Why Is This an Outlier? Explaining Outliers by Submodular Optimization of Marginal Distributions

Pasha Khosravi[1] **Antonio Vergari**[2] **Guy Van den Broeck**[1]

[1]Computer Science Dept., University of California, Los Angeles
[2]School of Informatics, University of Edinburgh

Abstract

Detecting outliers is an important task in machine learning, since if left unchecked they could hinder performance of our models. We focus on finding the reason an instance is an outlier, i.e. by finding the subset of features that if ignored the rest of the input is not an outlier anymore. We formulate the problem as a constrained monotonic submodular optimization task thanks to key properties of marginal distributions. Additionally, we leverage probabilistic circuits, which enable tractable marginal queries for arbitrary subsets, to further speed up the subset selection algorithm. We showcase the ability of finding the outlier features in a variety of different corruption scenarios, and show that finding and fixing the outlier features can help in downstream tasks such as classification.

1 INTRODUCTION

In classical machine learning (ML) tasks we rely on the assumption a model is going to be evaluated at test time on data points drawn from the same distribution that generated the data it has been trained on, also called the independently and identically distributed (i.i.d.) assumption [Murphy, 2022]. Therefore *outlier* and *anomalous* points, that is samples that are assumed not to follow the same data distributions [Chandola et al., 2009], can have a considerable impact on performance of ML models at deployment time [Hodge and Austin, 2004]. It becomes clear that detecting these data points, being natural or adversarially generated [Yuan et al., 2019, Ilyas et al., 2019], early and dealing with them, e.g., by cleaning them [Chu et al., 2016, Liu et al., 2004, Song et al., 2017], is an important task that can positively influence the performances of many ML models: from supervised classifiers [Acuña and Rodriguez, 2004] to unsupervised generative models [Nalisnick et al., 2018].

Classically, these approaches assume that a data point is either an outlier or not (i.e., it is an inlier) and that all features contribute to this dichotomy. As such, when trying to clean it, some methods try to change all its features at once, e.g., for an image data point, all pixels are reset to the most likely frequency values [Song et al., 2017], or more simply they discard the data as a whole. Clearly, this "whole-or-nothing" approach to outlier cleaning can be wasteful as precious data points can be lost.

Furthermore, in many real-world scenarios *a data point is an outlier because only a subset of its features have values that are unlikely according to their marginal distributions*. Consider for example tabular data collecting personal data of patients such as their city and country of birth and their date of birth. One common way to create an outlier patient record is by accidentally entering an invalid combination of city and country (e.g. London, Canada instead of London, England) or mistyping the date format (e.g. "day/month/year" vs "month/day/year") [van den Burg et al., 2019]. Instead of discarding the whole patient record, if one were able to precisely detect these "outlier features", they could clean them to restore the record.

In this paper, we specifically focus on the task of locating the subset of the features that make an outlier an outlier. Specifically, we cast this feature selection problem in a principled probabilistic framework in which a generative model is iteratively queried for the marginal probability of feature subsets. We propose two formulations for this optimization problem that exploit the *supermodularity* and monotonicity of probabilitity measures and distill efficient greedy algorithms as constrained submodular maximization. One variant tries to find the feature subset that maximises the probability of inlier features while the other retrieves the subset of outlier features that minimize the probability.

Next, we investigate which probabilistic model class can efficiently support our requirement of querying a joint probability distribution multiple times for computing the marginals of different feature subsets. We find this class in framework

Accepted for the 5th Workshop on Tractable Probabilistic Modeling at UAI (TPM 2022).

of probabilistic circuits (PCs) [Vergari et al., 2020, Choi et al., 2020b], tractable computational graphs that encode expressive joint distributions [Peharz et al., 2020a, Liu et al., 2022]. PCs are well-suited for our purposes because, by imposing certain structural constraints over their graphs, we can guarantee to exactly compute the marginal distribution of any feature subset in time linear in the size of the PC [Choi et al., 2020b, Darwiche and Marquis, 2002]. Our proposed methodology can be successfully applied to detect different patterns of outlier features when applied to image data (Section 5). Additionally, we show that finding the outlier features and replacing them with more reasonable values could help in downstream tasks such as increasing accuracy of a neural network classifier on corrupted test data. In some cases, the accuracy gains on cleaned data w.r.t. that on corrupted were as high as 30 percent.

2 OUTLIER FEATURE DETECTION

Notation. We use uppercase letters X for random variables (features) and lowercase letters x for their value assignments. Similarly, we denote sets of features in bold uppercase $\mathbf{X}$ and their assignments in bold lowercase $\mathbf{x}$. In certain cases, we also use uppercase letters, for example S, to denote sets. We use subscripts to denote subset of features (and their assignment), for example $\mathbf{x}_{in}$ ($\mathbf{x}_{out}$) corresponds to subset of inlier (outlier) features for assignment $\mathbf{x}$.

Problem statement. In most ML scenarios we make the assumption that our data points $\mathbf{x}$ are i.i.d. drawn from a distribution $p(\mathbf{X})$. With this interpretation, an outlier $\mathbf{x}'$ is intuitively an instance that is not drawn from $p(\mathbf{X})$. This means that in expectation $\mathbf{x}'$ has zero or very low probability under $p(\mathbf{X})$. This is why one popular solution to mark outliers is to select all those instances $\mathbf{x}'$ whose probability, according to a parameterized probabilistic model $p_\theta(\mathbf{X}) \approx p(\mathbf{X})$, falls under a certain threshold t [Bishop, 1994], i.e., $p_\theta(\mathbf{x}') \leq t$. While other criteria to deem an instance an outlier exist Thudumu et al. [2020], Wang et al. [2019], Boukerche et al. [2020], classically all of them assume that the whole $\mathbf{x}'$ is either an outlier or inliner (e.g., $p_\theta(\mathbf{x}') > t$), and they do not care about the reason *why*.

In this paper, we go one step further and want to find an *explaination* why $\mathbf{x}'$ is an outlier. We assume that a hidden corruption process altered a subset of the features of $\mathbf{x}$, denoted as $\mathbf{x}_{out}$ and turned them into *outlier features*, that is features with low-probability under p_θ. Furthermore we assume that the rest of the features $\mathbf{x}_{in} = \mathbf{x} \setminus \mathbf{x}_{out}$ are inlier features, that is the marginal $\mathbf{x}_{in}$ has high probability under p_θ. From our assumptions it follows that if we are able to precisely detect $\mathbf{x}_{in}$ or equivalently $\mathbf{x}_{out}$ we can either try to "fix" an outlier instance by replacing $\mathbf{x}_{out}$ with some high-probability values for the corresponding features, or simply marginalize them out if the ML model that operates on the downstream task is able to deal with missing values. For example, if the classifier can marginalize unobserved values or it is possible to compute its expectation w.r.t. the distribution $p_\theta(\mathbf{x}_{out} \mid \mathbf{x}_{in})$ [Khosravi et al., 2019, 2020]).

Ideally, given a data point $\mathbf{x}$ that is likely an outlier, we would like to find a *minimal subset* of outlier features $\mathbf{x}_{out}$, without which, $\mathbf{x}_{in}$ is an inlier. We relax the constraint of finding a minimal subset into the problem of finding a subset of features with bounded cardinality k.

Definition 1 (Minimizing the probability of outlier features)**.** Let $\mathbf{x}$ be an outlier according to model p_θ and k be the maximum number of outlier features $\mathbf{x}_{out}$ that turn $\mathbf{x}$ into an outlier. Then $\mathbf{x}_{out}$ can be found by optimizing the following cardinality constraint optimization problem:

$$\mathbf{x}_{out} = \operatorname*{argmax}_{|S| \leq k} 1 - p_\theta(\mathbf{x}_S). \tag{1}$$

The intuition behind Eq. (1) is that in order to minimize the probability of an outlier, we want to maximise the complement of its probability, a quantity sometimes called the probabilistic margin [Krishnakumar, 2007]. However, Definition 1 does not tell us if and under which conditions Eq. (1) can be solved efficiently. The next section answers affirmatively, showing that the cardinality constrained problem in Eq. (1) can be efficiently approximated in $O(kn)$ time, where k is the subset cardinality, n is the maximum number of features. This can be achieved by noting that the probability measure p_θ is a *supermodular function* when defined over discrete features $\mathbf{X}$.

The above computational result, however, assumes that we can efficiently and exactly compute the marginals $p_\theta(\mathbf{x}_S)$ for all possible feature subsets $S \subseteq \{1, \ldots, n\}$. In Section 4, we discuss a class of efficient and expressive tractable computational graphs, called smooth and decomposable probabilistic circuits [Vergari et al., 2020, Choi et al., 2020b], that enable us to compute arbitrary marginals exactly in time linear in their size.

3 SUBMODULAR OPTIMIZATION FOR EFFICIENT OUTLIER FEATURE DETECTION

Submodular and supermodular functions naturally occur in many real world applications and hence their properties are a well studied topic, specially under the light that they enjoy efficient approximation algorithms with guarantees under different optimization settings Liu et al. [2020], Krause and Golovin [2014]. In this section, we give a quick background of submodular optimization relevant to our task.

Given a set of elements Ω, the *set function* $f : 2^\Omega \to \mathbb{R}$ assigns a value to each subset of Ω. Intuitively, a function is submodular if the marginal benefit of adding an specific element to the current set decreases as the set grows. Similarly,

a set function is supermodular if its negation is submodular. Additionally, a submodular function is monotone if adding new elements always increases the value. The next definitions ground these concepts in a more formal way.

Definition 2 (Submodularity)**.** Let Ω be a set, then the function $f : 2^{\Omega} \to \mathbb{R}$ is submodular iff for all A, B such that $A \subset B \subseteq \Omega$, for all $i \in \Omega \setminus B$, we have

$$f(A \cup \{i\}) - f(A) \geq f(B \cup \{i\}) - f(B).$$

Definition 3 (Supermodularity)**.** A set function $f : 2^{\Omega} \to \mathbb{R}$ is supermodular iff $-f$ is submodular.

Definition 4 (Monotonicity)**.** A set function $f : 2^{\Omega} \to \mathbb{R}$ is monotone iff for all A, B such that $A \subseteq B \subseteq \Omega$ then $f(A) \leq f(B)$.

Given a monotone submodular set function f, minimizing and maximizing $f(S)$ is trivial if there are no constraints, since $f(\Omega)$ is the maximum and the minimum is $f(\{i\})$ for some $i \in \Omega$. In the presence of cardinality constraints, such as our case in Eq. (1), greedily maximizing a monotone submodular function gives us a $1 - \frac{1}{e}$ approximation factor to the following optimization task Williamson [2019]:

$$\underset{S}{\operatorname{argmax}} \quad f(S) \quad \text{s.t.} \quad |S| \leq k. \tag{2}$$

We refer to appendix A.1 for more details about the greedy algorithm. In summary, the greedy algorithm needs $O(|\Omega| \cdot k)$ evaluations of $f(S)$.

To show that Eq. (1) enjoys these approximation guarantees, we need to show that $1 - p_\theta(\mathbf{x}_S)$ is a monotone submodular function, or alternatively that $p_\theta(\mathbf{x}_S)$ is a monotone supermodular function. We do this in the next section, where we will use the fact that the complement of a submodular function is still submodular

Lemma 1 (Complement)**.** *If $f : 2^{\Omega} \to \mathbb{R}$ is a submodular set function, then the complement function $g(A) = f(\Omega \setminus A)$ is also submodular.*

3.1 SUPERMODULARITY AND MONOTONICITY OF MARGINAL PROBABILITIES

In this section, we review the background for two desired properties of marginals of probability distributions: supermodularity and monotonicity. These properties help us formulate our subset selection problem as a monotone submodular optimization task which enjoys fast greedy algorithms with approximation guarantees.

Given a joint distribution $p(\mathbf{x})$ on all features, we can marginalize a subset of features by summing them out. For example, if we partition the features into two subsets S, S' then we can marginalize S' as follows [1]:

$$p(\mathbf{x}_S) = \int_{\text{val}(\mathbf{X}_{S'})} p(\mathbf{x}_S, \mathbf{x}_{S'}) \, d\mathbf{X}_{S'}.$$

[1] In case of discrete features, the integrals turn into summations.

Where $\text{val}(\mathbf{X}_{S'})$ is set of potential values for $\mathbf{X}_{S'}$. Now given an instance $\mathbf{x}$, we can treat the marginal probability function as a set function by defining $f(S) = p(\mathbf{x}_S)$. If p is a probability measure, then we can show that f will be supermodular:

Lemma 2 (Supermodularity of probability measures)**.** *Given an assignment $\mathbf{x}$ and a probability measure p, the marginal probability function $f : 2^{\mathbf{X}} \to \mathbb{R}$ such that $f(S) = p(\mathbf{x}_S)$ is supermodular.*

Lemma 3 (Monotonicity of probability measures)**.** *Let A, B be two events such that $A \subseteq B$, and p be a probability measure, then $p(A) \leq p(B)$.*

3.2 OTHER GREEDY APPROXIMATIONS (WITHOUT GUARANTEES)

In the following, we explore an alternative formulation of our outlier feature selection problem. We start from the intuition that a subset of features is an outlier given the specific values of the remaining features.

Definition 5 (Minimizing marginal conditional probability of outlier features)**.** Let $\mathbf{x}$ be an outlier according to model p_θ and k be the maximum number of outlier features $\mathbf{x}_{out}$ that turn $\mathbf{x}$ into an outlier given the features $\mathbf{x}_{in} = \mathbf{x} \setminus \mathbf{x}_{out}$. They can be found by minimizing the following cardinality constraint optimization problem:

$$\mathbf{x}_{out} = \underset{|S| \leq k}{\operatorname{argmin}} \, p_\theta(\mathbf{x}_S \mid \mathbf{x} \setminus \mathbf{x}_S). \tag{3}$$

Interestingly, solving the minimization problem of Eq. (3) can be done by finding the set of features that *maximises* the probability of a data point of being an inliner.

Proposition 1. *Let $\mathbf{x}$ be an outlier according to model p_θ and k be the maximum number of outlier features $\mathbf{x}_{out}$ that turn $\mathbf{x}$ into an outlier. Solving the cardinality constraint minimization problem of Eq.* (3) *is equivalent to solving the following maximization problem:*

$$\mathbf{x}_{out} = \underset{|S| \leq k}{\operatorname{argmax}} \, p(\mathbf{x} \setminus \mathbf{x}_S) \tag{4}$$

The proof is in appendix B. Unfortunately, this problem does not enjoy the $1 - 1/\epsilon$ greedy approximation as Eq. (1), because $p(\mathbf{x} \setminus \mathbf{x}_S)$ is supermodular and not submodular. Nevertheless, we still use the greedy algorithm and as we see in Section 5 it produces reasonable results in practice.

4 PROBABILISTIC MODELS FOR TRACTABLE MARGINALS OF ARBITRARY SUBSETS

Our algorithms are model-agnostic: one can model p_θ with any family of generative models that can provide fast and ac-

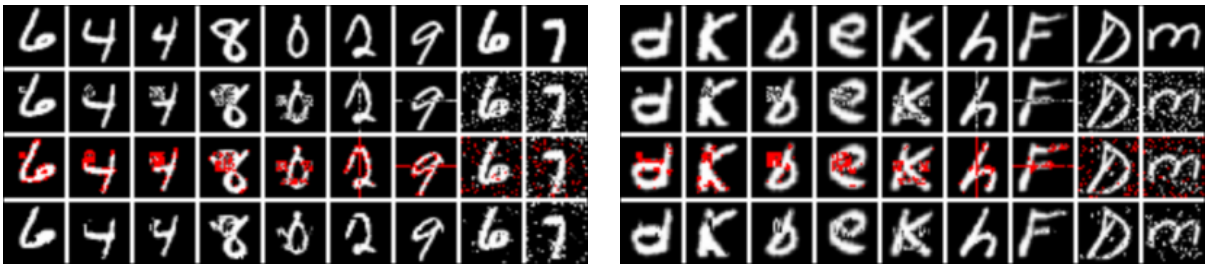

Figure 1: **Effective outlier feature detection with inlier maximization with a fixed budget.** Each row corresponds to (top to bottom): original images, corrupted images, outliers found with budget $k = 60$ (denoted with red pixels), and images that have been cleaned by the generative model. Each column corresponds to a different corruption process (left to right): Square(3x3), Square(5x5), Square(7x7), Square(9x9), Smiley, Column(14), Row(14), Random(120), Random(200).

curate approximation of marginal probabilities for arbitrary subset of features. For example, one can use sampling to approximate marginals of Variational Auto Encoder (VAEs), and some variants of VAEs such as HiVAE even provide a fast approximate marginals with one feed-forward evaluation Nazabal et al. [2018]. However, the quality of the solutions of the greedy approximation algorithms can quickly degrade if the internal routine to compute marginals delivers unreliable approximations. In the following we focus on a class of deep generative models that can be as expressive as VAEs [Liu et al., 2022] but at the same time guarantee the *exact* computation of any marginals in linear time. These are probabilistic circuits (PCs).

More background on PCs is in appendix A.2. In summary, we can compute any marginals exactly in time linear to the size of the PC. In addition, the computation of marginals is highly parallizable and can be accelerated by GPUs. For example, for our experiments, and $k = 60$, finding the outlier subset takes about 0.5 - 0.75 seconds per image.

5 EXPERIMENTS

In this section, we showcase a few experiments to answer the questions: i) Is the method decent at finding outlier features in different corruption scenarios? ii) Can finding and fixing a subset of outlier features help in downstream tasks such as classification?

For a fair comparison, we do not want to tell our method exactly how many outliers they are, so we choose a fixed budget for k. Unless otherwise noted, we choose $k = 60$ throughout our experiments. For more experiment details such as datasets, preprocessing steps, corruption scenarios, and how to fix the outliers we refer to appendix C.

Tables 2 and 3 (in appendix) summarize the results for finding the outlier features in each scenario and optimization method. For each dataset, the first column (%) shows the percentage of corrupted found successfully, and the second column ("Found") shows the average number of corrupted features found by the algorithm. As we see, given our fixed budget of $k = 60$, both methods successfully find high percentage of outliers when there is less than 60 outliers. In general, the maximizing inliers method does better at finding the outliers specially in bigger patch shapes, almost all of the budget is correctly used on finding corrupted features. Figure 1 provides few a image examples of our process by showing the clean images randomly samples from the test data, their corruption (one column for each corruption type), outliers found by our method, and the corresponding fixed image. More image examples can be found in the appendix.

We record the accuracy before and after fixing the images, results are summarized in Tables 4 and 5 (in appendix). In except a few cases, our method of finding and fixing outliers gives a decent boost to classification accuracy of the classifier on the corrupted test data. In some cases, we see performance boosts as high as 30% better accuracy. However, our methods do not always increase the classification accuracy, noticeably for bigger 9x9 square patches of MNIST dataset we even see a noticeable decrease in accuracy after replacing the outlier features. One reason could be due to having more corruptions (81) than our budget. Upon closer inspection of the fixed images for that scenario we noticed that the outlier detection does a decent job at finding the outlier features, however during imputation the outlier feature we did not find mess up the imputations. One might solve this issue by using more complex methods for fixing the outliers such as in-painting methods.

6 CONCLUSION

We introduced two novel algorithms for the challenging problem of finding the subset of outlier features by taking advantage of supermodularity and monotonicity of marginal probability distributions to formulate the problem as an optimization task that can be greedily optimized. We empirically show that the proposed methods are promising at finding and explaining outliers, and can help increase accuracy of classifiers on corrupted data. We hope this paper will motivate more explorations on taking advantage of marginal likelihoods in finding outlier features.

References

Edgar Acuña and Caroline Rodriguez. On detection of outliers and their effect in supervised classification. *University of Puerto Rico at Mayaguez*, 15, 2004.

Christopher M Bishop. Novelty detection and neural network validation. *IEE Proceedings-Vision, Image and Signal processing*, 141(4):217–222, 1994.

Azzedine Boukerche, Lining Zheng, and Omar Alfandi. Outlier detection: Methods, models, and classification. *ACM Comput. Surv.*, jun 2020. ISSN 0360-0300. doi: 10.1145/3381028. URL https://doi.org/10.1145/3381028.

Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch, 2017. URL https://arxiv.org/abs/1712.09665.

Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41 (3):15:1–15:58, July 2009. ISSN 0360-0300.

Ping-yeh Chiang*, Renkun Ni*, Ahmed Abdelkader, Chen Zhu, Christoph Studor, and Tom Goldstein. Certified defenses for adversarial patches. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HyeaSkrYPH.

YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models. oct 2020a. URL http://starai.cs.ucla.edu/papers/ProbCirc20.pdf.

YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic modeling. 2020b.

Xu Chu, Ihab F Ilyas, Sanjay Krishnan, and Jiannan Wang. Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 international conference on management of data*, pages 2201–2206, 2016.

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: an extension of mnist to handwritten letters, 2017. URL https://arxiv.org/abs/1702.05373.

Meihua Dang, Antonio Vergari, and Guy Van den Broeck. Strudel: A fast and accurate learner of structured-decomposable probabilistic circuits. *International Journal of Approximate Reasoning*, 140:92–115, 2022.

Adnan Darwiche. A differential approach to inference in bayesian networks. *J.ACM*, 2003.

Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial intelligence review*, 22(2):85–126, 2004.

Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. *Advances in neural information processing systems*, 32, 2019.

Rishabh Iyer, Ninad Khargoankar, Jeff Bilmes, and Himanshu Asanani. Submodular combinatorial information measures with applications in machine learning. In Vitaly Feldman, Katrina Ligett, and Sivan Sabato, editors, *Proceedings of the 32nd International Conference on Algorithmic Learning Theory*, volume 132 of *Proceedings of Machine Learning Research*. PMLR, 2021. URL https://proceedings.mlr.press/v132/iyer21a.html.

Pasha Khosravi, YooJung Choi, Yitao Liang, Antonio Vergari, and Guy Van den Broeck. On tractable computation of expected predictions. In *Advances in Neural Information Processing Systems*, pages 11169–11180, 2019.

Pasha Khosravi, Antonio Vergari, YooJung Choi, Yitao Liang, and Guy Van den Broeck. Handling missing data in decision trees: A probabilistic approach. In *Proceedings of The Art of Learning with Missing Values, Workshop at ICML*, 2020.

Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Probabilistic sentential decision diagrams. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, July 2014. URL https://lirias.kuleuven.be/bitstream/123456789/444268/1/kisa-kr14.pdf.

Andreas Krause and Daniel Golovin. Submodular function maximization. *Tractability*, 3:71–104, 2014.

Anita Krishnakumar. Active learning literature survey. *Tech. rep., Technical reports, University of California, Santa Cruz.*, 42, 2007.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*,

KDD '07. Association for Computing Machinery, 2007. ISBN 9781595936097. URL https://doi.org/10.1145/1281192.1281239.

Alexander Levine and Soheil Feizi. (de)randomized smoothing for certifiable defense against patch attacks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6465–6475. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/47ce0875420b2dbacfc5535f94e68433-Paper.pdf.

Anji Liu and Guy Van den Broeck. Tractable regularization of probabilistic circuits. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, dec 2021a. URL http://starai.cs.ucla.edu/papers/LiuNeurIPS21.pdf.

Anji Liu and Guy Van den Broeck. Tractable regularization of probabilistic circuits. *Advances in Neural Information Processing Systems*, 34, 2021b.

Anji Liu, Stephan Mandt, and Guy Van den Broeck. Lossless compression with probabilistic circuits. *International Conference of Learning Representations*, 2022.

Hancong Liu, Sirish Shah, and Wei Jiang. On-line outlier detection and data cleaning. *Computers & chemical engineering*, 28(9):1635–1647, 2004.

Yajing Liu, Edwin KP Chong, Ali Pezeshki, and Zhenliang Zhang. Submodular optimization problems and greedy strategies: A survey. *Discrete Event Dynamic Systems*, 30(3):381–412, 2020.

Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrak, and Andreas Krause. Lazier than lazy greedy, 2014.

Kevin P Murphy. *Probabilistic machine learning: an introduction*. MIT press, 2022.

Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan. Do deep generative models know what they don't know? *arXiv preprint arXiv:1810.09136*, 2018.

Alfredo Nazabal, Pablo M. Olmos, Zoubin Ghahramani, and Isabel Valera. Handling incomplete heterogeneous data using vaes, 2018. URL https://arxiv.org/abs/1807.03653.

Weili Nie, Brandon Guo, Yujia Huang, Chaowei Xiao, Arash Vahdat, and Anima Anandkumar. Diffusion models for adversarial purification. In *International Conference on Machine Learning (ICML)*, 2022.

Robert Peharz, Steven Lang, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Guy Van den Broeck, Kristian Kersting, and Zoubin Ghahramani. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *International Conference of Machine Learning*, 2020a.

Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Xiaoting Shao, Martin Trapp, Kristian Kersting, and Zoubin Ghahramani. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In *Uncertainty in Artificial Intelligence*, pages 334–344. PMLR, 2020b.

Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690. IEEE, 2011.

Tahrima Rahman, Prasanna Kothalkar, and Vibhav Gogate. Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of chow-liu trees. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 630–645. Springer, 2014.

Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. *arXiv preprint arXiv:1710.10766*, 2017.

Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJUYGxbCW.

Srikanth Thudumu, Philip Branch, Jiong Jin, and Jugdutt Jack Singh. A comprehensive survey of anomaly detection techniques for high dimensional big data. *Journal of Big Data*, 7(1):1–30, 2020.

Gerrit JJ van den Burg, Alfredo Nazábal, and Charles Sutton. Wrangling messy csv files by detecting row and type patterns. *Data Mining and Knowledge Discovery*, 33(6): 1799–1820, 2019.

Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. Simplifying, regularizing and strengthening sum-product network structure learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 343–358. Springer, 2015.

Antonio Vergari, YooJung Choi, Robert Peharz, and Guy Van den Broeck. Probabilistic circuits: Representations, inference, learning and applications. In *Tutorial at the The 34th AAAI Conference on Artificial Intelligence*, 2020.

Hongzhi Wang, Mohamed Jaward Bah, and Mohamed Hammad. Progress in outlier detection techniques: A survey. *Ieee Access*, 7:107964–108000, 2019.

David P. Williamson. Bridging continuous and discrete optimization, 2019. URL `https://people.orie.cornell.edu/dpw/orie6334/lecture23.pdf`.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems*, 30(9):2805–2824, 2019.

A BACKGROUND

A.1 SUBMODULAR OPTIMIZATION

Greedy algorithm In the presence of cardinality constraints, such as our case in Eq. (1), greedily maximizing a monotone submodular function gives us a $1 - \frac{1}{e}$ approximation factor to the following optimization task Williamson [2019]:

$$\underset{S}{\operatorname{argmax}} \quad f(S) \quad \text{s.t.} \quad |S| \leq k. \tag{5}$$

The algorithm starts with an empty sets $S = \varnothing$, and at each step chooses the "best" element to add to S until $|S| = k$, hence we need $O(|\Omega| \cdot k)$ evaluations of $f(S)$. There are many other variants of the greedy algorithm such as lazy greedy Leskovec et al. [2007], or even lazier Mirzasoleiman et al. [2014] that provide similar approximate guarantees in expectation but with much less evaluations of $f(S)$. However, as we see in Section 4, the lazier approaches might not be easily adaptable to our use case to provide better runtime performance than the basic greedy algorithm. We leave adaption of lazier approaches for future work.

A.2 PROBABILISTIC CIRCUITS

Probabilistic Circuits (PCs) Choi et al. [2020a] are a family of models that unify representation of many families of tractable probabilistic models such as arithmetic circuits Darwiche [2003], probabilistic sentential decision diagrams Kisa et al. [2014], and sum product networks Poon and Domingos [2011].

Definition 6 (Probabilistic circuits)**.** A probabilistic circuit p_θ over variables $\mathbf{X}$ is a parameterized computational graph that defines a joint probability distribution over $\mathbf{X}$. Its structure consists of three types of units: sum, product and distribution units. Each unit n defines a joint probability distribution over its scope, i.e., the set of variables it is defined on and recursively defined as: $\phi(n) = \bigcup_{c \in \mathsf{ch}(n)} \phi(c)$, where $\mathsf{ch}(n)$ denote the set of children, or inputs, of an inner unit n. The output of unit n in the circuit on input $\mathbf{x}$ is:

$$p_n(\mathbf{x}) = \begin{cases} g_n(\mathbf{x}) & \text{if } n \text{ is a distribution unit} \\ \prod_{c \in \mathsf{ch}(n)} p_c(\mathbf{x}) & \text{if } n \text{ is a product unit} \\ \sum_{c \in \mathsf{ch}(n)} \theta_{n,c} \cdot p_c(\mathbf{x}) & \text{if } n \text{ is a sum unit} \end{cases}$$

where $\theta_{n,c} > 0$, is the parameter associated to the edge (n, c). For an input distribution unit n, $g_n(\mathbf{x})$ is usually a simple distribution defined on one of the features, for example Categorical or Gaussian. Finally, the likelihood $p(\mathbf{x})$ defined by the circuit is the output of its root unit.

The remarkable property of PCs is the ability to compute *many complex functions of p in polytime if their computational graphs satisfy certain structural properties*. As we are interested in efficient and exact marginals, we will require only two structural properties: *smoothness* and *decomposability*.

Definition 7 (Smoothness & Decomposability)**.** A circuit is *smooth* if for every sum unit n, its inputs depend on the same variables: $\forall\, c_1, c_2 \in \mathsf{ch}(n), \phi(c_1) = \phi(c_2)$. It is *decomposable* if the inputs of every product unit n depend on disjoint sets of variables: $\mathsf{ch}(n) = \{c_1, c_2\}, \phi(c_1) \cap \phi(c_2) = \varnothing$.

Proposition 2 (Tractable marginalization, [Choi et al., 2020b])**.** *Let p be a smooth and decomposable circuit over $\mathbf{X}$ with input distributions that can be tractably marginalized. Then for any variables $\mathbf{Y} \subseteq \mathbf{X}$ and their assignment $\mathbf{y}$, the marginal $\int_{\mathbf{z} \in val(\mathbf{Z})} p(\mathbf{y}, \mathbf{z}) d\mathbf{Z}$ can be computed exactly in $\Theta(|p|)$ time, where $\mathbf{Z}$ denotes $\mathbf{X} \setminus \mathbf{Y}$.*

Smooth and decomposable PCs are both expressive and efficient: they can encode distributions with hundred millions of parameters and be effectively learned by gradient ascent [Peharz et al., 2020b]. The structure of their computational graph can be either specified manually [Poon and Domingos, 2011, Peharz et al., 2020b,a] or acquired automatically from data [Vergari et al., 2015, Rahman et al., 2014, Dang et al., 2022], e.g., by first learning a latent tree model and then compiling the latter into a circuit [Liu and Van den Broeck, 2021b]. These circuits are competitive with intractable models such as variational autoencoders and normalizing flows scores on several benchmarks [Liu et al., 2022].

Furthermore, one advantage PCs provide is that at each round of the greedy search we do $O(n)$ independent calls to marginals, so we can batch the queries together and compute in parallel using GPUs. The size of the circuit is constant and controllable by us, hence the main bottleneck is the number of outlier k. However, this limitation might not be a big issue

in practice since we assumed k is relatively small compared to n, and the computation cost only grows linearly w.r.t. k. For example, for our MNIST experiments, and $k = 60$, finding the outlier subset takes about 0.5 - 0.75 seconds per image (depending on the dataset) which seems reasonable. One potential avenue for getting an algorithm with runtime independent of k is exploring continuous relaxation of our submodular optimization tasks.

A.3 RELATED WORK

The idea of using generative models for fixing adversarial perturbations of images to improve accuracy of classifiers has been widely explored, for example PixelDefend Song et al. [2018] uses PixelCNN models to fix outlier images by bringing closer to the original distribution, more recently DiffPure Nie et al. [2022] uses diffusion models to purify adversarial perturbations. In many of these approaches, the assumption is that the perturbations do not change pixel values by too much, however they generally allow all features to be changed slightly. In contrast, we allow arbitrary changes to pixel values but limit the number of corrupted pixels.

Taking advantage of Submodularity of information theoretic measures such as entropy and mutual information in machine learning has been explored before Iyer et al. [2021]. To the best of out knowledge, we are the first to find applications for submodularity of marginal probability distributions in machine learning scenarios.

Both defences and attacks on fixed shape patches (such as squares) with arbitrary perturbations of pixels has been widely explored Levine and Feizi [2020], Chiang* et al. [2020], Brown et al. [2017].

B PROOFS

Proof of Proposition 1

Proof. First we can use Bayes rule to rewrite the conditional probability $p(\mathbf{x}_S \mid \mathbf{x} \setminus \mathbf{x}_S)$, thus obtaining

$$\begin{aligned}\mathbf{x}_{out} &= \operatorname*{argmin}_{|S|\leq k} p(\mathbf{x}_S, \mathbf{x} \setminus \mathbf{x}_S)/p(\mathbf{x} \setminus \mathbf{x}_S) \\ &= \operatorname*{argmin}_{|S|\leq k} p(\mathbf{x})/p(\mathbf{x} \setminus \mathbf{x}_S).\end{aligned}$$

Then we can invert the fraction and turn the minimization into a maximization problem. Finally, by noting that $p(\mathbf{x})$ is constant in our setting, we retrieve Eq. (4):

$$\begin{aligned}\mathbf{x}_{out} &= \operatorname*{argmin}_{|S|\leq k} p(\mathbf{x})/p(\mathbf{x} \setminus \mathbf{x}_S) \\ &= \operatorname*{argmax}_{|S|\leq k} p(\mathbf{x} \setminus \mathbf{x}_S)/p(\mathbf{x}) \\ &= \operatorname*{argmax}_{|S|\leq k} p(\mathbf{x} \setminus \mathbf{x}_S).\end{aligned}$$

□

C EXPERIMENT DETAILS

Table 1: Datasets

Name	# Features	# Train	# Test
MNIST	784	60000	10000
FASHION	784	60000	10000
EMNIST Letters	784	124800	20800

Setup In all the experiments, we only utilize one CPU core and one GPU (NVIDIA A5000). We choose three datasets MNIST Lecun et al. [1998], EMNIST Letters Cohen et al. [2017], and FASHION Xiao et al. [2017] to test generality of our method. For each dataset we use the default train and test splits provided. Table 1 provides more info about each dataset.

Preprocessing Steps For each dataset we assume there was no corruptions in the training data. We learn a probabilistic circuit using hidden Chow-Liu Tree (HCLT) algorithm Liu and Van den Broeck [2021a]. Additionally, we train a classifier for each dataset using Convolutional Neural Network architecture LeCun et al. [1998]. Training the circuits are a one time cost for each dataset taking about 5-10 minute each.

Corruption Scenarios We try a wide variety of corruption patterns to showcase generality of our methods. Square(ixi) refers to a square corruption patch of size i by i relatively close to middle of the image. Row(i) or Column(j) refers to corrupting i'th row or j'th column respectively. Random(i) refers to choosing random pixels to corrupt repeated for i times. Smiley refers to corruption consisting of 2 squares for the eyes and one wide rectangle for the mouth.

Fixing Outlier Features In certain downstream tasks such as classification, in addition to finding the outlier features, we might need to also fix the outlier features. The first step is to find the outliers using either methods from Section 2. Since the main focus of the paper is finding the outlier subset, we use a fairly simple multiple imputation method to fix the the outlier features. We reuse the probabilistic circuit we learned from the data to generate the multiple imputations,

since PCs also allow for fast conditional sampling of the form $p(\mathbf{X}_{out} \mid \mathbf{x}_{in})$. We do 100 conditional samples for each corrupted image and replace the outlier features with the sampled features from the circuit, then we input all the 100 samples into our classifier and use the average logits to perform the classification. We record the accuracy before and after fixing the images, results are summarized in Tables 4 and 5.

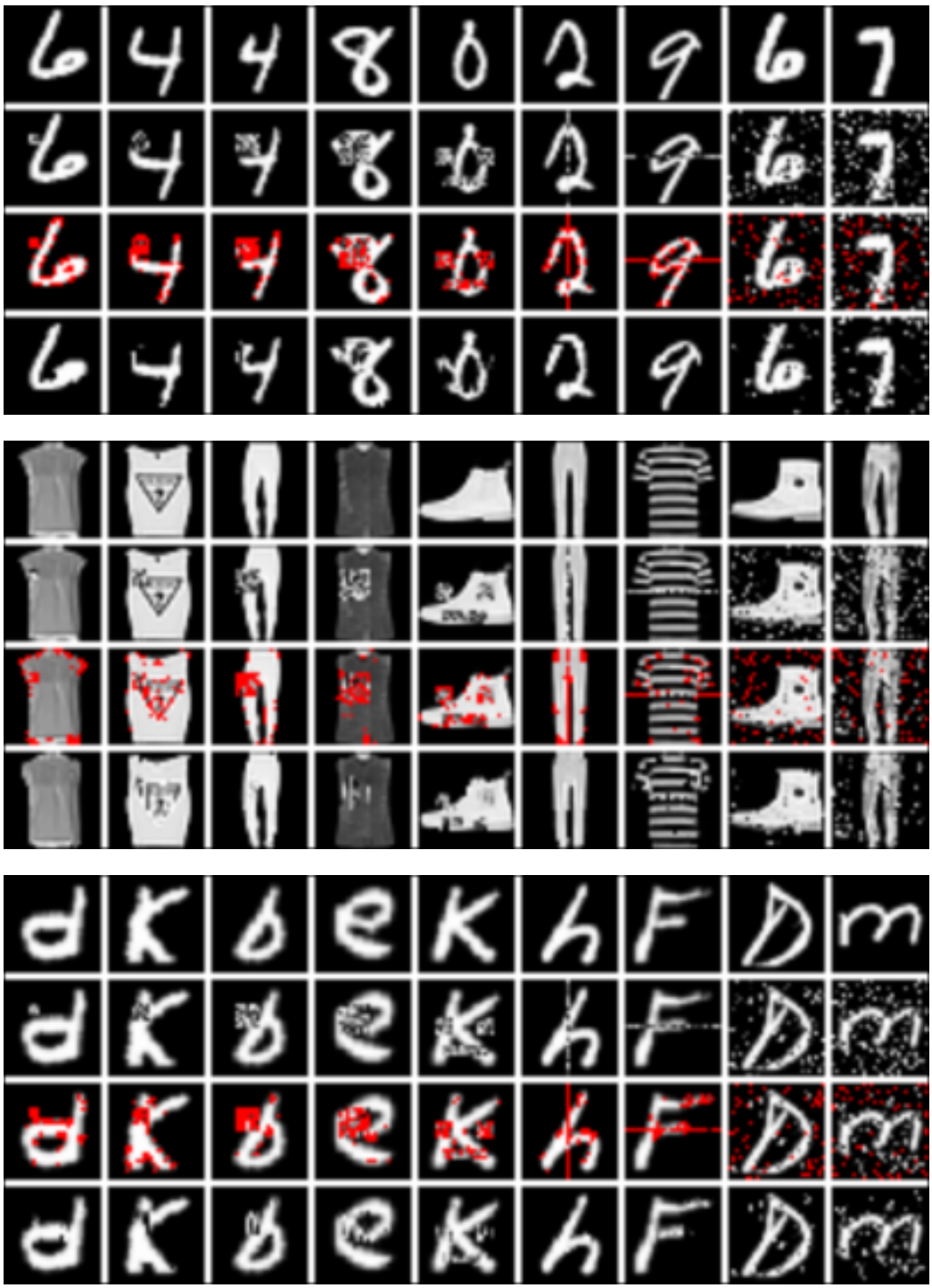

Figure 2: **(Similar to figure 1 but with larger images and more datasets) Effective outlier feature detection with inlier maximization with a fixed budget.** Each row corresponds to (top to bottom): original images, corrupted images, outliers found with budget $k = 60$ (denoted with red pixels), and images that have been cleaned by the generative model. Each column corresponds to a different corruption process (left to right): Square(3x3), Square(5x5), Square(7x7), Square(9x9), Smiley, Column(14), Row(14), Random(120), Random(200).

Table 2: Percentage and Avg Outliers found for Maximizing Marginals of Inliers ($k = 60$)

		MNIST		FASHION		EMNIST LETTERS	
CORRUPTION	OUTLIERS	%	FOUND	%	FOUND	%	FOUND
SQUARE(3X3)	9	97	8.7	70	6.3	99	8.9
SQUARE(5X5)	25	86	21.5	56	13.9	87	21.7
SQUARE(7X7)	49	69	33.7	51	24.9	76	37.2
SQUARE(9X9)	81	54	44.1	38	30.5	63	51.0
SMILEY	89	52	46.0	44	38.7	59	52.8
COLUMN(14)	28	88	24.7	62	17.3	93	25.9
ROW(14)	28	91	25.4	81	22.8	97	27.0
RANDOM(120)	110	55	59.0	52	57.0	54	59.9
RANDOM(200)	180	33	59.9	33	58.1	32	60.0

Table 3: Percentage and Avg Outliers found for Minimizing Marginals of Outliers ($k = 60$)

		MNIST		FASHION		EMNIST LETTERS	
CORRUPTION	OUTLIERS	%	FOUND	%	FOUND	%	FOUND
SQUARE(3X3)	9	83	7.5	73	6.5	81	7.3
SQUARE(5X5)	25	80	20.0	57	14.2	77	19.4
SQUARE(7X7)	49	71	35.0	43	20.9	67	32.8
SQUARE(9X9)	81	53	42.8	35	28.0	56	45.3
SMILEY	89	46	40.9	33	29.6	48	42.5
COLUMN(14)	28	72	20.2	41	11.5	87	24.4
ROW(14)	28	70	19.6	48	13.5	74	20.7
RANDOM(120)	110	40	43.6	31	35.1	41	44.8
RANDOM(200)	180	29	52.3	22	39.2	27	48.5

Table 4: Accuracy Gains for Maximizing Marginals of Inliers ($k = 60$)

	MNIST			FASHION			EMNIST LETTERS		
CORRUPTION	CORRUPT	FIXED	DIFF	CORRUPT	FIXED	DIFF	CORRUPT	FIXED	DIFF
SQUARE(3X3)	98.6	97.2	-1.4	83.5	86.1	**+2.6**	89.1	90.5	**+1.3**
SQUARE(5X5)	97.3	96.8	-0.5	79.1	82.1	**+3.0**	84.5	90.1	**+5.6**
SQUARE(7X7)	90.0	90.3	**+0.3**	79.8	79.4	-0.4	79.5	85.5	**+6.0**
SQUARE(9X9)	72.3	65.8	-6.6	76.0	75.9	-0.1	70.1	77.5	**+7.4**
SMILEY	84.2	83.9	-0.3	74.1	76.8	**+2.7**	60.4	74.7	**+14.3**
COLUMN(14)	97.7	97.6	-0.1	83.3	87.7	**+4.4**	86.2	90.3	**+4.1**
ROW(14)	96.2	98.1	**+1.9**	86.5	87.5	**+1.0**	83.4	91.0	**+7.6**
RANDOM(120)	87.7	97.3	**+9.6**	76.6	87.5	**+10.9**	54.7	86.6	**+31.8**
RANDOM(200)	72.6	81.9	**+9.3**	50.2	80.3	**+30.1**	25.9	46.2	**+20.4**

Table 5: Accuracy Gains for Minimizing Marginals of Outliers ($k = 60$)

	MNIST			FASHION			EMNIST LETTERS		
CORRUPTION	CORRUPT	FIXED	DIFF	CORRUPT	FIXED	DIFF	CORRUPT	FIXED	DIFF
SQUARE(3X3)	96.7	97.7	**+1.0**	86.5	88.3	**+1.8**	88.5	90.8	**+2.3**
SQUARE(5X5)	95.9	95.5	-0.4	83.0	84.5	**+1.5**	86.0	88.0	**+2.0**
SQUARE(7X7)	86.0	87.3	**+1.2**	75.4	76.9	**+1.5**	82.0	83.9	**+1.9**
SQUARE(9X9)	72.4	69.2	-3.2	74.1	74.3	**+0.2**	68.6	67.6	-1.0
SMILEY	79.4	80.4	**+1.1**	73.6	73.9	**+0.4**	57.9	59.2	**+1.3**
COLUMN(14)	97.2	97.4	**+0.2**	84.3	86.3	**+2.0**	87.5	91.3	**+3.8**
ROW(14)	93.7	97.7	**+4.0**	87.6	87.4	-0.2	79.8	89.5	**+9.7**
RANDOM(120)	94.8	96.8	**+2.0**	69.6	78.4	**+8.8**	54.7	70.1	**+15.4**
RANDOM(200)	85.5	87.8	**+2.3**	47.9	52.7	**+4.8**	20.3	25.1	**+4.8**