each phase offers specific intervention strategies:

Envelope phase The mail-gateway tasked with spam detection receives a TCP *syn* package from a sending server. A DNS PTR lookup on the sender IP is performed to establish a first sender domain. We refer to this context as *rdns* (reverse DNS) in our dataset. Once the connection is established, the sender initiates a SMTP session [33] and issues several commands in order to transmit a message. The SMTP HELO/EHLO and MAIL FROM commands contain a second set of sender domains. We refer to these contexts as *helo*, and *envFrom* respectively. A Sender Policy Framework (SPF) [32] lookup can be performed at this stage to verify if the sending IP is sanctioned to deliver mail on behalf of the *helo* and/or *envFrom* domains. A blocking verdict during this phase can be enforced before the actual message is transmitted.

Data phase If the first phase didn't allow to render a blocking verdict, the receiving mail-gateway issues the SMTP DATA command. The

sender then transmits the actual message. Email headers contain a third set of several domains associated with the sender [50]. The contexts here are labeled *hdrFrom*, *hdrReplyto*, and *listunsub* referring to sender domains extracted from the From, Reply-To and List-Unsubscribe headers respectively. In addition to these new domains, given that the full email is now available, DomainKeys Identified Mail (DKIM) [18] signatures, and Domain-based Message Authentication, Reporting and Conformance (DMARC) [35] can be evaluated and used to establish the verdict. If a blocking verdict is established, the mail-gateway can issue a SMTP QUIT command to indicate that the message was not accepted.

We can roughly distinguish two methods of establishing reputation: 1) State-full approaches identify and track entities over time. They employ real-time feedback loops and volumetric measures to assess behavior and automatically update reputation as information is aggregated [56, 61]. 2) State-less approaches use features available for any entity, even those that have never been observed or can't be authenticated, to try and predict whether associated message can be safely rejected. State-less systems can benefit from state-full inspection if insights from the latter are accessible. A common approach for sharing insights is implemented as block-lists that track known-bad entities. This approach has been applied for a very long time (*e.g.* RFC5782 mentions IP block-lists as early as 1997 [38]).

Most real-world systems combine these two approaches. In this study, we focus on the latter state-less approach only. That is, we make predictions using a fixed set of features associated with sender domains, without dynamically adjusting reputation of a domain based on past behavior observed in our dataset.

2.3 Gradient Boosting Machines

2.3.1 Basic Concepts

Predictive learning, sometimes referred to as function estimation, can be formulated as follows: Given labeled data $X = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, approximate function $\mathcal{F}(\mathbf{x}) = y$ that maps an input (*i.e.*, a feature vector) $\mathbf{x}$ onto the appropriate response y , such that the expected value E of a loss-function L is minimized:

$$\hat{F} = \operatorname{argmin}_{F^*} E_{y,\mathbf{x}} L(\mathcal{F}(\mathbf{x}), F^*(\mathbf{x})) \quad (2.1)$$

Gradient Boosting Machines (GBMs), as described by Friedman in 2001 [23], present a framework for predictive learning that is applicable to a variety of loss-functions, and can employ various base-learners (weak models) to produce an ensemble.

Output of GBMs is presented as an additive expansion of the form:

$$F(\mathbf{x}, b_m, \mathbf{a}_m) = f_0 + \sum_{m=1}^M b_m h_m(\mathbf{x}, \mathbf{a}_m) \quad (2.2)$$

$f_0 = \operatorname{argmin}_{\rho} \sum_{i=1}^N L(y_i, \rho)$ is an initial "guess" which is produced based solely on the labels presented in the training data without using the corresponding inputs. Its calculation depends on the exact GBM formulation. For example, if squared-error loss is applied, $f_0 = \bar{y}$ is simply the average over training labels.

Remaining summands $b_m * h_m$ are "corrections" with a specific sign and magnitude that shift the ensemble response away from the naïve output f_0 .

Function estimates in each summand h_m represent base-learners. In practice, they are usually taken from a single set of parameterized functions where $\mathbf{a}$ is the set of parameters defining a specific instance of the base-learner. b_m is the weight associated with the base-learner. Its value is determined via line-search to minimize expected loss over training data in each boost.

The GBM expansion is developed in a forward stage-wise manner:

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + b_m h_m(\mathbf{x}, \mathbf{a}_m) \quad (2.3)$$

The process starts with an initial model $h_1(\mathbf{x}, \mathbf{a}_1)$. In each iteration m , referred to as a boost, the new term $b_m h_m(\mathbf{x}, \mathbf{a}_m)$ is optimized to improve the shortcomings of

the ensemble $F_{m-1}(\mathbf{x})$ up to that point.

In early boosting algorithms, the improvement in each step is achieved by weighing the training data according to their specific residuals $y_i - F(\mathbf{x}_i)$. The leap to GBMs and their support for any monotonic, continuous differentiable loss-function comes from the insight that residuals can be seen as a special case of a negative gradient: When the loss-function is $L(y, F) = \frac{1}{2} (y - F(\mathbf{x}))^2$ (squared error), each component of the negative gradient over the training data can be expressed as $-g(\mathbf{x}_i) = -\frac{\partial}{\partial F(\mathbf{x}_i)} \frac{1}{2} (y - F(\mathbf{x}_i))^2 = y - F(\mathbf{x}_i)$.

The function estimate of GBM is improved in a greedy fashion, applying gradient descent in function space: In each boost, h is oriented along the negative gradient and an appropriate step-size b is determined via line search. Given constraints imposed by the class of base-learners and due to finite data, orientation towards the negative gradient generally needs to be approximated. This is achieved by using the data-based analogue to the smooth negative gradients $-g_m(\mathbf{x}_i) = -\frac{\partial L(y, F_{m-1}(\mathbf{x}_i))}{\partial F_{m-1}(\mathbf{x}_i)}$, $i \in [1, N]$ as weights for training data when creating the next base-learner.

Common loss-functions such as squared- or absolute-error and the Huber loss-function for regression, and negative binomial log-likelihood [23] or exponential Ada-loss [52] are discussed in detail for specific implementations of GBM algorithms. In general, any monotonic, continuously differentiable loss-functions tailored towards a dataset and the biases a given problem might impose, can be utilized in GBMs.

For regression and binary classification, GBMs consist of a single ensemble. In the regression setting, GBM output directly aims to approximate the response:

$$\hat{y}(\mathbf{x}) = F_M(\mathbf{x}) \quad (2.4)$$

For binary logistic regression and classification ($c_k \in \{-1, 1\}$), GBM output can be interpreted as a probability estimate for a class label (*e.g.* via log-odds). With $c(c_k, c_l)$ as the cost of predicting label c_k when the correct class is c_l , the prediction $\hat{c}(\mathbf{x})$ becomes:

$$\begin{aligned} p_+(\mathbf{x}) &= \hat{Pr}(y = 1|\mathbf{x}) = \frac{1}{1 + \exp(-2F(\mathbf{x}))} \\ p_-(\mathbf{x}) &= \hat{Pr}(y = -1|\mathbf{x}) = \frac{1}{1 + \exp(2F(\mathbf{x}))} \\ \hat{c}(\mathbf{x}) &= 2 * \mathbb{1}[c(-1, 1)p_+(\mathbf{x}) > c(1, -1)p_-(\mathbf{x})] - 1 \end{aligned} \quad (2.5)$$

Here and in the following, $\mathbb{1}(\cdot)$ is an indicator function that evaluates to one when its argument is true, otherwise to zero.

In a multi-class logistic regressions and classification setting, K trees are trained for each boosting step resulting in K ensembles, one for each class [23]. We note that cost-efficient formulations exist, that allow prediction of K classes using $K - 1$ trees in each boost by defining a "base class" [1, 39]. We'll refer to the output of the k -th ensemble, responsible for determining likelihood of class label c_k as F_k . Similar

to binary classification, the predicted class label is determined as the most likely, weighted according to the cost of a misclassification:

$$p_{c_k}(\mathbf{x}) = \frac{\exp(F_k(\mathbf{x}))}{\sum_{l=1}^K \exp(F_l(\mathbf{x}))} \quad (2.6)$$

$$\hat{c} = c_{\hat{k}}; \hat{k}(\mathbf{x}) = \operatorname{argmin}_{1 \leq k \leq K} \sum_{l=1}^K c(c_k, c_l) p_{c_l}(\mathbf{x})$$

Note that this formulation collapses into equations (2.5) for $K = 2$ by setting $F_1 = -F_2$ and requiring that the penalty (or benefit) of a correct classification is identical for the two classes (*i.e.*, $c(-1, -1) = c(1, 1)$).

2.3.2 Decision Trees as Base-Learners

A variety of based-learners are employed in different formulations. Since decision trees will be the focus of our work, let us review them in more detail.

Let h represent a J -terminal decision tree. It partitions the feature space in J disjoint regions $\{R_j\}_{j=1}^J$, each representing a leaf node associated with a specific response $f(n_j)$. In the following, $f(n_j)$ denotes a function that returns the split variable $f_i \in D$ associated with node n_j or the node value if n_j is a leaf. The parameters $\mathbf{a}$ then specify split variables and thresholds that define the leaves, as well as their corresponding responses. Due to the disjoint nature of leaves, the tree

prediction can be expressed in additive form:

$$h\left(\mathbf{x}, \{f(n_j), R_j\}_{j=1}^J\right) = \sum_{j=1}^J f(n_j) \mathbb{1}(\mathbf{x} \in R_j) \quad (2.7)$$

For a given input $\mathbf{x}$, equation (2.7) thus returns the value associated with the leaf that contains it.

This additive form allows another optimization: Rather than computing a single step-size b_m for the whole tree as outlined in equation (2.2), an appropriate weight can be assigned to each disjoint leaf.

2.3.3 Differences Between GBM and RF

Since we will expand feature interpretation measures developed for RandomForest (RF) to GBM in section 4.3.3 (Feature Power for GBM Classifiers, page 66), we want to highlight some relevant differences between RF and GBM.

Let us first summarize how predictions are established. In the multi-class setting, RF employs multi-class decision trees directly and the prediction can be established by majority vote over a single ensemble. GBMs on the other hand train independent ensembles to predict probability for each class and return the most likely. In regression, RF returns the (unweighted) average over trees, while GBM returns the sum of weighted tree responses.

Further, in RF, trees are independent and identically distributed (*i.i.d.*) samples of a distribution defined by a set of parameters (such as the number of random samples of training data and features that are used). Trees in GBMs in contrast are not *i.i.d.* Each tree is grown deterministically, with no random elements [13]. The training data for each tree is weighted based on the errors made by the ensemble thus far.

We emphasize that once a GBM model is established, trees can be treated as independent algebraically due to the additive formulation of equation (2.2). In other words, given an input, the response of one tree is not influenced by the response of another in a trained model.

2.3.4 Simplified Notation for GBMs

To avoid overfitting and to limit model complexity, regularization techniques are commonly applied during GBM training. At its simplest, the number of base-learners M can be limited. Additionally, a learning rate can be introduced that increasingly limits the impact of base-learners trained at later stages in the ensemble.

As mentioned, various base-learners can be employed, each with their unique set of parameters and associated notations. For the example of decision trees, the step-size b takes the shape of a J -dimensional vector, assigning specific weight to each leaf.

While critical to the GBM, explicitly accounting for such details is not required for

the theoretical considerations in the remainder of this work. For convenience, we simplify equation (2.2) and represent the prediction of a GBM as follows:

$$F(\mathbf{x}) = f_0 + \sum_{m=1}^M r_m(\mathbf{x}) \quad (2.8)$$

In this notation, which will be used in section 4.3 (Expanding Feature Power, page 60), the output of each base-learner and their respective parameters and weights (*i.e.*, scaling factors) are consolidated into $r_m(\mathbf{x})$.

2.3.5 Cost Efficient Gradient Boosting

Peter *et al.* (2017) outline a method of cost efficient gradient boosting (CEGB) that is relevant to our considerations [47]. The authors propose three distinct penalties, considered during GBM training.

First, CEGB allows to assign a penalty for the first time a feature is used in a specific data point. This formulation is suitable when features are computed on the fly only when needed. While some of the features in the dataset used in this study might be of this nature (*e.g.* some computation that is triggered only when needed), we can't know if and when it is appropriate and have not further investigated this penalty.

Secondly, a penalty can be assigned for the first time a feature is used during training of the ensemble. Once a feature has been used, it can be reused cost-free in

subsequent splits and boosts. This formulation covers features that have to be acquired globally, irrespective of whether they are used for any specific data. Examples include deploying lists of known malicious domains. Such lists are available commercially or can be generated based on in-house intelligence. Another example is some function or lookup that is performed for each incoming data a priori, irrespective of the actual decision path. This formulation seems most appropriate to our setting. Once cost is defined for each feature, the penalty can be scaled by a multiplication parameter that we will refer to as γ .

Lastly, a penalty α can be assigned to each split, encouraging shorter decision paths on average. This directly impacts complexity and prediction time, stated goals of our optimizations.

2.4 Model Interpretation

The literature on tree-based models describes various measures aimed at explaining full models or single predictions, and to assess the impact of specific features and feature combinations (*e.g.* [23, 44, 43, 22]). In addition to model interpretation, these measures can also be the basis for feature selection. Relevant methods, used in the remainder of this work, are reviewed in the following sections.

2.4.1 Feature Importance Based on Gain and Split Count

Originally proposed for decision trees, Breiman *et al.* (1984) describe a measure of relative feature influence on variations in model predictions [15]. It is based on improvements of squared error (gain) during the training phase, summed over non-terminal nodes (splits). Per definition, this measure requires data obtained during model training and can thus not be computed on a model instance unless relevant data has been preserved.

Given a J -terminal regression tree T with nodes n_i , we summarize the non-terminal nodes in set $\{n_i | n_i \notin l(T)\}$, where $l(T)$ returns the set of leaf nodes in T . Further, let $f(n_i)$ denote the feature used in node n_i to produce a split, and $\hat{i}_{n_i}^2$ quantify the reduction of squared error induced by the split. Influence of feature f_j is approximated as follows:

$$\hat{I}_{gain, f_j}(T) = \sum_{\{n_i | n_i \notin l(T)\}} \hat{i}_{n_i}^2 \mathbb{1}(f(n_i) = f_j) \quad (2.9)$$

Note that the original formulation uses $\hat{I}^2$ to denote this measure. For consistency in this document, we simply use $\hat{I}$.

A simpler measure is given by the number of nodes that use feature f_j to produce a split, irrespective of gain:

$$\hat{I}_{split, f_j}(T) = \sum_{\{n_i | n_i \notin l(T)\}} \mathbb{1}(f(n_i) = f_j) \quad (2.10)$$

These concepts are applied in GBMs by simply averaging over all trees in the ensemble [23]. Given a GBM composed of M decision trees, $\{T_m\}_1^M$, importance of feature f_j is computed as:

$$\hat{I}_{f_j} = \frac{1}{M} \sum_{m=1}^M \hat{I}_{f_j}(T_m) \quad (2.11)$$

In this formula, $\hat{I}_{f_j}(T)$ can take the form of any tree-based feature importance measure, including equations (2.9) and (2.10).

In a K -class setting, with $T_{m,k}$ representing the m -th tree in the function estimate for class k , importance can be determined for each class separately as outlined above: $\hat{I}_{f_j,k} = \frac{1}{M} \sum_{m=1}^M \hat{I}_{f_j}(T_{m,k})$. An aggregate measure of feature importance is then established by averaging over all classes:

$$\hat{I}_{f_j} = \frac{1}{K} \sum_{k=1}^K \hat{I}_{f_j,k} \quad (2.12)$$

2.4.2 Shapley values

In cooperative game theory, an abstract game is defined by a characteristic function $v(S)$ assigning a specific value to coalition $S \in U$. The coalition space $U = 2^D$ is the power set of all players D . Shapley value $\phi_{f_j}[v]$ indicates the power of a given player f_j in game v by evaluating the characteristic function over all coalitions, and

producing a weighted difference of coalitions including and excluding f_j [55]:

$$\begin{aligned}
\phi_{f_j}[v] &= \sum_{S \in U; f_j \in S} \frac{|S-1|!(|D|-|S|)!}{|D|!} v(S) \\
&\quad - \sum_{S \in U; f_j \notin S} \frac{|S|!(|D|-|S|-1)!}{|D|!} v(S) \\
&= \sum_{S \in 2^{D \setminus \{f_j\}}} \frac{|S|!(|D|-|S|-1)!}{|D|!} [v(S \cup \{f_j\}) - v(S)] \tag{2.13}
\end{aligned}$$

The Shapley value has been proven to be the only such measure of a player's power satisfying a set of desirable properties, namely efficiency, linearity, symmetry, and zero value for a null player [55].

Since linearity will be used later on, we state its definition as it applies here:

Definition 2.1. Let v and w be the characteristic functions of two coalition games and $a \in \mathbb{R}$ an arbitrary real number. ϕ is linear if and only if the following statements are true:

1. $\phi_{f_j}(v + w) = \phi_{f_j}(v) + \phi_{f_j}(w), \forall f_j \in D.$
2. $\phi_{f_j}(av) = a\phi_{f_j}(v), \forall f_j \in D.$

2.4.3 SHAP and TreeSHAP

SHAP (SHapley Additive exPlanation) was introduced by Lundberg and Lee in 2017 [44] as a way to explain a given model prediction. The authors develop an additive feature attribution method where a given prediction is represented as a linear function of binary variables $z' \in \{0, 1\}^{|D|}$, weighted with feature attribution values ϕ_{f_j} :

$$g(z') = \phi_0 + \sum_{j=1}^{|D|} \phi_{f_j} z'_j \quad (2.14)$$

Here, D is the set of input features, z'_j indicates whether feature f_j is specified or unknown in $|D|$ -dimensional model input $\mathbf{x}$.

Feature attribution values for a given input and model prediction $F(\mathbf{x})$ are computed as Shapley values as defined in equation (2.13), where features take the role of players, and the characteristic function, or value of a coalition, is determined as $F_S(\mathbf{x})$, the expected output of the model, conditioned on a subset of input features:

$$\phi_{shap, f_j}(\mathbf{x}) = \sum_{S \subseteq D \setminus \{f_j\}} \frac{|S|!(|D| - |S| - 1)!}{|D|!} [F_{S \cup \{f_j\}}(\mathbf{x}) - F_S(\mathbf{x})] \quad (2.15)$$

The concept was soon extended to interaction effects, and a polynomial time algorithm for computing SHAP values for decision trees (TreeSHAP) was introduced [43]. With w representing the fraction of training samples associated with a given

node, in TreeSHAP, $F_S(\mathbf{x})$ is estimated by traversing the tree(s) and returning:

- $F(\mathbf{x}) * w$, if the features present in S allow determination of a specific leaf.
- Otherwise a weighted average is computed over all leaves that may be reached conditional on absent features. Weights are assigned to subtrees resulting from splits on features not present in S proportional to covered training samples and aggregated for all leaf nodes terminating such subtrees.

Based on the linearity of Shapley values per definition 2.1, the authors assert the following corollary:

Corollary 2.1. *Since the Shapley value of a sum of functions is identical to the sum of Shapley values over the distinct functions, SHAP values for an ensemble of trees (such as RF or GBM) can be computed by simple summation over all trees.*

While extremely beneficial to understand a specific model output, local (per-prediction) feature attribution values do not indicate how important a feature is in general (say over a dataset X). For a more comprehensive view, feature attribution values are aggregated over many inputs and visualized in SHAP summary plots [42]. Those provide detailed information about the prevalence, direction, and magnitude in which features affect model predictions. It can be beneficial to collapse such rich information into a single value (*e.g.* to create a ranking of features as the basis

of feature selection). This is done by summing absolute magnitudes over individual inputs (note that we use capitalized Φ for aggregated values throughout this document):

$$\Phi_{shap,f_j} = \sum_{\mathbf{x} \in X} abs(\phi_{shap,f_j}(\mathbf{x})) \quad (2.16)$$

2.4.4 Feature Power

Fotion and Okada (2018) introduced a model-instance specific approach to determining feature and coalition power for classification trees and Random Forest classifiers [22]. Their method uses only the structure of a trained model and does not rely on information from training such as node-wise impurity or prevalence values, out-of-bag samples, or a dataset, to create predictions. Similar to TreeSHAP, feature- and coalition power are based on Shapley values pertaining to a specific class label c_k that are estimated and efficiently computed by limiting the coalition space to feature subsets that are actually present in a model. The authors introduced three definitions of a coalition, leading to three distinct sets of measures. For each, computation is based on Shapley values as per equation (2.13), with distinct substitutions for the coalition space U (and consequently the size of a coalition S), and the characteristic function v as outlined below. In the following, let $p = (n_i, n_j)$ be a (sub-)path starting at node n_i and ending at node n_j . $d(p)$ specifies the number of edges traversed in path p , the number of nodes constituting the path is thus $d(p) + 1$.

Formulation 1: Path Iteration (*pathPow*)

The coalition space is limited to existing root-to-leaf paths in tree T . With $l(T)$ returning the set of leaf nodes in T and n_0 the root: $U = \{(n_0, n_i) | n_i \in l(T)\}$, $S = d(p)$, and $v_{c_k}(p) = \mathbb{1}(f(n_i) = c_k)$. Further, let U_{f_j} denote the subset of coalitions (paths) where at least one node is associated with feature f_j (i.e., $\exists n_i \in p; f(n_i) = f_j$). Feature power is then computed as:

$$\begin{aligned} \phi_{pow, f_j}[v_{c_k}](T) &= \sum_{p \in U_{f_j}} \frac{(d(p) - 1)! (|D| - d(p))!}{|D|!} v_{c_k}(p) \\ &\quad - \sum_{p \in U - U_{f_j}} \frac{d(p)! (|D| - d(p) - 1)!}{|D|!} v_{c_k}(p) \end{aligned} \tag{2.17}$$

Formulation 2: Cumulative Node Iteration (*cumNodePow*)

Here, the coalition space is defined as root-originating paths ending at an internal node: $U = \{(n_0, n_i) \in T | n_i \notin l(T)\}$. The characteristic function of such a sub-path is given by the ratio of leaves in sub-tree t'_{n_i} rooted in n_i that correspond to c_k . With $L(T, c_k) = \sum_{n_i \in l(T)} \mathbb{1}[f(n_i) = c_k]$ counting the number of leaves associated with c_k in tree T , the characteristic function becomes $v_{c_k}(p) = \frac{L(t'_{n_i}, c_k)}{\sum_{l=1}^K L(t'_{n_i}, c_l)}$. With $p[n_0 : n_{f_j}]$ specifying the sub-path ending in n_{f_j} , the first node from the root at

which feature f_j is used as the split variable, feature power is defined as:

$$\begin{aligned} \phi_{pow, f_j}[v_{c_k}](T) = & \sum_{p \in U_{f_j}} \frac{d(p[n_0 : n_{f_j}])! (|D| - d(p[n_0 : n_{f_j}]) - 1)!}{|D|!} v_{c_k}(p) \\ & - \sum_{p \in U - U_{f_j}} \frac{(d(p) + 1)! (|D| - d(p) - 2)!}{|D|!} v_{c_k}(p) \end{aligned} \quad (2.18)$$

Formulation 3: Strict Node Iteration (*strictNodePow*)

Finally, to minimize complexity and avoid over-counting nodes at low depths, in this last formulation the coalition space is limited to single nodes: $U = \{(n_i, n_i) \in T | n_i \text{ is an internal node in } T\}$. The characteristic function is unchanged from the *cumNodePow* formulation. Feature power then becomes:

$$\begin{aligned} \phi_{pow, f_j}[v_{c_k}](T) = & \sum_{p=(n_i, n_i) \in U_{f_j}} \frac{d((n_0, n_i))! (|D| - d((n_0, n_i)) - 1)!}{|D|!} v_{c_k}(p) \\ & - \sum_{p=(n_i, n_i) \in U - U_{f_j}} \frac{(d((n_0, n_i)) + 1)! (|D| - d((n_0, n_i)) - 2)!}{|D|!} v_{c_k}(p) \end{aligned} \quad (2.19)$$

Feature Power Aggregation

Feature power is aggregated over classification trees in a Random Forest as the average of feature power values computed for each tree:

$$\Phi_{pow,f_j}[v_{c_k}] = \frac{1}{M} \sum_{m=1}^M \phi_{pow,f_j}[v_{c_k}](T_m) \quad (2.20)$$

Similarly, aggregation over classes in a single tree is also achieved by averaging:

$$\Phi_{pow,f_j}(T_m) = \frac{1}{K} \sum_{k=1}^K \phi_{pow,f_j}[v_{c_k}](T_m) \quad (2.21)$$

Finally, aggregation over both classes and the trees in an ensemble is achieved by combining equations (2.20) and (2.21):

$$\Phi_{pow,f_j} = \frac{1}{KM} \sum_{k=1}^K \sum_{m=1}^M \phi_{pow,f_j}[v_{c_k}](T_m) \quad (2.22)$$

2.4.5 Comparison of Model Interpretation Methods

Feature importance based on split gain and count $\hat{I}_{gain,split}$ has been applied to RF [14] and GBM [23] from their inception. While fast to compute, studies have shown weaknesses with respect to consistent model interpretation and alignment with human intuition [43]. Our own experiments highlight that gain and split importance

are also highly susceptible to assigning similar values to strongly correlated features (see experiments following page 108 and figure 5.12).

TreeSHAP addresses consistency issues while remaining computationally efficient [43]. Variable importance assigned by TreeSHAP is strongly correlated with gain and split importance in our experiments (see section 5.6.2). However, susceptibility to noise is greatly reduced. Experiments in section 5.6.3 indicate that TreeSHAP variable rankings are suitable for feature selection and result in fastest convergence to performance achieved on the full feature set.

Feature power [22] is another computationally efficient variable importance measure based on Shapley values. In our experiments, it is most resilient against assigning similar rank to noisy or strongly correlated features (figure 5.12). This property makes it a good choice for feature selection in datasets where that is relevant. Variable importance assigned via the *pathPow* formulation is unique in that it is correlated with feature precision (while other methods are correlated with how often a feature is active). Models trained with subsets of features, selected based on *pathPow* rankings, have highest precision when very few features are used.

2.5 Feature Selection Based on Variable Rankings

Feature selection is a way to reduce model input dimensionality, and with it, model complexity.

Cohen *et al.* develop a feature selection algorithm based on rankings created via multiperturbation Shapley analysis (MSA) [17]. They propose to iteratively either select highly ranked features (forward selection) or excluded low ranked features (backward elimination). They compared eight feature ranking methods and show that backward elimination using MSA-ranked features performs best of an array of datasets.

Sun *et al.* employ the Banzhaf power index [10] to rank features for a RandomForest (RF) classifier to improve upon feature selection methods based on the information gain ratio [60]. They attribute achieved improvements to the ability of voting-theory based indices to consider dependencies between all features when determining importance.

Lee *et al.* investigate feature selection based on information gain in RF for the context of cost-sensitive spam classification [37]. Bian *et al.* further this line of research and consider feature acquisition cost as part of their feature selection and optimization efforts [11].

Chapter 3

Dataset

This research is performed on a dataset representing real-world email traffic passing through a spam detection pipeline. Each data is labeled with a score and verdict that reflects the consolidated output of this existing pipeline. The pipeline consists of a sequence of several state-of-the-art reputation and content-based scanning engines. A message is labeled as spam, if any of the engines in the pipeline issued a spam conviction. While not perfect (some level of FPs and FNs are expected from any classifier), we aim to approximate the pipeline label in absence of actual ground truth with the SDR model. In this setting, we cannot use SDR to increase conviction rate as positive predictions not already covered by the pipeline cannot be distinguished from FPs. Neither can we benefit from additional SDR TNs, those will count as FNs. An overview of the relation between (hidden) ground truth and the existing pipeline and novel SDR labels is given in figure 3.1.

Figure 3.1: $[+, -]$ represent (hidden) ground truth that is not known for our dataset. The tiled triangle covers positive pipeline verdicts (areas A,B are TPs, areas E,F are FPs). The grey oval shows positive SDR predictions. Area C contains novel TPs that we will have to penalize as FPs together with area D. Area F contains novel TNs that will be interpreted jointly with area A as FNs.

Data spans seven days. It is down-sampled and stratified to balance pipeline verdicts. The verdict is either of positive spam +, suspected spam ?, or neutral (*i.e.*, not detected as spam, legitimate mail) -. It is accompanied by a score ranging from 0 to 1, higher values are associated (in an unspecified way) with higher confidence that a message is spam. Verdicts relate to these pipeline scores via the following mapping:

spam verdict "-" $\Leftrightarrow 0 \leq \text{score} < 0.5$

suspect verdict "?" $\Leftrightarrow 0.5 \leq \text{score} < 0.9$

neutral verdict "+" $\Leftrightarrow 0.9 \leq \text{score} \leq 1.0$

The label is accompanied by a vector of categorical features that are either 1 (active) or 0 (inactive). The following groups of features can be distinguished: *dkim*, *dmARC*, and *spf* related features indicate whether the message passed the respective authentication/sanctioning schemes. There are then six groups of features each associated with a sender domain extracted from a specific context (see section 2.2). The features represent a wide variety of indicators pertaining to WHOIS and DNS information, the maturity as a sender (*i.e.*, how long the domain has been known to send email), the presence on block- or allow-lists, or the output of some other arbitrary function.

All feature-names and domains are masked with generic IDs to protect personally identifiable data (PII) and intellectual property (IP) of the users and owners of the underlying anti-spam system. More details and example data are presented in [Appendix A: Example Data](#) (following page 122).

Data is ordered according to time of scan. That is, data further down in the set represent messages that were sent and scanned later in the time window.

3.1 Data Duplication and Implications for Predictions

The dataset contains significant duplication in the feature vectors. A large fraction of email originates from a relatively small number of senders such as large web-mail providers, bulk senders that run marketing campaigns for external brands, or the ever more prevalent cloud email providers that an increasing number of corporations are using to route their email traffic. The actual emails sent from these entities vary widely in their content (and consequently in their labels), while the sender domain related features are largely identical. Legitimate senders are motivated to keep their email clean to ensure deliverability. This is reflected in our data as large clusters of identical or very similar feature vectors together with mostly benign verdicts. The actual distribution of verdicts for a sender reflects how successful they are in weeding out spammers among their customers and preventing bad mail from leaving

their platforms. These senders, while not perfect, cannot be reasonably blocked by SDR; a verdict can only be established based on the content of an actual message.

Given high volume of mail associated with such senders, an FP on a feature vector associated with their platforms can have devastating efficacy impact in the real-world. While we aim to reduce data duplication overall, we need to reflect real-world data distribution to some extent to make FPs on very common input less likely. This is in addition to general FP avoidance strategies such as differentiated class weight in training or increased penalties for FPs in tuning and evaluation.

High prevalence of messages from senders that SDR cannot address is reflected in limited achievable conviction rate of SDR overall. As our experiments demonstrate, classification accuracy on de-duplicated data is only on the order of 65% to 75% (see section 5.4 for details).

Due to stratified down-sampling, the real-world distribution of data is already compromised. Strict de-duplication would completely break any resemblance of the dataset with real-world traffic. As outlined above, we want to retain some semblance of the actual data profile. To that end, the following steps are taken to reduce data duplication:

- Split the data into 20 parts of roughly equal size, maintaining order. Each part then contains data pertaining to a specific block of time.

- De-duplicate records in each of the 20 parts, considering only the verdict and features. Also considering the pipeline score would lead to very little data being removed. This step cuts data size by roughly 90%.
- Recombine the data.

This procedure allows common items to be duplicated (up to 20 times) while keeping all unique records and retaining the temporal ordering.

3.2 Dataset Size and Verdict Distribution

After applying the pre-processing steps outlined above, data consists of 131,973 records, each represented as a 252 dimensional binary feature vector, accompanied by a label and a score. 34,336 (26%) of records are unique considering the verdict and corresponding features.

Figure 3.2 illustrates the distribution of labels (verdict and score). Legitimate messages make up 49% of data, 17.5% represent suspicious email, while the remaining 33.5% consist of spam. Anti-spam systems are often trained with more imbalanced data. A generic way to tackle imbalance is to assign higher weight to less prevalent classes during training. This approach is investigated in section 5.5.1. Specifically in spam detection, it is also common to assign penalties for false predictions that

Figure 3.2: Distribution of pipeline verdicts and scores after stratification and de-duplication. Positive and suspect spam (+ and ?) together have similar volume as messages not detected as spam (-). Scores are concentrated at the upper and lower bound with noteworthy steps at 0.5 and 0.9, marking transition between verdicts. The spike around 0.7 indicates score pinning for (not further specified) conditions.

reflect both, real-world impact of different types of errors, as well as the lack of insight into some classes of messages (see section 4.2.2, *TCR*). We note that low accuracy observed in this work is not due to imbalance, but in line with expectations and reflective of the nature of SDR data.

Chapter 4

Theoretical Considerations

In this chapter we outline a framework for modeling and optimizing spam detection pipelines and expand existing feature power measures [22] to GBMs, both in classification and regression context. These are novel formulations to our knowledge and thus constitute major theoretical contributions of this work.

4.1 Modeling a Spam Detection Pipeline

The anti-spam pipeline can be thought of as a sequence of n engines $E_k, k \in [1, n]$. An engine is an efficacy-producing system. It uses input features, comprised from engine-specific input data (such as message content and headers, IPs, sender domains, etc...) and intelligence relevant to this data (block-lists, regular expressions, etc...), to issue a verdict indicating whether the message can be convicted as spam

or not. Convictions may be appropriately issued for spam (true positive) or falsely assigned to legitimate messages (false positive). If no conviction verdict is established for a message, it is passed on to the next engine. A conviction on the other hand is considered a final action: Convicted messages are not processed further. Each engine thus reduces the number of messages that have to be evaluated by the remainder of the pipeline. Messages that are not convicted in the pipeline by any engine are considered legitimate, and delivered to the intended recipient.

We note that while data contains three labels $(+, ?, -)$, the decision to drop a message or to pass it on to the next engine is by its very nature binary. While irrelevant for our derivations, we refer to experiments in section 5.2 (Types of Predictions, page 72) where several settings for classification and regression are reviewed in detail.

The rate at which an engine E_k issues convictions, over a representative set of messages (or, in practice, a sufficiently large time window), is referred to as its "block rate" B_k :

$$B_k = \frac{TP_k + FP_k}{V_k} \quad (4.1)$$

In equation (4.1), V_k specifies the number of messages engine E_k has to process, while TP_k, FP_k are the number of true and false convictions issued by E_k , respectively.

With V_1 representing the total number of messages entering the pipeline, the number

of messages passed on to engine E_{k+1} is:

$$V_{k+1} = V_k(1 - B_k) = V_1 \prod_{i=1}^k (1 - B_i) \quad (4.2)$$

Each engine E_k will require a certain amount of time $t_{k,i}$ to process a given message i . This comprises computations, as well as delay due to network look-ups, or waiting for other processes. Averaged over a representative set of V_k messages (as is done for B_k), the expected processing time of each engine E_k is denoted as:

$$P_k = \frac{\sum_{i=1}^{V_k} t_{k,i}}{V_k} \quad (4.3)$$

With this terminology, we can define the expected time a pipeline of n engines takes to process a message $P_{pipeline}$ as:

$$P_{pipeline} = P_1 + (1 - B_1)P_2 + \cdots + (1 - B_1)(1 - B_2) * \cdots * (1 - B_{n-1})P_n \quad (4.4)$$

Finally, developing, maintaining, operating, and supporting an engine comes at a certain pecuniary cost C_k . The details of how cost is computed is highly application and business specific.

4.2 Adding an Engine to the Pipeline

When introducing a new engine (with associated, yet initially unknown P_{new} and B_{new}), the pipeline model in equation (4.4) needs to be updated. In some cases, the location of a new engine in the pipeline is constraint by data availability (*e.g.* an engine working on message content cannot run before the message is transmitted). In other cases, the pipeline designed may be asked to identify the optimal order at which engines should be run. Quantitative comparison can be facilitated by the methods outlined below if multiple insertion points are possible. The decision depends on distinct block rates and processing times of the proposed and existing engines respectively. To investigate a possible insertion point, the pipeline is partitioned into a slice of engines that run prior to the proposed addition (with combined P_{pre} and B_{pre}), and the components that run after it (with combined P_{post} and B_{post}). Figure 4.1 provides an illustration.

4.2.1 Excuse Into Cost Feasibility of Pipeline Modification

In real-world business settings, the financial cost of developing, maintaining, and supporting a new engine C_{new} needs to be considered when evaluating feasibility in light of business constraints.

Implicitly, increased throughput might offset cost by lowering cost-per-conviction or

Figure 4.1: The full email volume V_1 comprising both legitimate messages (smileys) and spam (bold "S") enters the pipeline from the left. A first slice of engines, E_{pre} , with combined B_{pre}, P_{pre} filters out a subset of $V_1 B_{pre}$ presumed bad messages. A conviction is considered a final action and identified spam is not further processed. Reduced volume $V_{post} = (1 - B_{pre}) V_1$ is passed on to the proposed new engine E_{new} , which in turn filters out $V_{post} B_{new}$ messages over P_{new} . It emits further reduced volume $V_{new} = V_{post}^* = (1 - B_{new}) V_{post}$ for final processing by E_{post} .

by increasing profit margin (more messages can be processed while hardware and operation cost stays constant and customers pay the same price). Following the reasoning put forth in equation (4.4), we refer to the expected time a message needs to pass through the existing pipeline prior to adding E_{new} as $P_{existing}$ and to the time including the additional engine as $P_{updated}$. $\frac{P_{existing}-P_{updated}}{P_{existing}}$ may then serve as a proxy for these cost savings. The term becomes 1 when the modified pipeline process messages without any delay at all, the ideal yet unattainable outcome. If pipeline operation is not affected by the modification, the term is 0. A negative term indicates the pipeline is operating less efficiently after the addition of E_{new} . Assuming the current operation as fixed parameters, the term can be computed as a function of P_{new}, B_{new} . An engine that produces cost savings adheres to the following inequality:

$$\begin{aligned}
& P_{updated} \leq P_{existing} \\
& \iff P_{pre} + (1 - B_{pre})P_{new} + (1 - B_{pre})(1 - B_{new})P_{post} \leq P_{pre} + (1 - B_{pre})P_{post} \\
& \iff \frac{P_{new}}{P_{post}} \leq B_{new} \tag{4.5}
\end{aligned}$$

With $C_{operation}$ accounting for the cost associated with operating the pipeline (*e.g.* hardware, cloud infrastructure, ...), the monetary value of the cost savings (or increase) can be computed as $\frac{P_{existing}-P_{updated}}{P_{existing}}C_{operation}$

Explicitly, if a new engine excels at convicting spam that competing solutions strug-

gle with, or that cause high financial liability, customers might be willing to pay an additional license fee attached to a proposed engine. This fee can be used to offset C_{new} . We have to distinguish several classes of convictions observed after the addition of a new engine and their relation to the original convictions. $B_{new} = B_{new+} + B_{overlap}$ has two components: B_{new+} is due to novel convictions that the existing pipeline missed. $B_{overlap}$ is due to messages that E_{post} would have labeled as spam already, but that are now convicted earlier on in the pipeline. Assuming that E_{new} can enrich the feature space (*e.g.* by exposing its features to be considered in conjunction with additional context in the remainder of the pipeline), B_{post+} pertains to additional convictions that E_{post} was able to issue due to this new signal. $B_{novel} = B_{new+} + B_{post+}$ quantifies all novel convictions attributable to the new engine. Figure 4.2 illustrates these concepts.

Market analysis allows assigning a specific value ζ that customers are willing to pay for each novel true positive $TP_{novel} = TP_{new+} + TP_{post+}$ produced by the pipeline after addition of E_{new} .

The expected financial impact of adding a new engine to the pipeline can now be assessed by combining these factors. The following inequality is true, when the expected efficiency gains, together with the additional revenue, amortize the cost of the proposed engine, indicating it is worthwhile to add from a financial standpoint:

$$\left(\frac{P_{existing} - P_{updated}}{P_{existing}} \right) C_{operation} + \zeta TP_{novel} - C_{new} \geq 0 \quad (4.6)$$

Accuracy of the pipeline is only implicitly considered as part of C_{new} via support cost (FPs are likely to cause complaints that a pipeline operator will have to address). More explicitly, the cost associated with errors in novel convictions pertaining to B_{new+} can be considered when tuning the proposed engine via TCR_{new} (see equation on page 51). When assessing the entire pipeline, as is done here, such considerations need to be extended to additional changes associated with B_{post+} . More in-depth analysis is performed in section 4.2.2.

While critical for practitioners, cost feasibility of the entire pipeline is not further investigated in this work. The dataset does not contain ground truth labels beyond the existing pipeline verdicts. We can thus not easily distinguish spam from legitimate mail in novel convictions and thus have to consider all of B_{new+} as contributions to the FP rate. Further, since we cannot assess to what extent the features expressed in our dataset have been (or could be) used by the existing pipeline, B_{post+} is assumed to be 0. Future research, beyond the scope of this thesis, might be conducted to try and identify whether novel convictions might be accurate. One possible way of doing so may be facilitated by combining domain knowledge with powerful model interpretation methods as illustrated in section 5.6.1 (Improving Ground Truth, page 99).

Figure 4.2: The left pie-chart shows convictions of the existing pipeline. Grey, homogeneous areas indicate mail that was allowed to pass the pipeline and was delivered (alleged legitimate). The several patterned areas highlight messages identified as spam: Dotted areas are convictions from E_{pre} , vertical stripes are associated with E_{post} . Right illustrates how a newly added engine E_{new} (horizontal stripes) influences the pipeline verdicts. $B_{new} = B_{new+} + B_{overlap}$ is composed of novel convictions and overlap with the existing verdicts. B_{post+} are additional conviction that E_{post} can issue due to features exposed by E_{new} .

4.2.2 Feasibility Considerations for a New Engine

A novel engine can be tuned independently to ensure it provides maximum utility. Two distinct characteristics, throughput and prediction errors, are reviewed in more detail in the following sections and then optimized jointly.

Feasibility Threshold for Throughput

One motivation for adding an engine (especially to the early stages of the pipeline) is to increase efficiency, or message throughput. For a new engine to be worthwhile in light of efficiency, the expected time to scan a message has to be decreased overall. While also relevant for saving cost, we can redeploy inequality (4.5) here again. It specifies the minimum required block rate of the new engine as the fraction of its processing time, over the processing time of the remaining pipeline. For example, if a new engine is 20 times faster than the remainder of the pipeline, said engine needs to convict at least $\frac{1}{20} = 5\%$ of the messages it is exposed to.

While accuracy of convictions is paramount in a real-world setting, it is not relevant for throughput optimization: If throughput was the only objective, a trivial engine simply convicting every message without delay would be the best possible outcome. To avoid such degenerate optimizations, throughput has to be considered jointly with prediction errors.

Feasibility Threshold for False Predictions

We note that there is incredibly harmful spam (phish, malware) and low-value ham (certain bulk-mail, reminders, ...). Still, spam detection in general operates under a strong bias against FPs. When considering real world email traffic and anti-spam systems, causing an FP (convicting a legitimate message) is considered a sever mishap. In addition, it is in practice much easier to collect spam for a training set (*e.g.* from spam traps) than it is to harvest representative legitimate mail which may carry private or sensitive data. Any FP observed in training or validation is thus likely to multiply and cause higher FP rate once a model is deployed. For these reasons, total cost ratio (TCR , see below) is used by SpamAssassin with $\lambda = 50$ [6], reflecting a significantly higher penalty for FPs versus FNs. $\lambda = 1$ signifies equal penalty, values $\lambda > 1$ place higher penalty on FPs. The following measures have been introduced by Androutsopoulos *et al.* [4] specifically for the context of spam detection:

$$\text{Weighted Accuracy } WAcc = \frac{\lambda TN + TP}{\lambda Neg + Pos}$$

$$\text{Weighted Error Rate } WErr = 1 - WAcc = \frac{\lambda FP + FN}{\lambda Neg + Pos}$$

$$\text{Total Cost Ratio } TCR = \frac{Pos}{\lambda FP + FN} \left(= \frac{WErr^b}{WErr} \right), \text{ with } WErr^b = \frac{Pos}{\lambda Neg + Pos}$$

In these definitions, $Pos = (TP + FN)$ is the number of positive (spam) messages, while $Neg = (TN + FP)$ is the number of negative (non-spam) messages. These

measures allow weighing FPs against FNs using a single parameter $\lambda \in \mathbb{R}^+$, which simplifies adjusting the model to a given application context.

TCR can be interpreted as the ratio of a baseline error rate $WErr^b$ representing the absence of a classifier where all messages pass, and the error rate of the proposed system. Greater TCR indicates better performance; harmful classifiers, where the impact of errors outweighs the benefits of convictions, have $TCR < 1$. With this interpretation, and considering that $FN = Pos - TP$, a new engine is beneficial when the following inequality is satisfied:

$$\begin{aligned} TCR_{new} &\geq 1 \\ \iff \frac{TP_{new}}{FP_{new}} &\geq \lambda \end{aligned} \tag{4.7}$$

Given a specific relative impact of FPs and TPs in an anti-spam system expressed by λ , equation (4.7) identifies it as the minimum ratio of TPs over FPs a proposed engine has to deliver in order to be worthwhile.

Joined Optimization Using ROC-Curve Analysis

A ROC-curve illustrates achieved FP and TP rate of a binary classifier, while a threshold distinguishing the two classes is modulated. The concept can be extended to compare model instances in light of their FP and TP rates, resulting from modulating any arbitrary model parameter (referred to as ρ in the following). Analyzing

a ROC-curve, considering application-specific constraints on FP and TP rates (and their ratio), allows selecting an appropriate value for the underlying model parameter. In practice, the parameter is modulated over a range of values and the ROC-curve can be described as a set of tuples, associating discrete value of a parameter ρ with corresponding FP and TP rates:

$$roc = \{(\rho, FPR(\rho), TPR(\rho))\} \quad (4.8)$$

Considering that FP and TP rates, together spanning the ROC-space, can be computed as $FPR = \frac{FP}{Neg}$, $TPR = \frac{TP}{Pos}$, we can reformulate equation (4.1) in terms of ROC-space:

$$\begin{aligned} B &= \frac{TP_k + FP_k}{V_k} \\ &= \frac{TPR * Pos + FPR * Neg}{Pos + Neg} \end{aligned} \quad (4.9)$$

Let us assume a minimum block rate B_{min} , serving as a threshold for throughput, as per inequality (4.5). The actual value of B_{min} is determined empirically by measuring the respective processing times P_{new} and P_{post} , where $B_{min} = \frac{P_{new}}{P_{post}}$, and is treated as a constant parameter in the following derivations. This facilitates superimposing the ROC-curve with a subspace containing model instances that are worthwhile. For a model instance with specific FP and TP rate to increase

throughput, the following inequality must hold:

$$\begin{aligned} \frac{TPR_{new} * P + FPR_{new} * Neg}{Pos + Neg} &\geq B_{min} \\ \Leftrightarrow TPR_{new} &\geq \frac{Neg}{Pos} \left(B_{min} \frac{Pos + Neg}{Neg} - FPR_{new} \right) \end{aligned} \quad (4.10)$$

Similarly, TCR can be expressed in ROC-space as:

$$\begin{aligned} TCR &= \frac{Pos}{\lambda FPR + Pos - TPR} = \frac{Pos}{\lambda FPR * Neg + Pos - TPR * Pos} \\ &= \frac{1}{1 + \lambda \frac{Neg}{Pos} FPR - TPR} \end{aligned} \quad (4.11)$$

Applying this formulation to equation (4.7), for a model instance to be worthwhile from a prediction accuracy standpoint, the following inequality has to be satisfied:

$$\begin{aligned} TCR_{new} &\geq 1 \\ \Leftrightarrow \frac{1}{1 + \lambda \frac{Neg}{Pos} FPR_{new} - TPR_{new}} &\geq 1 \\ \Leftrightarrow TPR_{new} &\geq \lambda \frac{Neg}{Pos} FPR_{new} \end{aligned} \quad (4.12)$$

To facilitate ROC-based optimization, the different ways an engine can benefit the pipeline need to be scaled, weighted and combined into a single utility function in ROC-space. We introduce a weighting parameter $\eta \in [0, 1]$ that allows to balance utility associated with conviction errors (TCR), and throughput (governed by block

rate B).

We further want to use the measures underlying utility in a way that ensures positive utility falls into $[0, 1]$ with 0 meaning no utility at all (neutral), and 1 associated with the best possible outcome. Negative values indicate that a model is harmful given the application constraints set forth by the pipeline operator. In other words, the pipeline is operating more optimally without addition of the proposed engine E_{new} if its utility is negative, suggesting E_{new} should not be added. Block rate B inherently satisfies this requirement. Quality of predictions (weighted and assessed via TCR) has to be mapped accordingly. $1 - \frac{1}{TCR}$ ranges from $-\lambda \frac{Neg}{Pos}$ to 1 and becomes 0 for $TCR = 1$, signifying a neutral model.

We note that for a dataset with no positive samples (no spam at all, $Pos = 0$), TCR is always 0. This simply states that in a world with no spam, no benefit can be gained from an anti-spam engine. We preclude this special case to avoid division by 0 in $1 - \frac{1}{TCR}$.

This allows the following formulation of the utility function in ROC-space:

$$\begin{aligned}
 u(FPR, TPR) &= \eta B + (1 - \eta) \left(1 - \frac{1}{TCR} \right) \\
 &= \eta \frac{TPR * Pos + FPR * Neg}{Pos + Neg} + \\
 &\quad (1 - \eta) \left(TPR - \lambda \frac{Neg}{Pos} FPR \right)
 \end{aligned} \tag{4.13}$$

Figure 4.3 illustrates how utility values are distributed over the ROC-space for relevant parameter settings.

For $\eta = 1$, only block rate B is considered and λ becomes irrelevant. Utility is then always positive and ranges from 0 when nothing is convicted ($FPR = TPR = 0$) to 1 when all messages are blocked.

At $\eta = 0$, utility is defined only by contributions pertaining to TCR . λ discriminates positive from negative utility in ROC-space: utility is positive, when $TPR > \lambda \frac{Neg}{Pos} FPR$.

For mixed utility with $0 < \eta < 1$, formulas are less intuitive given dependencies on several parameters. In the general case, positive utility is achieved when the following inequality holds:

$$TPR * \left[1 - \eta \left(1 - \frac{Pos}{Pos + Neg} \right) \right] > FPR * \left[\lambda \frac{Neg}{Pos} (1 - \eta) - \eta \frac{Neg}{Pos + Neg} \right] \quad (4.14)$$

As a simplified example, let us assume equal-sized classes ($Neg = Pos$). Equation (4.14) then indicates positive utility when $TPR \left(1 - \frac{\eta}{2} \right) > FPR \left(\lambda - \lambda\eta - \frac{\eta}{2} \right)$.

Maximizing utility as expressed by equation (4.13) under constraints (4.10), (4.12), and limited to (FPR, TPR) -coordinates on the ROC-curve, results in the optimal value for TPR_{new} and FPR_{new} . These values in turn determine the best possible parameter value underlying the ROC-curve. While utility and constraints are dif-

Figure 4.3: Utility distribution over ROC-space for different parameter settings, assuming equal size of the positive and negative class ($Neg = Pos$). 4.3a considers block rate only, 4.3b shows utility when only considering TCR at equal penalties for FPs and TPs. 4.3c and 4.3d show utility combining block rate and TCR and illustrate the effect of increasing FP penalty λ . We note that utility values have been truncated at -1 for better visualization but range down to -2.25 in 4.3d)

ferentiable, the ROC-curve is generally empirically computed for discrete parameter values as outlined in (4.8). This precludes analytical solutions. Instead, $TCR(\rho)$, $B(\rho)$, and the corresponding value of the utility function $u(\rho)$ have to be computed exhaustively for (FPR, TPR) -tuples associated with the ROC-curve and complying with constraints. The best tuple $(FPR, TPR)^*$ and corresponding parameter value ρ^* is then found among qualifying $(\rho, FPR, TPR) \in roc$ as:

$$(FPR, TPR)^* = \operatorname{argmax}_u u(FPR, TPR) \quad (4.15)$$

To summarize, differentiated cost and benefit may be assigned to the FPs and TPs incurred by a new engine. For example, an engine that convicts extremely harmful spam that is otherwise hard to address may be worthwhile, even if it reduces throughput and causes additional FPs. In our formulation this could be expressed by setting $\eta = 0$ and choosing $\lambda < 1$. On the other hand, to tune an engine designed purely to offload the remainder of the pipeline (that is, to optimize throughput), parameters $\eta \approx 1$ and $\lambda \gg 1$ would be appropriate.

As an example for tuning the discrimination threshold with superimposed constraints for prediction errors and throughput, assume the operator of an anti-spam pipeline specifies a value $\lambda = 5$ from empirically comparing the cost of being exposed to spam (requiring manual deletion and bearing the risk of infection or falling victim to a phishing attack) with the cost of having to restore a falsely convicted

Figure 4.4: ROC-curve for a GBM regression model and the lower bounds on TPR induced by $\lambda = 5$ and $B_{min} = 5\%$. Only the upper-left quadrant, above the dotted and dashed lines, is considered worthwhile. The blue cross indicates the point on the ROC-curve inside of the worthwhile section, for which the utility function reaches its maximum.

message. Let us further assume measurements of P_{new} and P_{post} suggest a minimum required block rate $B_{min} = 5\%$. These constraints when applied via equations (4.10) and (4.12) can be used to define a subspace containing feasible engines, when tuning the TPR vs FPR trade-off in a ROC-curve. Figure 4.4 illustrates this example.

The utility function and its maximization are influenced in several ways by its parameter values: B_{min} does not affect the utility function directly, but lower values allow for a higher discrimination threshold as fewer convictions required for a model to be worthwhile to increase throughput. If $\eta = 1$, TCR and thus conviction errors are

not considered in the utility function and λ becomes irrelevant to its shape. Highest utility is then always associated with maximum (worthwhile) conviction rate. However, λ still plays a role in defining the worthwhile threshold range. Higher values of λ correspond with increased FP penalty and thus control the lower bound for the discrimination threshold. As η decreases, penalty incurred by FPs as controlled by λ increases (assuming $\lambda > 1$). Lower values of λ result in more aggressive (FP tolerant) models, while larger values result in more conservative models. Figure 4.5 shows some combinations of these parameters and visualizes their impact on shape and feasible optimum of the utility function.

4.3 Expanding Feature Power

Feature power as outlined in section 2.4.4 (Feature Power, page 28) was developed for classification trees and Random Forest (RF) classifiers only. The following extends their original formulation to regression and GBM.

4.3.1 Feature Power for Regression Trees

In its original formulations, feature power is computed with respect to each class label c_k . The characteristic function of a coalition $v_{c_k}(p)$ is equally dependent on associations with a given class label. While association is defined differently in

Figure 4.5: Utility function for several parameter values: Colorful bold sections highlight worthwhile discrimination thresholds. Marks identify operating points as local maxima of the utility function limited to worthwhile thresholds. Lowering B_{min} extends the worthwhile range towards the right (higher decision thresholds). Decreasing λ extends the lower bound of the worthwhile range. For $\eta = 1$ (solid line), the operating point is always associated with maximum (worthwhile) conviction rate. As η decreases, λ controls the utility penalty incurred by FPs. Higher FP penalty requires higher discrimination thresholds (dashed line), while lower values result in lower thresholds (dotted line).

the three formulations *pathPow*, *cumNodePow*, *strictNodePow*, conceptually, if a coalition is a member of the class under consideration by some notion, it is referred to as a winning coalition. In regression, there strictly is no concept of class membership (and thus winners or losers).

We can re-cast a regression tree in terms of classification when we consider each possible leaf as its own class. With $l(t)$ returning the set of leaf nodes in (sub-)tree t , let $\{f(n_j), n_j \in l(T)\}$ be the set of all leaf node values in a tree. Note that we assume a bijective mapping: Even if two leaves have the same numerical value, they would still be considered separate classes. We indicate this requirement by using the $\equiv$ symbol. For J -terminal trees, the number of classes is then $K = J$. To simplify notation, it is convenient to define an abstract formulation of the Shapley value introduced in equation (2.13) as $\sum_{p \in U_{inc}} q_{inc}(p)v(p) - \sum_{p \in U_{exc}} q_{exc}(p)v(p)$, where U, q and v depend on the specific feature power formulation. It doesn't make much sense to compute feature power for each class in this setting. However, aggregate feature power for a regression tree can be computed as per equation (2.21):

$$\begin{aligned}
\Phi_{pow, f_j}(T) &= \frac{1}{K} \sum_{k=1}^K \phi_{f_j}[v_{c_k}](T) \\
&= \frac{1}{K} \sum_{k=1}^K \left[\sum_{p \in U_{inc}} q_{inc}(p)v_{c_k}(p) - \sum_{p \in U_{exc}} q_{exc}(p)v_{c_k}(p) \right] \\
&= \frac{1}{K} \left[\sum_{p \in U_{inc}} q_{inc}(p) \sum_{k=1}^K v_{c_k}(p) - \sum_{p \in U_{exc}} q_{exc}(p) \sum_{k=1}^K v_{c_k}(p) \right] \quad (4.16)
\end{aligned}$$

Depending on the feature power formulation and corresponding characteristic function, summation over all class labels (or all leaves in this context) yields the following:

1. *pathPow*, each coalition is a root-to-leaf path $p = (n_0, n_i), f(n_i) \equiv c_i$:

$$\sum_{k=1}^K v_{c_k}(p) = \sum_{k=1}^K \mathbb{1}[f(n_i) \equiv c_k] = 1 \quad (4.17)$$

Since there is exactly one path leading to any given leaf, the sum is always 1.

2. *cumNodePow*, each coalition is a root-originating path to an internal node $p = (n_0, n_i), n_i \notin l(T)$, and $n_k \in l(T)$ is a leaf node with $f(n_k) \equiv c_k$:

$$\begin{aligned} \sum_{k=1}^K v_{c_k}(p) &= \sum_{k=1}^K \frac{L(t'_{n_i}, c_k)}{\sum_{l=1}^K L(t'_{n_i}, c_l)} \\ &= \sum_{k=1}^K \frac{\sum_{n_s \in l(t'_{n_i})} \mathbb{1}[f(n_s) \equiv c_k]}{\sum_{l=1}^K \sum_{n_s \in l(t'_{n_i})} \mathbb{1}[f(n_s) \equiv c_l]} \\ &= \frac{\sum_{k=1}^K \mathbb{1}[n_k \in l(t'_{n_i})]}{\sum_{l=1}^K \mathbb{1}[n_l \in l(t'_{n_i})]} = 1 \end{aligned} \quad (4.18)$$

Since summing the number of leaves in a (sub-)tree that relate to a given class over all classes yields simply the number of leaves in the tree, the sum over characteristic functions in the *nodePow* scenarios are also always 1.

3. *strictNodePow*, each coalition is a single internal node $p = (n_i, n_i), n_i \notin l(T)$:

While coalition space differs, that does not influence computation of the sum

over characteristic functions. Equation (4.18) can be applied as stated.

Following these considerations, feature power in its three formulations can now be computed for regression trees:

$$\Phi_{pow, f_j}(T) = \frac{1}{K} \left[\sum_{p \in U_{inc}} q_{inc}(p) - \sum_{p \in U_{exc}} q_{exc}(p) \right] \quad (4.19)$$

Coalition space U , q_{inc} , and q_{exc} are computed according to the originally definitions for *pathPow* in equation (2.17), *cumNodePow* in equation (2.18), and *strictNodePow* in equation (2.19) starting on page 29.

Regression trees can be used with a discrimination threshold to perform binary classification. Feature power can be applied in such a setting as well. The extension from regression to regression-classification is explained in detail on page 67 in context of GBM ensembles.

4.3.2 Feature Power for GBM Regressors

Due to linearity of Shapley values and following the insight applied to TreeSHAP (see corollary 2.1), we propose that feature power can be expanded to GBMs in general. We can define the feature power pertaining to a GBM as the average of values computed for distinct trees in the ensemble, as outlined in equation (2.20).

Both operations involved in computing an average, summation and division by a constant, are permitted per definition 2.1.

Since output of a GBM regression model is simply the sum over individual regression trees (scaling is already considered by our formulation), considerations in section 4.3.1 (Feature Power for Regression Trees, page 60) are applicable with minor adjustments.

Considering a GBM ensemble of M trees with J nodes each, the total number of distinct classes is $K = J * M$. To maintain consistent bounds for aggregated feature power values, we use the average over trees (as opposed to the sum) following the original formulation in equation (2.22). The GBM equivalent to equation (4.19) for regression trees is:

$$\begin{aligned}
 \Phi_{pow, f_j} &= \frac{1}{M} \sum_{m=1}^M \Phi_{pow, f_j}(T_m) \\
 &= \frac{1}{M} \sum_{m=1}^M \frac{1}{J} \left[\sum_{p \in U_{m, inc}} q_{inc}(p) - \sum_{p \in U_{m, exc}} q_{exc}(p) \right] \\
 &= \frac{1}{JM} \left[\sum_{p \in U_{inc}^M} q_{inc}(p) - \sum_{p \in U_{exc}^M} q_{exc}(p) \right] \tag{4.20}
 \end{aligned}$$

$U^M = \bigcup_{m=1}^M U_m$ is the union over coalition spaces computed for distinct trees (*i.e.*, for $pathPow$ it consists of all root-to-leaf paths present in any of the M trees). The decision whether a given coalition p is an element of U_{inc}^M or U_{exc}^M and the exact

computation of q_{inc}, q_{exc} adhere to the original definitions.

4.3.3 Feature Power for GBM Classifiers

Some modifications to original formulations of feature power are necessary to account for distinctive output produced by trees in a GBM and their aggregation into the ensemble response (refer to section 2.3.1 for details).

We focus on binary logistic regression and classification ($c_k \in \{-1, 1\}$) with equal penalties ($c(-1, 1) = c(1, -1)$).

We note that this is done without loss of generality for the sake of notation simplicity. The underlying concepts are trivially expandable to more complicated settings: Since a K -class setting is realized via K independent ensembles in a GBM, feature power can first be determined for a specific class as outlined below and then aggregated over several trees and class labels as per the original formulation.

Differentiated misclassification penalties can be expressed by incorporating them directly into the p_{c_k} used to predict a class label in equations (2.5). Let us identify $c(-1, 1) = \exp(a) \in \mathbb{R}^+$. The term p_+ (and similarly its counterpart p_-) can be adjusted to directly incorporate the cost as follows: $p'_+(\mathbf{x}) = \frac{1}{\exp(1-a) + \exp(-2F(\mathbf{x})-a)}$. With this reformulation, we can use equal penalties for simplified notation as desired since $c(-1, 1)p_+ = p'_+$.

Using notation developed in section 2.3.4 (Simplified Notation for GBMs, page 20), and per equation (2.8), the response of tree T_m in a GBM, $r_m(\mathbf{x})$, can be interpreted as a shift away from a naïve prediction f_0 . $r_m(\mathbf{x}) = f(n_i)$ takes the value (including any potential scaling factors) of leaf n_i associated with $\mathbf{x}$ per equation (2.7). The response can be identified with a class label as $\hat{c}_m = \text{sign}(r_m)$. Magnitude $\text{abs}(r_m)$ then indicates the assertiveness (or weight) associated with the predicted class label in context of the ensemble.

This is an important distinction from Random Forest (RF) classifiers where the final class prediction is computed via a majority vote and the response of each tree is strictly one of the class labels (*i.e.*, $r_m \in \{-1, 1\}$ in RF). Even considering the different nature of the individual tree responses, the ensemble prediction for both GBM and RF can be expressed as:

$$\hat{c}(\mathbf{x}) = 2 * \mathbb{1} \left[\sum_{m=1}^M r_m(\mathbf{x}) > 0 \right] - 1 \quad (4.21)$$

While an unusual formulation for a majority vote as applied in RF, equation (4.21) returns the class label in $\{-1, 1\}$ that is most commonly returned by the M trees.

Before we proceed, let us quickly review how a GBM regression model can be used to produce binary classifications, when a discrimination threshold is applied to its output. For example, even though our setting is ultimately one of classification (spam or not), we employ a regression model as it is suitable for tuning multiple

objectives as outlined in section 4.2.2 and allows definition of distinct discrimination thresholds ρ for suspect and certain spam all using a single model.

In this setting, the predicted class label based on the GBM regression responses (each labeled as r'_m) is determined as:

$$\begin{aligned}\hat{c}(\mathbf{x}) &= 2 * 1 \left[f_0 + \sum_{m=1}^M r'_m(\mathbf{x}) > \rho \right] - 1 \\ &= 2 * 1 \left[f_0 + \sum_{m=1}^M \left(r'_m(\mathbf{x}) - \frac{\rho}{M} \right) > 0 \right] - 1\end{aligned}\tag{4.22}$$

As a first step, mostly for illustration, let us assume the discrimination threshold is set to the naïve prediction: $\rho = f_0$. We then simply set $r_m = r'_m$. For a more general scenario where $\rho \neq f_0$, it is possible that even if all responses are positive, the final prediction would be associated with the negative class when the sum of magnitudes is not sufficient to surpass the discrimination threshold. To account for the actually deployed discrimination threshold determined by ROC-analysis, we modify responses as outlined in equation (4.22) and define $r_m = r'_m - \frac{\rho}{M}$. After adjustment, responses can be directly associated with class labels via their sign as outlined above.

Following these considerations, characteristic functions in the three feature power formulations are redefined as follows:

1. *pathPow*, with $p = (n_0, n_i)$ and $n_i \in l(T)$:

$$v_{c_k}(p) = \text{abs}(f(n_i)) * \mathbb{1}[\text{sign}(f(n_i)) = c_k] \quad (4.23)$$

The characteristic function returns response magnitude, if its sign matches class label c_k . Aggregating over class labels (as in equation (4.17)) yields:

$$\sum_{c_k} v_{c_k}(p) = \text{abs}(f(n_i)) \quad (4.24)$$

2. *cumNodePow*, with $p = (n_0, n_i)$ and $n_i \notin l(T)$:

$$v_{c_k}(p) = \frac{\sum_{n_s \in l(t'_{n_i})} \text{abs}(f(n_s)) * \mathbb{1}[\text{sign}(f(n_s)) = c_k]}{\sum_{c_l} \sum_{n_s \in l(t'_{n_i})} \text{abs}(f(n_s)) * \mathbb{1}[\text{sign}(f(n_s)) = c_l]} \quad (4.25)$$

In this setting, a coalition is characterized by the sum of absolute leaf-values where the sign matches c_k , divided by sum of all absolute leaf-values, limited to the sub-tree rooted at the terminal node of the coalition path. Aggregating over classes (as in equation (4.18)) yields:

$$\sum_{c_k} v_{c_k}(p) = 1 \quad (4.26)$$

3. *strictNodePow*, with $p = (n_i, n_i)$ and $n_i \notin l(T)$:

Again, computation of the characteristic function, either in aggregate or per

class, doesn't depend on the coalition space and equations (4.25), (4.26) apply.

Based on the definitions outlined above, feature power for a given class $\Phi_{pow,f_j}[v_{c_k}]$ can be computed per equation (2.20), and aggregate feature power Φ_{pow,f_j} over all classes via equation (4.20).

This concludes the expansion of feature power into regression trees, and GBMs in both, the classification and regression setting.

We note that implementing the formulas outlined above directly, yields a very inefficient algorithm. Speedup of 1.77 for *pathPow*, 2.0 for *cumNodePow*, and 2.2 for *strictNodePow* can be achieved simply by caching computation of the $q_{ext/inc}$ in equation (4.16). Since maximum tree depth constitutes an upper bound for the size of a coalition, memory footprint of the cache is trivial.

Chapter 5

Experiments

The following sections are dedicated to studying our dataset and testing methods put forth in section 4 ([Theoretical Considerations](#), page 41).

5.1 Software Environment

Experiments outlined in this chapter are implemented in python 3 and organized in jupyter notebooks. New software packages, most noteworthy the extension to feature power, are also implemented in python 3. If not otherwise specified, we employ scikit-learn 0.23 machine-learning models and training strategies such as cross-validation. The data-matrix is realized as a pandas DataFrame (see [Appendix A: Example Data](#)). Graphs are created using matplotlib. We measure timing on a 2.3 GHz 8-core Intel i9 processor using 16GB of RAM.

5.2 Types of Predictions

We train models F that use features $\mathbf{x}$ to predict the existing pipeline label (score or verdict). The score is used in regression models as is. The verdict can be interpreted in different ways: An aggressive stance that prioritizes security and spam conviction rates might drop messages with either a positive or suspect verdict and deliver only neutral messages. A conservative stance that is very sensitive to FPs will drop only positive spam and deliver both suspect and neutral messages. Finally, a three-class approach can be taken where neutral messages are delivered, suspect messages are treated with a less severe action such as being placed in a quarantine or being delivered with a modified subject line, while positive spam is dropped. The following four types of predictions $F(\mathbf{x})$ are investigated in our experiments:

aggressive $F(\mathbf{x}) \rightarrow \{+ \text{ if verdict } \in \{+, ?\}, - \text{ if verdict } \in \{-\}\}$

conservative $F(\mathbf{x}) \rightarrow \{+ \text{ if verdict } \in \{+\}, - \text{ if verdict } \in \{?, -\}\}$

three-class $F(\mathbf{x}) \rightarrow \{+, ?, -\}$

regression $F(\mathbf{x}) \rightarrow [0, 1]$

5.3 Mean, Mode, and Aggregation Baselines

A first set of baselines establishes a lower bound of what we should expect from more sophisticated models. The naïve baseline refers to a model simply returning the average score for regression or the most prevalent verdict (mode) for classification in our dataset.

To improve on this approach, we create a lookup table: Verdicts and scores are aggregated for identical feature vectors. Then the mode of verdicts and the average of scores are computed and associated for each unique feature vector. We also append to this lookup-table a catch-all entry for unseen feature vectors associated with the naïve verdict mode and average score from the dataset. We refer to this approach as the "LookupTable" going forward. We note that it is akin to combining the naïve baseline model as the default, with a radius-nearest-neighbor model requiring identity (distance 0).

5.4 Baseline Predictions and Model Comparison

In this section, we experimentally quantify the potential of SDR to approximate verdicts of a state-of-the-art spam classifier using low-fidelity features available at this early stage of the prediction pipeline. Using the verdict of a high-fidelity and high-confidence spam classifier as ground-truth, we identify baselines for spam detection

using a selection of standard ML models with our SDR dataset. While the existing model verdict is not necessarily correct (*i.e.*, there can be false positives, FPs, and false negatives, FNs), the new classifier with SDR features is trained specifically and exclusively to approximate it. In other words, no attempt is made to increase the overall rate of spam convictions. Moreover, if the new SDR classifier identifies a message as spam when the original verdict was not spam, this will be considered an FP and penalized as such.

To assess the predictive power of the SDR features, we split the dataset and use the first 100,000 entries (76%) for training and the remaining data for validation. We set out to compare several state-of-the-art models. Parameters for each are tuned via randomized search using 5-fold cross-validation (CV) over 20 iterations. Parameter ranges for the search are manually adjusted over several rounds: Starting with a wide bracket around commonly recommended values, in each round the ranges are narrowed to hone in on most promising values. Once parameters are tuned, we validate the models and compute a set of test scores via 20-fold cross-validation. Models (with established parameters) are trained using the full training data and a validation score is computed using the validation set. Accuracy score is computed for classification, coefficient of determination (R^2) for regression. We report the average over 20 cross-validation scores. Models are compared pairwise via a Wilcoxon signed-rank test to assess whether they are statistically distinguishable (at $p = 5\%$) from each other. Training and prediction times are also recorded.

Tables 5.1, 5.2, 5.3, and 5.4 summarize our findings. LightGBM (and other GBM implementations) perform best and deep neural networks and decision trees are competitive in some settings. LookupTable outperforms many standard models and ranks around the middle. AdaBoost, RandomForest (RF), Support Vector Machines (SVM), and Naïve Bayes and Perceptron (the latter two are limited to classification) perform poorly. This is surprising and in contrast to common assumptions and results in anti-spam where those models usually perform very well [59, 65].

5.4.1 Comparison of GBM Implementations

In the previous section we compared four GBM implementations and found varying performance among them with statistically significant difference. This is at least in part due to varying success in parameter tuning. To compare implementations in a more controlled way, we run the following experiment. We identify a set of parameters that critically affect performance, namely learning-rate, number of trees, number of leaves in each tree, and L2 regularization. For each setting, tuned parameter values from the best performing model (LightGBM in all settings) are applied to all four GBM implementations. We also consistently set the tree growth strategy to best-first tree growth. This is the default for LightGBM [30], while CatBoost defaults to symmetric trees and XGBoost grows trees depth-wise by default.

Results of our comparison are detailed in table 5.5. LightGBM performs best in

model	score (R^2)		time (seconds)	
	average CV	validation	training	prediction
LightGBM	0.3688	0.3766	58.1	2.7
XGBoost	0.3668	0.3733	3742.8	211.1
CatBoost	<i>0.3474</i>	0.3559	316.6	15.3
GBM	<i>0.3460</i>	0.3535	343.2	21.3
NeuralNet	<i>0.3403</i>	0.3462	1579.7	63.4
KNearestNeighbors	0.3197	0.3351	514.4	10.0
DecisionTree	0.3176	0.3260	15.3	0.7
LookupTable	0.3132	0.3145	441.2	20.9
Ridge	0.2090	0.2183	8.7	0.3
SGD	0.1844	0.1915	14.3	0.6
RandomForest	0.1694	0.1762	350.5	11.7
LinearSVM	0.0837	0.0916	182.0	9.7
AdaBoost	0.0827	0.0862	160.9	9.8
ElasticNet	0.0589	0.0582	10.6	0.4
Lasso	-0.0036	-0.0020	8.0	0.3

Table 5.1: Model comparison for the regression setting, ordered by validation score. Bold (maroon) numbers highlight best values in the set. LightGBM produces best scores, followed by XGBoost. Other GBMs and NeuralNet rank third and are statistically indistinguishable from each other at $p < 5\%$ (indicated by italics and shading).

model	score (accuracy)		time (seconds)	
	average CV	validation	training	prediction
LightGBM	0.7594	0.7618	64.0	3.0
XGBoost	0.7573	0.7605	3307.1	165.3
DecisionTree	<i>0.7513</i>	0.7563	15.1	0.7
NeuralNet	<i>0.7512</i>	0.7554	5063.6	205.3
CatBoost	0.7421	0.7467	388.0	19.5
KNearestNeighbors	0.7351	0.7445	527.4	10.2
GBM	0.7410	0.7426	278.5	13.7
LookupTable	0.7333	0.7333	417.3	19.4
LinearSVM	0.6888	0.6963	135.6	6.4
AdaBoost	0.6887	0.6956	4633.8	242.1
LogisticRegression	0.6885	0.6950	4758.2	240.5
RandomForest	0.6908	0.6922	230.1	8.7
SGD	0.6733	0.6801	16.0	0.7
Perceptron	0.6008	0.6440	14.5	0.8
NaïveBayes	0.6220	0.6385	12.9	0.5

Table 5.2: Model comparison for binary classification in the aggressive setting, ordered by validation score. Baseline is given by prevalence of most common label as 0.5103. Bold (maroon) numbers highlight best values in the set. LightGBM produces best scores, followed by XGBoost. NeuralNet and DecisionTree rank third and are statistically indistinguishable at $p < 5\%$ (indicated by italics and shading).

model	score (accuracy)		time (seconds)	
	average CV	validation	training	prediction
LightGBM	<i>0.7588</i>	0.7698	55.7	2.6
XGBoost	<i>0.7593</i>	0.7694	3190.6	167.4
GBM	0.7517	<i>0.7619</i>	243.8	10.0
CatBoost	0.7507	<i>0.7619</i>	278.3	13.9
NeuralNet	0.7542	0.7603	5613.6	317.9
KNearestNeighbors	0.7439	0.7567	594.9	11.4
DecisionTree	0.7468	0.7541	14.4	0.6
LogisticRegression	0.7285	0.7442	5042.4	237.1
AdaBoost	0.7272	0.7437	4975.8	259.3
LinearSVM	0.7279	0.7436	951.6	51.3
LookupTable	0.7367	0.7414	418.5	19.3
RandomForest	0.7133	0.7290	313.1	11.9
SGD	0.7087	0.7274	19.6	0.9
NaïveBayes	0.7038	0.7225	13.8	0.6
Perceptron	0.6529	0.5518	14.2	0.5

Table 5.3: Model comparison for binary classification in the conservative setting, ordered by validation score. Baseline is given by prevalence of most common label as 0.6642. Bold (maroon) numbers highlight best values in the set. LightGBM produces best validation score, XGBoost best training scores in cross-validation. The two models are statistically indistinguishable at $p < 5\%$ (indicated by italics and shading). Other GBMs rank seconds and are also statistically indistinguishable.

model	score (accuracy)		time (seconds)	
	average CV	validation	training	prediction
LightGBM	0.6552	0.6623	147.0	6.9
XGBoost	0.6525	0.6616	6388.6	334.1
CatBoost	0.6506	0.6589	407.4	20.5
NeuralNet	<i>0.6447</i>	0.6576	4755.4	300.5
GBM	<i>0.6470</i>	0.6517	717.8	19.8
DecisionTree	0.6404	0.6505	16.0	0.7
KNearestNeighbors	0.6320	0.6460	584.6	10.0
LinearSVM	0.6148	0.6293	5662.2	320.5
LogisticRegression	0.6152	0.6288	3204.2	177.9
LookupTable	0.6208	0.6263	417.9	19.1
AdaBoost	0.6099	0.6238	5017.2	261.4
RandomForest	0.5956	0.6140	235.7	8.4
SGD	0.5724	0.5947	15.7	0.7
NaïveBayes	0.5535	0.5745	14.2	0.6
Perceptron	0.5153	0.5540	18.2	1.0

Table 5.4: Model comparison for three-class setting, ordered by validation score. Scores reflect accuracy. Baseline is given by prevalence of most common label as 0.4897. Bold (maroon) numbers highlight best values in the set. LightGBM produces best scores, followed by other GBM implementations.

all cases. As in the baseline comparison, XGBoost remains competitive in the conservative setting. Practically, since parameter tuning requires several iterations of training and evaluation, the speed with which such iterations can be performed, should be a significant criteria in selecting a library. LightGBM is the fastest implementation by far. Given competitive prediction quality at significantly lower training and prediction time, we decide to use LightGBM as the implementation of choice for the remainder of this work.

5.4.2 Regression Versus Classification

Even when ultimately requiring a prediction of a verdict (classification), we can achieve that goal in two different ways: (i) train classification models directly for the different scenarios (aggressive, conservative, and three-class), or (ii) train a regression model and apply thresholds on the scores, mapping ranges to appropriate verdicts. The latter has benefits since a single model can be used to produce verdicts for all three scenarios. On the other hand, the former may benefit from higher accuracy. Addressing this question, we compared the two scenarios below.

Figure 5.1 provides a high-level comparison between the two approaches. When applying the same discrimination thresholds that are used in the pipeline (≥ 0.5 for a suspect, ≥ 0.9 for a positive verdict), regression-classification approach produces significantly more suspect verdicts while native classification results in more

model	score (R^2 or accuracy)		time (seconds)	
	average CV	validation	training	prediction
regression				
LightGBM	0.3688	0.3766	57.7	2.9
XGBoost	0.3651	0.3721	3884.1	202.0
GBM	<i>0.3554</i>	0.3646	438.4	23.8
CatBoost	<i>0.3517</i>	0.3594	951.1	47.1
aggressive				
LightGBM	0.7594	0.7618	65.3	3.3
CatBoost	<i>0.7573</i>	0.7605	1057.3	56.1
XGBoost	<i>0.7570</i>	0.7594	3517.7	188.6
GBM	0.7496	0.7536	394.0	20.4
conservative				
LightGBM	0.7588	0.7698	58.1	2.7
XGBoost	<i>0.7578</i>	0.7679	3038.0	159.5
CatBoost	0.7569	0.7674	877.3	44.9
GBM	0.7542	0.7659	360.3	22.4
three-class				
LightGBM	0.6552	0.6623	149.9	6.7
XGBoost	0.6530	0.6618	8738.9	465.8
CatBoost	0.6510	0.6595	1041.7	53.3
GBM	0.6487	0.6590	561.4	39.3

Table 5.5: Comparison of several GBM implementations. Bold (maroon) numbers highlight best values in a setting. LightGBM consistently produces best CV and validation scores. It is also significantly faster than competing implementations. Models with CV scores that are statistically indistinguishable from each other at $p < 5\%$ are highlighted with italics and shading.

Figure 5.1: Comparison of existing pipeline verdicts and scores and model predictions in native classification and regression-classification approaches. Native classification pushes distribution towards the boundaries with more-than-original negative and significantly fewer suspect verdicts. Regression-classification approach shown in orange does the opposite and concentrates many samples in the suspect range.

positives. Both approaches exceed the number of existing pipeline verdicts in the negative class. Reviewing scores of the regression-classification model shown in the right plot of figure 5.1 in more detail, we observe that the distribution of scores is much more even than existing pipeline scores. This indicates that the regression-classification model struggles to tease scores away from the center based on the low fidelity features.

Visualizing predicted scores and labels offers a glimpse in the nature of the data and models. However, we have to also review how accurate predictions are. Figure 5.2 shows the confusion matrices produced by different approaches. We again observe that regression-classification models generate fewer positive convictions. Whether the loss in TPs together with the associated reduction in FPs constitutes a good

trade-off depends on application specific evaluations. At this stage, neither approach is clearly superior. Tuning of the thresholds using ROC curves might improve results of the regression models. We defer more detailed investigations to section 4.2.2 (Joined Optimization Using ROC-Curve Analysis, page 52).

5.4.3 Observed Lack of Overfitting in Deep Trees

During parameter tuning, we observe that maximum depth of trees (or number of leaves) critically influences model quality. Deep trees (standalone, in a Random Forest, or in GBMs) outperformed shallower trees while maintaining high CV and validation scores. Generally, deeper trees are expected to produce higher training scores but start to perform worse in validation due to overfitting. This does not seem to be the case here. To investigate this observation further, we trained multiple DecisionTree and GBM models over a range of 2 to 200 leaves per tree and measured mean absolute training and validation error. Tree depth has not been constrained explicitly and might vary depending on tree growing strategies employed by the models. Figure 5.3 shows mean absolute error of tree-based models in training and validation as a function of the number of leaves. Both training and validation errors strictly decrease.

Lack of overfitting is likely due to data duplication (described in section 3.1). As a reminder, frequently occurring feature vectors representing highly prevalent email

Figure 5.2: Comparison of native classification and regression-classification models.

Figure 5.3: Mean average error in training and validation both decrease with tree depth. Overfitting would be indicated by increasing validation errors after a certain point. This is not observed here.

senders are expected to be repeated up to 20 times distributed roughly evenly over our dataset. Different splits during cross-validation (and the data in the validation set) are thus expected to also contain such identical feature vectors and can thus be similar to each other. This assessment is further supported by the surprisingly high score achieved by the "LookupTable" baseline model as shown in tables 5.1, 5.2, 5.3, and 5.4: If the test split (or validation set) contained a lot of data that were not represented in the lookup table, we would expect its performance to degrade to the naïve baseline (predict verdict mode for classification, score average for regression). This is not the case; LookupTable ranks around the middle of all models under consideration.

This result also suggests significant high-order feature interactions. To illustrate, let us consider the modeled function in its ANOVA decomposition:

$$F(\mathbf{x}) = \sum_{j=1}^{|D|} \varphi_j(x_j) + \sum_{j,k=1}^{|D|} \varphi_{jk}(x_j, x_k) + \sum_{j,k,l=1}^{|D|} \varphi_{jkl}(x_j, x_k, x_l) + \dots \quad (5.1)$$

Each sum consists of functions φ that consider only a subset of features in $(\mathbf{x})$. The first sum covers the effect of features considered independently (referred to as the "main effects"), the second covers interactions of all feature pairs, and so forth.

Considering trees, each path can be seen as an interaction of features leading to a specific prediction. The maximum depth of a tree limits the number of features that can interact. In many datasets reviewed in the literature, lower order interactions

(*i.e.*, trees with less than 8 leaves) are sufficient to produce best possible results (*e.g.* [23] and citations therein). For our data, higher order interactions encoded in deeper trees (all the way up to 200 leaves) keep improving results. Poor performance of Naïve Bayesian and linear models (such as Lasso, ElasticNet, Ridge, and LogisticRegression) which by their very nature utilize only the main effects, also support this assertion.

5.5 Pipeline Optimization for Application Constraints

In section 4.2.2 we introduced a framework that can be used to tune a considered engine in context of an existing pipeline and external application constraints. In the following, we propose several ways to reduce engine cost and investigate resulting trade-offs with regards to engine utility. Our experiments are performed using the dataset described in chapter 3 (Dataset, page 34).

5.5.1 FP Avoidance

If a model is introduced mostly to enhance computational efficiency, not to convict additional messages, FP avoidance becomes paramount. A simple way to disproportionately penalize FPs is to assign higher weights to data in non-positive classes during model training. We investigate the effectiveness of this strategy by assigning

penalties (somewhat arbitrarily but reflecting intention behind different classification settings) to types of predictions underlying the confusion matrix (see table 5.6). We proceed by interpreting the sum over penalties for each of the three existing pipeline verdicts as base weights w_c for each class $c \in \{+, ?, -\}$, thus w_+ , $w_?$, and w_- denote weights for positive, suspect, and neutral ground-truth labels, respectively. Finally, we increase weights $w(c, l)$ for classes with a factor l ranging from 0 to 500 as follows: $w(c, l) = l(w_c/w_+ - 1) + 1$. This formulation ensures weight w_+ for the positive class (that cannot produce FPs) is always 1, while weights for non-positive classes $w_?, w_-$ are penalized more with increasing l . Figure 5.4 illustrates the correlation of FP and TP rate depending on l in a ROC curve for the aggressive, conservative, and three-class formulation.

Determination of appropriate class weights (controlled by parameter l) depends on the exact application requirements. As a precursor, parameters for cost feasibility as per section 4.2.1 can be established as follows: The pipeline operator determines cost of implementing and operating the new engine C_{new} . While changing class weights might slightly alter prediction speed, a rough estimate of average processing time of the new engine and P_{new} can be established. In the current scenario of throughput optimization under FP avoidance, we'll assume $TP_{novel} = 0$ which makes ζ irrelevant. From the cost perspective, the engine is worthwhile only if $P_{updated} < P_{existing} \left(1 - \frac{C_{new}}{C_{operations}}\right)$. This allows setting a fixed lower bound to B_{new} from a business perspective.

	existing pipeline verdict		
prediction penalty	−	?	+
aggressive			
−	3	5	
+	9	1	
conservative			
−	3	5	
+	13	1	
three-class			
−	$3 = (3 + 3)/2$	$4 = (5 + 3)/2$	$5 = (5 + 5)/2$
?	$6 = (9 + 3)/2$	$2 = (1 + 3)/2$	$3 = (1 + 5)/2$
+	$11 = (9 + 13)/2$	$7 = (1 + 13)/2$	$1 = (1 + 1)/2$

Table 5.6: Penalties $c(prediction, pipeline)$ for different types of (mis-)classifications and settings outlined in section 5.2. When the sole purpose of a new engine is to off-load the remainder of the pipeline, TNs produce no gain, FNs no harm (they are thus associated with low penalties). TPs are the desired outcome. FPs are to be avoided. Higher FP aversion in the conservative setting is reflected by a higher penalty. The three-class setting inherits and combines penalties from binary scenarios. Relative class weight is determined as the sum over three-class penalties: $w_- = 3 + 6 + 11 = 20$, $w_? = 4 + 2 + 7 = 13$, $w_+ = 5 + 3 + 1 = 9$.

Figure 5.4: FP and TP rates for a range of multiplier l applied to base weights of negative and suspect classes as defined in table 5.6. Higher values of multiplier l result in models with lower FP and TP rates. Solid lines represent the FP and TP rate achieved with $l = 500$ (downward triangle) on the very left, and its development for lower values ending with $l = 0$ (no differentiated class weight, upward triangle) at the right for native classification GBMs. Dashed lines show the same for regression-classification models operating at original pipeline thresholds (spam at ≥ 0.9 for conservative and ≥ 0.5 for aggressive). Thin dotted lines show full ROC-curves for regression-classification models. Higher values of multiplier l are located at very steep sections of the ROC curve indicating a high TPR/FPR ratio (though at low conviction rates overall). Modifying the discrimination threshold even slightly leads to drastic changes in TPR. For lower values of multiplier l the ROC-curves are flatter and tuning the discrimination threshold has lesser effect on the TPR/FPR ratio. Overall, native classification models are generally more aggressive and produce both, higher TPR and FPR for any given value of multiplier l .

Concurrently, trade-off between FPs and TPs needs to be established to determine an appropriate parameter λ . Trade-off between increased cost savings by raising throughput and additional cost incurred by FPs can be formulated via the utility function that linearly combines these factors via parameter η .

For the classification setting, produced TP and FP rates pertaining to a choice of l have to be considered as is. For regression, an additional tuning step can be performed to establish a discrimination threshold that maximizes utility.

Table 5.7 provides an overview of how GBM models react to different values of l . We assume a value of $\lambda = 20$ and required TP rate of 15% to exemplify some specific application context. Results show that several models can be outright disqualified based on insufficient TP convictions or low *TCR*. Among worthwhile candidates best TCR is achieved in the regression setting with $l = 10$.

Figure 5.5 illustrate the utility landscape over a grid of values for λ , η , and l . Utility is optimized for regression-classification in the aggressive setting. Such depictions can be the basis of quantitative reasoning about design choices considering unknown or more flexible application constraints. We find highest utility for moderate values of l between 0 (no differentiated weights) when only throughput is considered and FP penalty is low, up to 2 for higher values of λ .

	classification			regression		
l	$TCR(\lambda = 20)$	TPR	FPR	$TCR(\lambda = 20)$	TPR	FPR
0	0.411	33.6%	10.2%	0.534	29.6%	7.1%
1	0.824	23.7%	3.4%	0.942	21.7%	2.5%
2	0.989	21.2%	2.2%	1.049	19.7%	1.8%
5	1.090	18.5%	1.5%	1.165	17.2%	1.0%
10	1.139	17.0%	1.1%	1.182	15.4%	0.8%
20	1.154	16.1%	1.0%	1.172	13.9%	0.7%
50	1.142	15.0%	0.9%	1.160	12.6%	0.6%
100	1.144	14.5%	0.9%	1.160	12.1%	0.6%
500	1.133	14.0%	0.8%	1.143	11.3%	0.5%

Table 5.7: Assume a TP rate of 15% at the aggressive stance is required to justify introducing a new model. Let us further assume the cost of dealing with an FP is equivalent to the savings produced by 20 TP convictions (suggesting a value of $\lambda = 20$). The table rows contain $TCR(\lambda = 20)$ as well as FP and TP rates for GBMs trained in the aggressive classification and the regression formulation over a range of multiplier l for non-positive class-weights. Higher weight of non-positive training data reduces convictions. Grayed out rows represent models that do not satisfy the required TPR or result in $TCR < 1$ and can thus be eliminated. Bold, shaded values highlight the best model (highest feasible TCR), achieved in the regression setting trained with $l = 10$.

Figure 5.5: Heatmaps illustrate the utility landscape over a grid of values for λ , and l for select utility trade-off parameters η . Utility is computed for regression-classification in the aggressive setting. Assigning higher weight to positive classes at fixed FP penalty λ yields utility gains. Red dots mark highest utility for each value of λ .

5.5.2 Reducing the Number of Features

Verdicts should be produced parsimoniously with as few features as possible. This goal is induced by constraints on performance and cost. We will investigate two means to that end in the following sections.

Penalizing First Use of each Feature

As outlined in section 2.3.5 (Cost Efficient Gradient Boosting, page 21), cost efficient gradient boosting (CEGB) allows optimizing GBMs to user fewer (and cheaper) features.

In this experiment, we define cost of a model via cost of its features. The assumption is that acquiring and serving a feature incurs ongoing operational cost. We further assume that cost of a feature correlates with size of the feature (in our case the number of unique sender domains associated with a feature). While grossly oversimplifying cost estimation for a proposed model or its features, this approximation is suitable here since it can be computed directly from the dataset.

First, we establish cost of each feature by counting the unique domains associated with it over the entire dataset. We then scale cost into $[0, 1]$ by dividing with the maximum cost value.

Figure 5.6: Increasing penalty γ leads to faster and cheaper models. Higher values of α reduce prediction speed but do not significantly limit the number of features used by the GBM (and thus do not reduce feature cost). Accuracy is negatively impacted by both types of penalties.

We proceed with training GBMs regression models over a grid of parameters controlling the penalty to include new features (γ) and creating of splits (α). We then determine the discrimination threshold used to distinguish spam from legitimate mail by optimizing the utility function (equation (4.13)) developed in section 4.2.2 (Feasibility Considerations for a New Engine, page 50). For each model, we also track its cost defined as the sum of cost associated with the features in use. Figure 5.6 shows how increasing penalties reduce feature cost and prediction time, at the penalty of reduced accuracy.

We then simulate two scenarios representing different application requirements. The results are illustrated in figure 5.7.

In the first situation, speed of predictions is at the essence. We set utility function

parameters accordingly to $\lambda = 1$ reflecting no disproportionate FP penalty and $\eta = 0.9$ which places high weight on increasing block rate of the engine. Increased prediction speed and lower impact of decreased accuracy yields utility gains with a maximum at $\gamma = 0, \alpha = 5 * 10^5$ (figure 5.7a).

Secondly, we use parameters $\lambda = 5$, and $\eta = 0.5$ representing a balance between throughput and accuracy, and high penalty for FPs. Utility strictly decreases with increasing penalties as speed-up alone never outweighs the loss in accuracy (figure 5.7b). When considering a business setting where reduced feature cost is considered in tuning an engine (simulated by subtracting 10% of cost from utility), a new optimum is found at $\gamma = 500, \alpha = 0$ (figure 5.7c).

Reducing SDR Contexts

In section 2.2, we distinguish two phases in which SDR can produce and enforce verdicts. The earlier envelope phase facilitates more efficient blocking actions and also operates on lower dimensional data. Limiting the model to operate solely on the envelope is thus expected to allow significant cost savings. Concurrently, we should expect reduced number of convictions or more prediction errors of the model.

We investigate how the model reacts to drastic pruning of input features by removing all dimensions pertaining to the "data phase" (specifically, *hdrFrom*, *hdrReplyto*, *listunsub*, *dmarc*, and *dkim*). This reduces the number of features from 252 to 125.

(a) utility (throughput) $\lambda = 1, \eta = .9$ (b) utility (balanced) $\lambda = 5, \eta = .5$ (c) utility (balanced) - 10% of feature cost

Figure 5.7: Utility landscapes by CEGB models with three different business settings. Red circles indicate the global maximum. Depending on the parameters of the utility function, CEGB can yield improvements for specific penalties (left). When that is not the case (middle), reduced cost might offset the loss of model quality (right). Distinct color pallet in the left panel is used to indicate different range of utility values compared to the other two.

To simulate a range of application constraints, LightGBM regressions-classification models are optimized for accuracy and throughput for the aggressive setting via utility function (4.13) over a grid of parameters η, λ . Our findings are visualized in figure 5.8. Maximum achievable model utility generally decreases with more stringent constraints (higher FP penalty λ combined with lower values for η placing more weight on accuracy). Panel 5.8a shows utility for the model trained on complete feature set, including features available during the data phase (referred to as the full model going forward), as a reference.

While the model using only envelope features, referred to as the envelope model in the following, follows a similar trend, the pairwise difference of model utility (panel 5.8b), and corresponding TPR and FPR (panels 5.8c, 5.8d) shows a more nuanced picture: While utility is strictly lower in the envelope model, TPR and FPR can be increased or decreased when the discrimination thresholds are optimized for a specific choice of η, λ . This results in non-monotonous variations in utility loss. For example, utility loss is much smaller for $\eta = 0.6, \lambda = 4$ compared with $\eta = 0.8, \lambda = 3$. Depending on the actual cost savings that can be achieved by limiting the model to only operate on the envelope phase features, the trade-off might be worthwhile for the first set of constraints but not for the second.

Determining the best scenario depends on parameters driven by business and application requirements beyond the scope of this work. But conceptually, section 4.2.1 (Excuse Into Cost Feasibility of Pipeline Modification, page 44) and equation (4.6)

illustrate how a decision can be made: Limiting to the envelope phase reduced model complexity and fewer input features are used. This would lower C_{new} . More efficient conviction enforcement can increase model throughput, resulting in reduced $C_{operation}$. At the same time, revenue increase ζTP_{novel} would also be limited by lower TP rate. When specific parameters are known for a given environment, the two scenarios can be compared quantitatively and our framework allows the practitioner to make data driven decisions when faced with the design choice of whether or not to reduce model complexity.

5.6 Model Interpretation

5.6.1 Improving Ground Truth

The verdicts and scores in our dataset are imperfect. Clustering of data points based on their features yields clusters of varying purity with respect to their verdicts. Clusters with mixed verdicts might reflect actual diversity that low-fidelity features cannot separate (verdicts reflect ground truth), or similar messages the existing pipeline failed to convict consistently (*e.g.* due to dynamic learning that allowed the beginning of a spam campaign to bypass detection while the tail of the campaign is convicted). High dimensionality and uncertainty about the nature and quality of features makes interpreting clusters difficult. Noisy features are treated similar to

Figure 5.8: Maximum utility (and corresponding TPR and FPR) is computed over a grid of parameter values η, λ for a model trained on the full 252 features and one that is limited to 125 features from the envelope phase. Models use regression-classification in the aggressive setting. Distinct color pallet in the upper panels is used to indicate different range of utility values. 5.8a shows raw utility of the model trained on full data for reference. 5.8b shows Δ utility computed as utility achieved by the envelope model, subtracted from corresponding values of the full model. Higher value indicates higher utility loss. When throughput is the main concern (high values of η) and FP penalty is low, the envelope model has the same utility as the full model, but at drastically reduced cost. A secondary region of low utility loss is found around $\eta = 0.6, \lambda = 4$. Depending on the cost savings, limiting the model to the envelope phase features might be worthwhile. Remaining graphs indicate the magnitude by which the FPR and TPR can be improved based on additional features.

features with high predictive power and can make messages that are very similar in reality appear distant in feature space. Rather than clustering over the raw features, the authors of TreeSHAP propose clustering over SHAP values computed for predictions [43]. This induces a notion of distance, that directly reflects the way in which features contributed to the model decision, preventing noisy or otherwise distracting features from disturbing clustering. Further, when reviewing a cluster, this approach elucidates how a certain predictions was established. A domain expert can now much more readily interpret the cluster and re-assign labels if appropriate.

To illustrate the approach, we compute SHAP values and predictions over the dataset using a LightGBM regression model. We then filter out only data that was assigned a strongly positive verdict (established as a predicted score of 0.99 or above). On this subset, we perform kMeans clustering over SHAP values. For each cluster, we compute purity with respect to the existing pipeline label as the fraction of cluster members with a pipeline score of 0.9 or above. Clusters with purity < 1 contain samples that were not originally convicted as spam. We use SHAP summary plots to understand how predictions pertaining to such impure clusters are generated to assess whether they are more likely novel TPs (or at least easily defensible) or FPs created by the new engine.

Table 5.8 contains details of features relevant to this experiment. Figure 5.9 contains two examples. 5.9a illustrates a cluster (93.43% purity) that is convicted due to activation of a very rare but extremely precise (with respect to the pipeline spam

verdicts) feature (ID 14). Prevalent features (IDs 2 and 4) that are negatively correlated with the pipeline spam verdict, and thus indicate legitimacy, are largely inactive. This suggests messages don not originate from well-known infrastructure. A domain expert would consider the prediction intuitive and defensible, and might be inclined to consider convictions as novel TPs. 5.9b relates to a cluster (95.21% purity) where predictions are based on inactivity of a rare feature (ID 6) that is largely uncorrelated with the pipeline verdict, and inactivity of extremely prevalent features (dmarc and spf). Mixed activity of a very prevalent feature (ID 2) suggests messages might originate from a variety of different infrastructure. Based on these characteristics, a domain expert might not be able to conclude that the cluster is safe to convict in its entirety.

5.6.2 Correlation of Model Interpretation Measures and Features

As a precursor to the next set of experiments, we investigate how measures outlined in section 2.4 and expanded in 4.3 correlate with each other. We also aim to understand how measures correlate with the nature of a features. We consider three characteristics: Prevalence *prev* is simply the ratio of a feature being active in the dataset. *corr₊* is the correlation between feature values and the existing pipeline spam verdict, *prec₊* specifies precision of the feature being active as it relates to the pipeline verdict. For the latter two characteristics, we applied the aggressive stance

(a) Convictions are intuitively justified by activity of highly spammy features. Samples that didn't have a pipeline spam verdict can be considered novel TPs and relabeled (ground truth updated).

(b) Conviction are largely due to absence of features that indicate legitimacy. They are harder to justify and samples should not be relabeled.

Figure 5.9: Red dots indicate the feature was active, blue indicates inactivity. Details for relevant features are summarized in table 5.8.

Feature	prevalence	spam correlation	spam precision
headerFrom.feature.14	0.05	0.23	1.00
envFrom.feature.14	0.05	0.23	1.00
headerReplyto.feature.14	0.02	0.15	1.00
rdns.feature.4	0.20	-0.15	0.35
headerFrom.feature.2	0.43	-0.12	0.44
headerFrom.feature.6	0.08	0.09	0.66
spf.pass	0.80	-0.05	0.50
dmARC.align	0.77	-0.01	0.51
headerReplyto.feature.4	0.04	0.05	0.62

Table 5.8: Features relevant to examples in figure 5.9. Prevalence is simply the ratio of the feature being active in the dataset. Correlation and precision are computed against the pipeline spam verdict (aggressive formulation with threshold score 0.5).

where both +, and ? signify a spam verdict.

Figure 5.10 summarizes our findings, computed for a LightGBM regression-classification model using the aggressive setting, in a heat map. Feature power in the *pathPow* formulation $\Phi_{pow:pp}$ is distinctly different from all other measures, including $\Phi_{pow:cn}$ (*cumNodePow*) and $\Phi_{pow:sn}$ (*strictNodePow*). It correlates strongly with $prec_+$ and $corr_+$, and negatively with $prev$ while all other measures correlate with negatively with $prec_+$ and $corr_+$ and positively with $prev$. $\Phi_{pow:cn,sn}$ strongly correlate with each other and bridge the gap to the third group consisting of TreeSHAP Φ_{shap} and gain-, and split importance $\hat{I}_{gain,split}$.

Figure 5.10: Correlation between various feature importance measures computed over a LightGBM regression-classification model using the aggressive setting, and feature characteristics.

5.6.3 Feature Selection Based on Model Interpretation

Variable importance measures can be used to rank features according to how critical they are in explaining predictions. One might be tempted to use that insight to train less-complex models using only top ranking features. In the following we investigate that approach and compare the results produced by the rankings of different model interpretation methods.

Figure 5.11 presents R^2 score, utility, precision, and recall of LightGBM regression-classification models in the aggressive setting, trained with increasing number of top-ranked features. Black horizontal lines indicate 90% of the final R^2 and the final precision scores that can be achieved when all features are used.

Panel 5.11a shows that rankings based on gain- and split importance $\hat{I}_{gain,split}$, TreeSHAP Φ_{shap} , and feature prevalence $prev$ achieve high R^2 scores rapidly and surpass 90% of final R^2 at 70 features. All three feature power formulations $\Phi_{pow:pp,cn,sn}$ reach 90% of final R^2 scores at around 120 features. $prec_+$ and $corr_+$ are the worst selection criteria, requiring almost all features. R^2 was selected since it reflects how well a model approximates the true score directly, without having to consider discrimination threshold dependent classification.

Panel 5.11c below superimposes application constraints by tuning each model to optimize utility in the aggressive regression-classification setting. We used parameters $\eta = 0.2, \lambda = 8$ imposing high penalties on FPs. We have zoomed in on development

over the first 60 features in the graph. While overall shape of each curve remains intact, we observe transformations with regards to the starting point and rate of change that leads to reordering. For example, while $\Phi_{pow:pp}$ performed worse than $\Phi_{pow:cn,sn}$ and Φ_{shap} with regards to R^2 , it outperforms all three ranking schemes in recovering utility over the first 35 features. Most drastic change can be observed for $corr_+$. While inherently not a robust ranking scheme for feature selection (as is illustrated in figure 5.12), it produces highest utility by a large margin using the first 45 features. Such insights, derived for a relevant application setting, may be exploitable when trying to find an appropriate ranking scheme.

The two panels on the right (5.11b, 5.11d) are zoomed in on models trained with up to 40 features only. Precision and recall are computed consistently using the aggressive discrimination threshold used by the pipeline (spam verdict when score exceeds 0.5) to highlight trends without local (per-model) tuning. Two behaviors can be distinguished in these results.

The first group starts conservatively with very high precision and low recall. This is most pronounced (as expected) for ranking by $prec_+$. $\Phi_{pow:pp}$ and $corr_+$ follow similar trends, though less pronounced.

Behavior for less than 5 features is very noisy. Beyond that, a second group of rankings can be distinguished that produce aggressive models with high recall but low precision early on. This group consists of $prev$, $\hat{I}_{split,gain}$, Φ_{shap} , and $\Phi_{pow:cn,sn}$.

After wide swings in recall and precision at very low number of features (< 5) models corresponding to these methods gradually recover precision with $\hat{I}_{gain}$ and Φ_{shap} leading by small margins. In recovering recall, $\Phi_{pow:cn,sn}$ behave similar to each other and distinctly different from other methods in this group. Models trained on rankings from $\Phi_{pow:cn,sn}$ retain higher recall early on but fall behind when more than the top 10 features are used. Recovery of the final score is also significantly slower than for other methods.

Robustness Against Selecting Highly Correlated Features

As we have seen, measures under review produce rankings that correlate with basic statistics. These are computed for each feature independently and thus do not take variable interactions into account. Two features that are exactly identical, that is, assume the same value in all data points, will share values for $prev$, $prec_+$, $corr_+$. Using those characteristics in a naïve feature selection scheme, since both features have the same rank, they would be selected together in spite of the fact that no additional information is gained by including both. Any sensible feature selection scheme must be robust against such degenerate cases.

Dummy features such as exact duplicates or logical negations (where each feature value of 1 is replaced with 0 and vice versa) contain no additional independent information (random chance aside for noisy features). It is obvious that a single

Figure 5.11: Characteristics for regression-classification models in aggressive setting trained with top-k features (x-axis) selected according to various ranking methods. Graphs on the left provide an overview. 5.11a shows R^2 scores for regression models. 5.11c shows utility for $\eta = 0.2, \lambda = 8$ on the same models used for regression-classification in the aggressive setting. Graphs on the right are zoomed in on the same regression-classification models with few features. 5.11b shows precision while 5.11d illustrates recall. Rankings from highly correlated TreeSHAP, split-, and gain importance produce similar models, with split gain generally trailing behind the other two. Ranking by feature prevalence seems competitive when the goal is a model with high true positive rate. Feature power (*pathPow*) produces highly precise models so is a good choice when FP prevention is key. Ranking according to spam precision produces highly precise models. These methods, while tempting for their simplicity, are susceptible to selecting highly correlated features which limits their practical use (see figure 5.12).

decision tree trained over binary features, would pick only one version of the feature: In contrast to real-valued features where different thresholds can be defined in separate splits, once a feature variant providing the best gain is selected in a split, subsequent splits have nothing to gain from using a very similar feature. In the GBM setting, this intuition does not hold. While each tree would select only one of the feature variants, there is no guarantee that different trees would select the same subset of variants. In aggregate, the variants might show up as equally important for a model. CEGB, penalizing the first use of each feature in the tree ensemble but allowing subsequent use for free, should help avoid this scenario. We thus train two model variants, GBMs with and without CEGB penalty. Experiments are repeated 10 times and results are summarized in boxplots in figure 5.12. Top row (panels 5.12a, 5.12b) represents the base model, bottom (panels 5.12c, 5.12d) the CEGB variant.

To investigate how different model interpretation methods react to the presence of highly correlated features and different types of noise, we simulate two scenarios.

First, we create 45 noisy copies (dummy features) of the single top-ranked feature for each variable importance measure. We increase noise from 1% to 45% by randomly swapping the feature value in each data point with the specified probability. A robust method of feature selection would ensure that they select only few of the noisy copies. Specifically, very similar features should not be ranked close to each other. We note that, due to chance, a highly noisy feature might end up with

strong independent predictive power, but that should be rare. We compare the set of measures discussed in this work by counting the number of noisy dummy features ranked in the top 50 after retraining on the modified dataset. Results are reported in figures 5.12a and 5.12c. Split and gain importance are most susceptible to assigning high scores to dummies. $\Phi_{pow:pp}$ is most resilient, followed by $\Phi_{pow:cn,sn}$ and Φ_{shap} .

CEGB significantly improves resilience of models trained on rankings from all three feature power formulations $\Phi_{pow:pp,cn,sn}$ while TreeSHAP sees no such improvement. This is likely due to high correlation between TreeSHAP rankings and feature prevalence that might outweigh the CEGB penalty. Baselines for $prev$, and $corr_+$ act as expected. A noisy copy of a prevalent feature remains prevalent, same goes for correlation with the spam verdict. The fact that $prec_+$ does not select any of the noisy copies indicates that the dataset contains sufficient highly-precise features so that even low levels of introduced noise disqualify our duplicated copies. Given the apparent good results and simplicity of selecting models by precision, caution is due when using any rankings that utterly ignore mutual information in features. Also, as we have seen in previous studies, recall of models trained on only highly precise features can be prohibitively low.

Secondly, we start with creating an exact logical negation of top-5 features. We then create 4 noisy copies with noise probabilities of 1-4% of top 5 features and their negations. As in the first scenario, this results in 45 dummy features, much less noisy but spread over several features and simulating negative correlation. Results

of this setting are depicted in figures 5.12b and 5.12d. Lower and more subtle noise was expected make the models more susceptible to selecting dummy features. This turns out to not be the case in this scenario. In fact, TreeSHAP and split- and gain importance perform better on this type of noise. This is likely due to the fact that all three rank prevalent features highly. Logical negation makes dummy features rare and thus less likely to be considered important. Feature power appears largely agnostic and is comparably robust (similar to scenario 1). CEGB was less effective in protecting models from selecting noisy features in this scenario. This indicates that subtle noise produced more attractive dummy features which is in line with expectations.

Figure 5.12: For each method (x-axis), we investigate two noise-scenarios and two model variants. Graphs on top show results for regular GBMs, bottom for CEGB variants. Left panels illustrate a scenario where 45 increasingly noisy copies (noise level ranging from 1% to 45%) of only the top ranked feature for each variable importance method are introduced. Right shows results for more subtle noise: For top 5 features an exact logical negation is added. Then, 4 copies with noise ranging from 1% to 4% are created for both, top 5 features, and their negations. This again results in 45 dummy features in total. GBMs are retrained on the noisy dataset and features ranked again. y-axis indicates how many of the 45 noisy dummies get ranked among the top 50. Lower values indicate methods that are more resilient against the respective type of disturbance.

Chapter 6

Conclusion

6.1 Theoretical and Practical Contributions

This thesis contains three major contributions.

We outline methods that can be used in practice to reason about and improve performance of a spam detection pipeline in light of relevant business and application constraints. Specific focus has been placed on considerations relevant to prepending a new engine to an existing pipeline. This scenario has been reviewed from a monetary (business driven) point of view, as well as purely with regards to quality of classifications and pipeline efficiency. To that end, a way of modeling a prediction pipeline has been introduced, together with a parameterized utility function that can be used to optimize a distinct regression-classification predictor jointly over a set

of desired characteristics. We also define methods to establish thresholds that can be used to limit ROC-based analysis and tuning of models to only a subspace that is deemed worthwhile given application constraints. This allows to make quantitative decisions, not only about the best possible operation point of a proposed model, but also whether the model is worthwhile at all or if it should be simply omitted. The theoretical framework and the examples given in the experiments should be useful to practitioners and are likely transferable to any other pipeline-based setting in operations research.

High interest in model interpretation methods and the popularity of regression tree-based ensembles motivated a second theoretical contribution. Feature power, an existing methodology for model interpretation that was limited to classification trees and RandomForest ensembles, has been expanded to cover different types of machine learning models. We develop formulations for regression trees and gradient boosting machines with decision trees as base-learners, both in the regression and classification setting. The novel methods are application context agnostic and our contributions here are thus relevant to any ML task using the described models.

Lastly, we introduce a novel dataset representing features pertaining to domains associated with email senders, drawn over 7 days of real-world email traffic, and labeled by a state-of-the-art spam detection system. We validated and tested theoretical assertions on the dataset. Nature of the dataset has been described at a high level and investigated in detail to motivate and enable future research. Ex-

perimental results from training decision tree-based models indicate prevalence of very high-order feature interactions. Lack of overfitting and increased performance of very deep trees suggest relevant interactions of hundreds of features, while other dataset discussed in the literature often reach a performance plateau in ML models capturing interactions at the scale of less than ten features. As research into advancing explainability and interpretability of ML models becomes more popular, this type of data might prove useful.

6.2 Experimental Results

Contributions have been validated and used in a set of experiments to further understanding of how they can be applied in practice.

First, verdict and score distributions in the dataset have been illustrated and a roster of commonly used machine learning models have been trained and their performance compared. This allowed us to establish a baseline for the predictive power of the SDR features suggesting that they can reproduce existing pipeline verdicts with up to 75% accuracy. The result of this baseline experiment shows that GBM is consistently outperforming other standard models. We further investigated four popular GBM implementations in detail and compared their characteristics. LightGBM produces best results and is also the fastest GBM implementation by far. Practitioners that are required to choose a library may benefit from our review. We contrasted native

classification models with regression models used with a discrimination threshold (regression-classification models) and compared benefits of the approaches. We find that regression-classification produces good results and is most flexible since a single model can be tuned for different application constraints via ROC-curve analysis.

We observed a lack of overfitting on our data across models. We conclude that data is very uniform, leading to a lack of contrast between the training and validation set. We further assume that data contains high level of feature interactions that benefit deep decision trees.

We experimented with differentiated class weights to limit FP rate and find that moderate weight increase for samples in non-positive classes produces higher-utility models over a range of application constraints.

We proceeded with investigating several approaches that aim to reduce model complexity without disproportionate penalty to model performance. A generic approach using cost efficient gradient boosting and an application-centric approach that prunes a large portion of features informed by domain knowledge from spam detection have been investigated.

The CEGB approach is interesting since it allows to encode arbitrary notions of feature cost to produce models that are both faster and operate on fewer features. We show that prediction time and feature cost can be decreased but that either leads to lower accuracy. We show that model utility can be increased in spite

of lower accuracy when throughput is highly important. For application settings requiring high accuracy, CEGB might not strictly yield utility gains but may still be worthwhile when reduced feature cost is also considered.

For the second experiment we pruned roughly half of the features and investigated the effect on model utility. While this leads to reduced utility, the loss is shown to not be uniformly distributed across the full range of parameter settings specifying application constraints. To illustrate our findings, we produce a comprehensive map of utility parameters on FP and TP rate and the overall utility-loss. Our results highlight how design choices on whether and how to reduce model complexity can be made in practice based on quantitative analysis.

We then validated the novel feature power formulations for regression-classification GBMs. We show that the *pathPow* formulation gives high values to highly-precise features while *cumNodePow* and *strictNodePow* are very similar to each other and favor highly-prevalent features.

A last set of experiments has been performed to highlight how different model interpretation methods can be applied in several practical use-cases. We used clustering based on feature contributions to reason about the nature of clusters with mixed ground-truth labels. This allows propagating highly defensible verdicts to improve ground-truth on inconsistently (or semi-) labeled data.

We finalize experiments by illustrating how feature selection can be informed by

rankings from several variable importance measures. Characteristics of various ranking schemes and their influence on model performance as well as their susceptibility to degenerate features and noise have been investigated in different scenarios. Focusing on voting-theoretical methods, we find that rankings produced by TreeSHAP produce models with high recall that rapidly recover precision as features are added. This makes TreeSHAP a good choice for models employing a medium sized subset of features. Feature power in the *pathPow* formulation can be used to produce precise models with low recall on very few features making it a good choice for certain scenarios. All feature power methods demonstrate high resilience against duplicated or highly correlated features, especially when used in conjunction with CEGB.

6.3 Future Work

We hope that contributions in this thesis will inspire future research.

Our framework allows an engine to be inserted anywhere in an existing pipeline. We would like to perform more general pipeline optimization considering all possible insertion points, given more detail on distinct engine block rates and processing times.

We also limited our investigation to approximating an existing pipeline label. Future research should more systematically investigate how to handle label uncertainty in

our dataset to see if we can increase system level TPs. We highlight a way to improve ground-truth using clustering methods. Additional techniques should be developed to improve our dataset or to create a new spam detection dataset with true ground-truth. Also, while the nature of features in our dataset is not disclosed, there exist endless possibilities for deriving intelligence about an email sender. One such venue is given by introducing a feedback loop that propagates convictions from early data points to later ones pertaining to the same domains. Lastly, we limited our research to the binary features in our dataset. Other sources of information such as the (masked) top-level domains or novel features created from the data such as the number of distinct domains used in the transmission of a given message (just to give an example) should be investigated. Our experiments should be repeated on different and improved datasets to better validate our models.

We found that AdaBoost, RandomForest, Support Vector Machine, Naïve Bayes, and Perceptron performed poorly on our dataset. This is unexpected and the reasons should be researched.

The scope of our investigation is limited to spam detection. We see no reason the concepts should not work in other settings. Proposed pipeline optimization and interpretation methods should be validated in other problem domains. Further investigation is necessary to see if our design of the utility function, and modeling of business and application constraints therein, can be improved or adjusted for specific situations. We would like to perform case studies with real-world business

settings for our multi-faceted optimization framework to validate effectiveness of our method in a complex decision making process resulting in system design choices and business plans.

Our dataset is presented in form of binary feature vectors. We should theoretically investigate better optimization and analytic formulations that exploit the binary nature. Concurrently, our models should be validated with multi-level categorical, and continuous domain datasets.

We have formulated an extension of feature power to regression but have not systematically investigated our findings. Both, a more solid theoretical understanding, and experimental validation need to be established.

We surmise high-order feature interactions in our dataset. We suggest future research employing coalition power and similar concepts that highlight how groups of features contribute to model predictions. In this line of research, coalition power [22] should be expanded to regression tree-based models and GBMs.

Finally, we encourage investigation of feature selection employing more sophisticated methods, for example the Contribution-Selection algorithm (CSA) proposed in [17], in conjunction with voting-theoretical variable importance measures.

Appendix A: Example Data

For our experiments, we use one-hot-encoding to produce a binary data-matrix over features and use it with existing pipeline verdicts (see table 6.1). One-hot encoded data and raw, nested json (see page 123) are linked to from <https://github.com/JakobDohrmann/PredictionPipelineMS2020>. Files contain one data point per line (line-delimited JSON). The github repository also contains source code used to create data formats and the pandas DataFrame resulting from pre-processing and de-duplication (see section 3.1).

<i>verdict</i>	<i>score</i>	<i>dmarc.align</i>	<i>spf.align</i>	<i>envFrom.0</i>	<i>...</i>	<i>listUnsub.63</i>
–	0.000	1	1	0	...	0
–	0.003	1	1	1	...	0
?	0.773	0	1	1	...	0
+	0.997	0	0	0	...	1

Table 6.1: Example of one-hot encoded data. This format is used in our experiments.

The original dataset is presented in form of line-separated, nested json dictionaries.

The following is an example data point:

```
{ 'queryTS': 48563,
  'response': [ { 'field': 'envFrom', 'features': [0, 10],
                  'domain': { 'host': 292, 'registered': 139,
                              'suffix': 8, 'valid': True } },
                { 'field': 'headerFrom', 'features': [14],
                  'domain': { 'host': 195133, 'registered': 72899,
                              'suffix': 0, 'valid': True } },
                { 'field': 'headerReplyto', 'features': [14],
                  'domain': { 'host': 195133, 'registered': 72899,
                              'suffix': 0, 'valid': True } },
                { 'field': 'helo', 'features': [1, 0, 10],
                  'domain': { 'host': 4218, 'registered': 139,
                              'suffix': 8, 'valid': True } },
                { 'field': 'rdns', 'features': [14],
                  'domain': { 'host': 195134, 'registered': 72899,
                              'suffix': 0, 'valid': True } } ],
  'dkim': { 'align': True, 'valid': False }, 'dmarc': { 'align': True },
  'spf': { 'align': False, 'fromResult': 'pass',
           'heloResult': 'none' },
  'target_score': 0.9711451895933471, 'target_verdict': '+' }
```

- *target_verdict* represents the spam classification of an existing spam detection system.
- *target_score* provides higher granularity and is mapped to verdicts:
 $score < 0.5 \rightarrow -$, $0.5 \leq score < 0.9 \rightarrow ?$, $score \geq 0.9 \rightarrow +$
- *queryTS* represents the offset in seconds from the earliest query in the dataset.
- *dkim*, *dmARC*, *spf* specify whether the message passed any of these common authentication (sanctioning) schemes and how.
- *response* contains details and features pertaining to domains used by a sender:
 - *field* is the context (or location) that contained a domain. Either of:
 $rdns$, $helo$, $envFrom$, $headerFrom$, $headerReplyto$, $listUnsub$.
 - *domain.valid* specifies whether the value retrieved for the current field could be parsed as a valid domain.
 - *domain.fqdn* is the full host name, masked to protect PII.
 - *domain.registered* is the organizational domain, masked to protect PII.
 - *domain.suffix* is the top level domain, masked to protect PII.
 - *ruleHits* contains a list of rules (feeds or logic functions) associated with the domain in the current field (context). Names of rule hits are masked to protect IP. These *ruleHits* constitute the majority of features we use in our research.

Appendix B: Mathematical Symbols

Symbol	Meaning	First used
X	dataset	2.3.1
N	number of data points ($ X $)	2.3.1
$\mathbf{x}$	data point, model input or feature vector	2.3.1
y	data label (response, score, verdict)	2.3.1
D	set of input features	2.3.2
$ D $	number of features in each data point	2.4.2
f_j	feature	2.3.2
$\mathcal{F}(\mathbf{x})$	true function, mapping $\mathbf{x}$ to y	2.3.1
$F(\mathbf{x}, b_m, \mathbf{a}_m)$	(parameterized) model approximating $\mathcal{F}$	2.3.1
M	number of base-learners in ensemble	2.3.1
$h_m(\mathbf{x}, \mathbf{a}_m)$	base-learner output	2.3.1
f_0	naïve model prediction (bias)	2.3.1
$\hat{y}(\mathbf{x})$	predicted score	2.3.1
$\hat{c}(\mathbf{x})$	predicted class	2.3.1
K	number of classes	2.3.1
$c(c_{pred}, c_{true})$	cost of (mis-)classification	2.3.1
$\mathbb{1}(\cdot)$	1 if argument is true, else 0	2.3.1
T	decision tree	2.4.1
J	number of leaves in tree	2.3.2
n_t	node in a tree	2.3.2
$f(n_t)$	split variable in node n_t or value leaf	2.3.2
$r_m(\mathbf{x})$	scaled tree response in GBM	2.3.4

Symbol	Meaning	First used
γ	CEGB penalty for use of feature	2.3.5
α	CEGB penalty for each split	2.3.5
$\hat{I}$	variations of feature importance	2.4.1
$v(S)$	characteristic function of coalition	2.4.2
$S \in U$	coalition, usually a (sub-)path in a tree	2.4.2
U	coalition space	2.4.2
ϕ	(measure based on) Shapley value	2.4.2
Φ	aggregate over Shapley-based measures	2.4.3
$p = (n_i, n_j)$	path in tree	2.4.4
$d(p)$	edges traversed in path	2.4.4
$l(T)$	set of leaf nodes	2.4.4
$L(T, c_k)$	count of leaves associated with c_k	2.4.4
E_i	spam detection engine	4.1
C_i	cost of engine (operational, features, ...)	4.1
B_i	block rate of a spam detection engine	4.1
P_i	processing time of spam detection engine	4.1
V_i	input volume to spam detection engine	4.1
ζ	revenue per novel TPs	4.2.1
TCR	total cost ratio, weighs TPs vs FPs	4.2.2
λ	penalty multiplier for FPs	4.2.2
ρ	parameter tuned in ROC curve, <i>e.g.</i> discrimination threshold	4.2.2
u	utility function in ROC-space	4.2.2
η	weighting parameter in u	4.2.2

Table 6.2: Mathematical symbols and their meaning in this document.

Bibliography

- [1] N Abe, B Zadrozny, and J Langford, *An iterative method for multi-class cost-sensitive learning*, Proceedings of the 10th ACM SIGKDD international conference on knowledge discovery and data mining, 2004, pp. 3–11.
- [2] A AlMahmoud, E Damiani, H Otrók, and Y Al-Hammadi, *Spamdoop: A privacy-preserving big data platform for collaborative spam detection*, IEEE transactions on big data (2017).
- [3] D Alperovitch, P Judge, and S Krasser, *Taxonomy of email reputation systems*, Proceedings of the 27th international conference on distributed computing systems workshops (ICDCSW), IEEE, 2007, pp. 27–27.
- [4] I Androutsopoulos, J Koutsias, K V Chandrinos, G Paliouras, and C D Spyropoulos, *An evaluation of naive bayesian anti-spam filtering*, arXiv preprint cs/0006013 (2000).
- [5] Apache Software Foundation, *Spamassassin wiki: How scores are assigned*, <https://cwiki.apache.org/confluence/display/SPAMASSASSIN/HowScoresAreAssigned>, 2009, [Online; accessed 21-July-2020].
- [6] Apache Software Foundation, *Spamassassin wiki: Total cost ratio*, <https://cwiki.apache.org/confluence/display/SPAMASSASSIN/TotalCostRatio>, 2009, [Online; accessed 21-July-2020].
- [7] Apache Software Foundation, *Spamassassin introduction*, <https://spamassassin.apache.org/index.html>, 2019, [Online; accessed 21-July-2020].

- [8] Apache Software Foundation, *Spamassassin manual: Scoring options*, https://spamassassin.apache.org/full/3.1.x/doc/Mail_SpamAssassin_Conf.html#scoring_options, 2019, [Online; accessed 21-July-2020].
- [9] APWG, *Phishing activity trends report, q3/2020*, Tech. report, Anti-phishing working group, 2020.
- [10] J F Banzhaf III, *Weighted voting doesn't work: A mathematical analysis*, Rutgers Law Review **19** (1964), 317.
- [11] J Bian, XG Peng, Y Wang, and H Zhang, *An efficient cost-sensitive feature selection using chaos genetic algorithm for class imbalance problem*, Mathematical problems in engineering **2016** (2016), 1–9.
- [12] E Blanzieri and A Bryl, *A survey of learning-based techniques of email spam filtering*, Artificial intelligence review **29** (2008), no. 1, 63–92.
- [13] L Breiman, *Some infinity theory for predictor ensembles*, Tech. Report 579, Statistics department UCB, 2000.
- [14] L Breiman, *Random forests*, Machine learning **45** (2001), no. 1, 5–32.
- [15] L Breiman, J H Friedman, C J Stone, and R A Olshen, *Classification and regression trees*, CRC press, 1984.
- [16] G Caruana and M Li, *A survey of emerging approaches to spam filtering*, ACM computing surveys (CSUR) **44** (2008), no. 2, 1–27.
- [17] S Cohen, E Ruppín, and G Dror, *Feature selection based on the shapley value*, Proceedings of the 19th international joint conference on artificial intelligence, 2005, pp. 665–670.
- [18] D Crocker, T Hansen, and M Kucherawy, *Domainkeys identified mail (dkim) signatures*, STD 76, Internet requests for comments, September 2011.
- [19] J J Eberhardt, *Bayesian spam detection*, Scholarly horizons: University of minnesota, morris undergraduate journal **2** (2015), no. 1, 2.
- [20] M Felegyhazi, C Kreibich, and V Paxson, *On the potential of proactive domain blacklisting*, LEET **10** (2010), 6–6.

- [21] W Feng, J Sun, L Zhang, C Cao, and Q Yang, *A support vector machine based naive bayes algorithm for spam filtering*, Proceedings of the 35th IEEE international performance computing and communications conference (IPCCC), IEEE, 2016, pp. 1–8.
- [22] K Fotion, *Feature power: a new variable importance measure for random forests*, Master’s thesis, San Francisco State University, 2018.
- [23] J H Friedman, *Greedy function approximation: a gradient boosting machine*, Annals of statistics **29** (2001), no. 5, 1189–1232.
- [24] P Graham, *A plan for spam*, <http://www.paulgraham.com/spam.html>, 2002.
- [25] Y G Hu, C Guo, EWT Ngai, M Liu, and S Chen, *A scalable intelligent non-content-based spam-filtering framework*, Expert systems with applications **37** (2010), no. 12, 8557–8565.
- [26] L Huang, A D Joseph, B Nelson, B IP Rubinstein, and J D Tygar, *Adversarial machine learning*, Proceedings of the 4th ACM workshop on security and artificial intelligence, 2011, pp. 43–58.
- [27] IC, FBI, *Internet crime report*, Tech. report, Federal Bureau of Investigation, Internet Crime Complaint . . . , 2019.
- [28] A G K Janecek, W N Gansterer, and K A Kumar, *Multi-level reputation-based greylisting*, Proceedings of the 3rd international conference on availability, reliability and security, 2008, pp. 10–17.
- [29] A Karim, S Azam, B Shanmugam, K Kannoorpatti, and M Alazab, *A comprehensive survey for intelligent spam email detection*, IEEE Access **7** (2019), 168261–168295.
- [30] G Ke, Q Meng, T Finley, T Wang, W Chen, W Ma, Q Ye, and TY Liu, *Lightgbm: A highly efficient gradient boosting decision tree*, Advances in neural information processing systems, 2017, pp. 3146–3154.
- [31] W Z Khanir, M K Khan, F T B Muhaya, M Y Aalsalem, and HC Chao, *A comprehensive study of email spam botnet detection*, IEEE communications surveys & tutorials **17** (2015), no. 4, 2271–2295.

- [32] S Kitterman, *Sender policy framework (spf) for authorizing use of domains in email, version 1*, RFC 7208, Internet requests for comments, April 2014.
- [33] J Klensin, *Simple mail transfer protocol*, RFC 5321, Internet requests for comments, October 2008.
- [34] P Kosik, P Ostrihon, and R Rajabiun, *Ipv6 and spam*, Proceedings of the MIT spam conference, 2009.
- [35] M Kucherawy and E Zwicky, *Domain-based message authentication, reporting, and conformance (dmarc)*, RFC 7489, Internet requests for comments, March 2015.
- [36] B Kuchipudi, R T Nannapaneni, and Q Liao, *Adversarial machine learning for spam filters*, Proceedings of the 15th international conference on availability, reliability and security, 2020, pp. 1–6.
- [37] S M Lee, D S Kim, and J S Park, *Cost-sensitive spam detection using parameters optimization and feature selection*, Journal of universal computer science **17** (2011), no. 6, 944–960.
- [38] J Levine, *Dns blacklists and whitelists*, RFC 5782, Internet requests for comments, February 2010.
- [39] P Li, *Abc-boost: Adaptive base class boost for multi-class classification*, Proceedings of the 26th annual international conference on machine Learning, 2009, pp. 625–632.
- [40] Q Li, M Cheng, J Wang, and B Sun, *Lstm based phishing detection for big email data*, IEEE transactions on big data (2020).
- [41] D Lowd and C Meek, *Good word attacks on statistical spam filters*, Proceedings of the 2nd conference on email and anti-spam (CEAS), 2005.
- [42] S M Lundberg, G G Erion, H Chen, A DeGrave, J M Prutkin, B Nair, R Katz, J Himmelfarb, N Bansal, and SI Lee, *From local explanations to global understanding with explainable ai for trees*, Nature machine intelligence **2** (2020), no. 1, 2522–5839.

- [43] S M Lundberg, G G Erion, and SI Lee, *Consistent individualized feature attribution for tree ensembles*, arXiv preprint arXiv:1802.03888 (2018).
- [44] S M Lundberg and SI Lee, *A unified approach to interpreting model predictions*, Advances in neural information processing systems, 2017, pp. 4765–4774.
- [45] M3AAWG, *Email metrics program: Report #15–first, second and third quarter 2011*, Tech. report, Messaging, Malware and Mobile Anti-Abuse Working Group, 2011.
- [46] OECD, *Report of the oecd task force on spam*, Tech. Report 114, Organisation for Economic Co-operation and Development, 2006.
- [47] S Peter, R Diego, F A Hamprecht, and B Nadler, *Cost efficient gradient boosting*, Advances in neural information processing systems, 2017, pp. 1551–1561.
- [48] Z Qian, Z M Mao, Y Xie, and F Yu, *On network-level clusters for spam detection*, Proceedings of the network and distributed system security symposium (NDSS), 2010.
- [49] A Ramachandran, N Feamster, and S Vempala, *Filtering spam with behavioral blacklisting*, Proceedings of the 14th ACM conference on computer and communications security, CCS '07, 2007, p. 342–351.
- [50] P Resnick, *Internet message format*, RFC 5322, Internet requests for comments, October 2008.
- [51] P Resnick, K Kuwabara, R Zeckhauser, and E Friedman, *Reputation systems*, Communications of the ACM **43** (2000), no. 12, 45–48.
- [52] R E Schapire, *The boosting approach to machine learning: An overview*, Non-linear estimation and classification, Springer, 2003, pp. 149–171.
- [53] R Segal, J Crawford, J O Kephart, and B Leiba, *Spamguru: An enterprise anti-spam filtering system*, Proceedings of the 1st conference on email and anti-spam (CEAS), 2004.
- [54] S Seth and S Biswas, *Multimodal spam classification using deep learning techniques*, Proceedings of the 13th international conference on signal-image technology & internet-based systems (SITIS), IEEE, 2017, pp. 346–349.

- [55] L S Shapley, *A value for n-person games*, Contributions to the theory of games **2** (1953), no. 28, 307–317.
- [56] G Singaraju and B BH Kang, *Repuscore: Collaborative reputation management framework for email infrastructure*, Proceedings of the 21st large installations system administration conference (LISA), vol. 7, 2007, pp. 243–251.
- [57] S Sinha, M Bailey, and F Jahanian, *Shades of grey: On the effectiveness of reputation-based “blacklists”*, Proceedings of the 3rd international conference on malicious and unwanted software (MALWARE), 2008, pp. 57–64.
- [58] V Stakhov, *Rspamd introduction*, <https://rspamd.com/doc/index.html#introduction>, 2020, [Online; accessed 21-July-2020].
- [59] T Subramaniam, H A Jalab, and A Y Taqa, *Overview of textual anti-spam filtering techniques*, International journal of physical sciences **5** (2010), no. 12, 1869–1882.
- [60] J Sun, G Zhong, K Huang, and J Dong, *Banzhaf random forests: Cooperative game theory based random forests with consistency*, Neural networks **106** (2018), 20–29.
- [61] B Taylor, *Sender reputation in a large webmail service*, Proceedings of the 3rd conference on email and anti-spam (CEAS), 2006.
- [62] valli.org, *List of all rbls*, <http://multirbl.valli.org/list/>, 2020, [Online; accessed 21-July-2020].
- [63] Y Vorobeychik and M Kantarcioglu, *Adversarial machine learning*, Synthesis lectures on artificial intelligence and machine learning **12** (2018), no. 3, 1–169.
- [64] G L Wittel and S F Wu, *On attacking statistical spam filters*, Proceedings of the 1std conference on email and anti-spam (CEAS), 2004.
- [65] L Zhang, J Zhu, and T Yao, *An evaluation of statistical spam filtering techniques*, ACM Transactions on Asian Language Information Processing (TALIP) **3** (2004), no. 4, 243–269.

- [66] Y Zhou, M Kantarcioglu, B Thuraisingham, and B Xi, *Adversarial support vector machine learning*, Proceedings of the 18th ACM international conference on knowledge discovery and data mining (SIGKDD), 2012, pp. 1059–1067.