A Random Subspace Technique That Is Resistant to a Limited Number of Features Corrupted by an Adversary

Chris Mesterharm [1] **Rauf Izmailov** [1] **Scott Alexander** [1] **Simon Tsang** [1]

Abstract

In this paper, we consider batch supervised learning where an adversary is allowed to corrupt instances with arbitrarily large noise. The adversary is allowed to corrupt any l features in each instance and the adversary can change their values in any way. This noise is introduced on test instances and the algorithm receives no label feedback for these instances. We provide several subspace voting techniques that can be used to transform existing algorithms and prove data-dependent performance bounds in this setting. The key insight to our results is that we set our parameters so that a significant fraction of the voting hypotheses do not contain corrupt features and, for many real world problems, these uncorrupt hypotheses are sufficient to achieve high accuracy. We empirically validate our approach on several datasets including three new datasets that deal with side channel electromagnetic information.

1. Introduction

In this paper, we consider standard batch supervised learning with the additional assumption that an adversary can modify a subset of features during the evaluation/use of the algorithm's hypothesis. This type of analysis is useful since it gives a way to weaken the independent and identically distributed (iid) assumption of many theoretical machine learning results. Many practical problems cannot be accurately modeled with the iid assumption.

The feature corruption we consider is related to the popular adversarial technique where every feature is allowed to be shifted by a small amount (Szegedy et al., 2013). The difference is in how the features are perturbed. In general, the adversary can change instances in the test set by a bounded amount. Since the instance is a vector, a norm is often used to control this amount. If every feature is allowed to be changed then an infinity norm is used. In our case, we use the zero "norm" to allow l features to be changed to any value.

The zero norm is useful for a range of problems. For example, when dealing with categorical features, it often does not make sense to make small changes since there is no metric on the feature values. Another example is missing values. For real world problems, test/production data might be less thoroughly collected, causing more values to be missing. Again this is handled by our zero norm adversarial model since we do not care what value is used for the corrupt feature. While this paper primarily focuses on the zero norm, in Section 8, we briefly talk about other norms and how they relate to our techniques.

The main intuition of our technique is to use majority vote where more than half of the basic voting hypotheses do not use any corrupt features. This is essentially the random subspace method (Ho, 1998) with extra constraints on its parameters to ensure that a large fraction of the basic algorithms do not use corrupt features. In Section 6, we give voting experiments and parameter choices that often perform as well as algorithms using all the features but give much better performance as an adversary starts to corrupt features.

There has been significant work on these types of drift problems (Sugiyama & Kawanabe, 2012). In (Quionero-Candela et al., 2009), a SVM based optimization problem is given based on the assumption that features are removed by an adversary giving them a value of zero. Our adversary is more general in that it can generate any value for the corrupted features allowing it to greatly distort the prediction or hide the modifications. In (Bifet et al., 2012), a similar strategy of using few features per basic hypotheses is presented, but the technique uses a stacking approach that can give larger influence to hypotheses that might have been corrupted. They show experimentally that their technique is effective with their weaker adversarial assumptions. In (Biggio et al., 2010), intuitive arguments are given for ensemble methods, including the random subspace method, but again the results are experimental.

[1] Perspecta Labs, New Jersey, USA. Correspondence to: Chris Mesterharm <jmesterharm@perspectalabs.com>.

Distribution Statement A
Approved for Public Release Distribution Unlimited

There has also been a large amount of research on this problem in the online setting where constant label feedback is available and can be used to adapt the hypotheses to changes in the target function (Littlestone & Warmuth, 1994; Herbster & Warmuth, 2001). Our analysis is for the more difficult problem where no label feedback is available after training and our approach is to construct a fixed hypothesis that is robust to the adversarial changes.

This paper is organized as follows. In Section 2, we define the problem. Section 3 gives our main performance bounds in terms of the number of corrupt hypotheses. Section 4 specifies the algorithms and relates the number of features corrupted by the adversary to the number of corrupt hypotheses. In Section 5, we give techniques to improve computational performance. Our main experiments are given in Section 6. Section 7 gives some implications of our results for various types of target functions. We wrap up with a section on future work and a section with our conclusions.

2. Adversarial Learning Problem

Let $X = \mathrm{R}^n$ be the instance space and $y = \{1, \ldots, k\}$ be be a set of discrete labels. Let $T = \langle t_i | i \in 1, \ldots, m \rangle$ be a sequence of training instances selected independently from distribution $\mathrm{P}(X, y)$ and let $E = \langle e_i | i \in N \rangle$ be an infinite sequence of test instances selected independently from the same distribution. The adversary is allowed to arbitrarily corrupt l different features on every test instance. More formally, let $C = \langle c_i | i \in N \rangle$ be this corrupted sequence of instances where for all i, $||(e_i - c_i)||_0 \leq l$.

To help describe our algorithms we use the term *corrupt feature* to refer to any feature that has been modified by the adversary and *corrupt hypothesis* to refer to any hypothesis that contains a corrupt feature. For voting, we refer to any hypothesis used in the vote as a *basic* hypothesis. As mentioned, the testing has a potentially infinite number of instances and we refer to each one as a *trial*. While the term trial is often used in online learning (Littlestone, 1988), we should stress our results are not for the online model as we do not receive label feedback after making a prediction.

3. Majority Vote Data Dependent Bounds

The key component of our technique is majority vote where each basic hypotheses predicts a single label and the voting prediction is the label that occurs most frequently.[1] The main intuition is that we generate basic hypotheses for the voting such that a majority of them do not contain features that are corrupt. If these uncorrupt hypotheses predict the correct label than the predicted label will be correct. Of course, it is unlikely that these uncorrupt hypotheses will be perfect, so in this section we give a data dependent bound on the error rate using uncorrupt test data. In the next section, we will show how to control the number of corrupt hypotheses as a function of l.

We make the worst case assumption that corrupt hypotheses always predict the wrong label.[2] Therefore, we can generate an adversarial upper-bound on the error by assuming c hypotheses are corrupt. Using our uncorrupt test data, for each instance x let $\mathrm{score}(x)$ be the vector of label prediction counts over the h basic voting hypotheses. Define $\mathrm{Score}(x, y) = \mathrm{score}(x)[y] - \max_{y' \neq y}(\mathrm{score}(x)[y'])$. If $c = 0$, the majority vote is correct on instance x when $\mathrm{Score}(x, y) > 0$. To get an upper bound on the error with corruption, we modify this test slightly. An instance x must be correct if $\mathrm{Score}(x, y) - 2c > 0$. This makes the worst case assumption that not only do the corrupt hypotheses get the label wrong, but that all corrupt hypotheses switch their predictions to the label that has the next highest vote count. One can apply standard statistical results to generate a confidence interval for this bound.

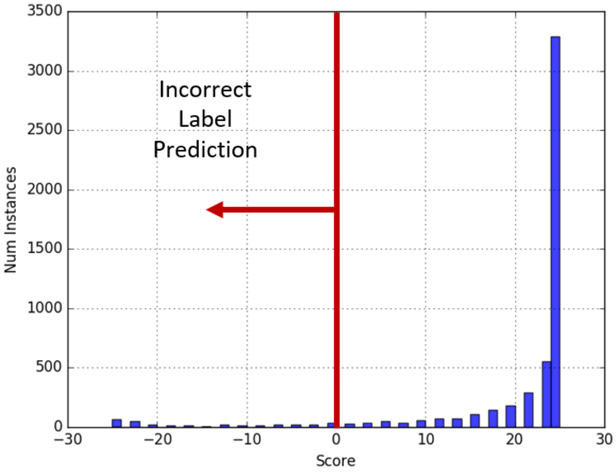

Figure 1. Histogram of Score function values on 5248 test data points with 25 hypothesis for UCI character font dataset. The concept is to determine if character is italic with Arial font. For this experiment, a corrupt feature corrupts at most one basic voting hypothesis.

It is easiest to understand this bound with a plot. In Figure 1, we give a histogram of the Score function over all the test data. Each corrupt hypothesis will shift this histogram by at most 2 to the left. By counting all the values that are still above 0, we get a bound on the error rate of majority vote. In this case, the test error rate is around 0.06, but the numbers in this plot can be used to show that in the worst case the error is at most 0.1 when 5 features are corrupt.

[1] We suggest predicting randomly to break ties.

[2] An sufficiently strong adversary can make this assumption true for some subset of algorithms such as typical hyperplane classifiers.

Notice that the histogram does not show the somewhat idealized case of independence of errors, but for our purpose, dependence is fine. Intuitively, what we are exploiting is redundancy of features. While the different hypotheses learned with these redundant features might be highly correlated, we assume the corruption of one feature has no effect on any related redundant features. Also, while it might seem restrictive to have a hard limit on l, and therefore c, it is straightforward to assume l comes from a distribution and use this distribution to create an upper-bound on the error.

4. Majority Vote Hypotheses Generation

In this section, we give two techniques to generate hypotheses for the majority vote. They are all related to the random subspace method since they only select a subset of features for each hypotheses. Our goal is to bound how many of these h hypotheses can be corrupted by an adversary that is allowed to corrupt at most l features on each instance. To help explain our results, let n be the total number of features and let g be the percentage of hypotheses that are not corrupt.

4.1. Fixed Feature Split

We call our first algorithm fixed-split as it partitions the features into approximately equal sized disjoint groups. We suggest, at a minimum, to randomize the selection of features, since feature relevance might be correlated in the instance space. Another option is to use some type of feature selection procedure, such as mutual information or domain knowledge in an attempt to equalize the quality of the features across the hypotheses. Using a uniform random approach can be considered a random subspace method where features are chosen without replacement.

Notice that any single corrupt feature will corrupt exactly one hypotheses, therefore $g \geq 1 - \frac{l}{h}$. For example, with 900 features, we could learn 9 hypotheses that each have 100 features. If these hypotheses have 0 error then majority vote must also have 0 error even with 4 corrupt features. Of course, the hypotheses will often have higher error rates. The techniques of Section 3 can be used with test data to determine how many corrupt features can be tolerated. In addition, if one has sufficient test data, then one can explore different ways to partition the features. For example, it might make sense to create more hypotheses to allow the algorithm to tolerate more corruption.

4.2. All Size k Feature Subsets

In this section, we consider the technique of generating every possible k feature hypothesis. We call this the k-subset algorithm. While this is generally intractable, the random subspace method is an approximation of this technique, and in Section 5, we will relate the bounds of these two algorithms.

Since we are considering every way to select k features, there are a total of $h = C(n, k)$ hypotheses. Since a corrupt hypothesis has one or more corrupt features, there are total of $C(n - l, k)$ uncorrupt hypotheses, and therefore, $g = \frac{C(n-l,k)}{C(n,k)} = \prod_{i=0}^{i=l-1} \frac{n-k-i}{n-i}$. To make this result easier to understand, we can use the fact that $\ln(x) - \ln(x - a) > \frac{2a}{2x-a}$ to show that $l < \ln\left(\frac{1}{g}\right)\left(\frac{n}{k} - \frac{1}{2}\right)$. What we really want is a lower bound on l, but we found it difficult to achieve a tight and simple lower bound. However, given that this upper bound is tight for $n \gg k$, we use $l \approx \ln\left(\frac{1}{g}\right)\left(\frac{n}{k} - \frac{1}{2}\right)$ as an effective approximation to better understand the result. If needed, one can always use the exact formula.

With the fixed-split algorithm from Section 4.1, the number of hypotheses is approximately n/k therefore, if each hypothesis has at most one corrupted feature, $l \approx (1 - g)\frac{n}{k}$. As can be seen, the ratio $\frac{n}{k}$ is important for both algorithms. Roughly speaking, if we want to double the number of corrupt features the algorithm can tolerate, we need the halve the number of features in each hypothesis. Of course, fewer features in a hypothesis will typically lower its non-corrupted accuracy. We will give examples of this trade-off in Section 6.

Also notice the similarity of the two inequalities. They are almost identical except for a $\ln(\frac{1}{g})$ factor for the k-subset algorithm and a $(1 - g)$ factor for the fixed-split algorithm. Let $r(g) = \frac{\ln(\frac{1}{g})}{1-g}$. Given that $r(0.5) = 1.3863$ and $r(1) = 1$, the k-subset algorithm has a distinct advantage against a worst case adversary as the percentage of corrupt hypotheses increases.[3]

5. Decreasing Majority Vote Cost

A straightforward technique to reduce the cost of majority vote is to only use a uniform random subset of the hypotheses. Assuming that fraction g of the hypotheses are uncorrupted, we can use Bennett's inequality to bound the number of corrupted hypotheses in the reduced set. The main flaw in this analysis is that it only holds for a fixed set of corrupted features. However, unless the adversary knows what hypotheses the algorithm selected, it will be difficult to exploit this weakness. In the experiment section, we get good performance setting $h = 500$. To ground this using the exact binomial distribution, if $g = 0.6$, we would only have a 0.00001 chance of more than half the randomly selected hypotheses being corrupt. This technique is essential for the

[3] Lower g values are not as relevant as too many hypotheses would be corrupt for majority vote to be effective.

fixed sized subset algorithm in Section 4.2 since it is based on a combinatorially large number of hypotheses. When using random sampling with replacement that algorithm is equivalent to the random subspace method.[4]

Even with a reduced set of hypotheses h, prediction time might still be too expensive. We recommend speeding up the prediction time using sequential sampling techniques that randomly sample the voting hypotheses until a prediction can be made with a controllable probability of correctness (Wald, 1947). As we saw in Figure 1, many times most of the hypotheses make the same prediction. Sequential sampling will quickly conclude that only a small amount of sampling is necessary. While most sequential sampling techniques only consider sampling with replacement, it should be possible to prove better bounds when sampling without replacement. Another refinement is to take advantage of a multi-core computer architecture and evaluate several hypotheses each step. As an added benefit, adding this random strategy to prediction can be beneficial against an adversary since it will obscure the behavior of the algorithm.

6. Experiments

First we need to confirm that, given our constraints of feature selection, random subspace methods work with no corrupt features. Second, we need to show our data dependent upper bounds give nontrivial results against worst case adversarial corruption. Last, we want to give some idea of how these algorithms perform with weaker adversaries.

In some ways, the last issue is the most difficult as we want to design a simple but effective adversary for our experiments. Our first attempt was to select features to corrupt by creating a distribution based on mutual information (MI). For each instance, after selecting l features to corrupt from the created MI distribution, we selected new values by uniformly sampling over the range of values seen in the training data. Surprisingly both random forests and support vectors did well with large amounts of corruption.

We were able to degrade the algorithms more by corrupting the features values in a nonuniform way. The results in this paper pick features based on the previously described MI sampling, but for each feature we compute the mean value over the training data and always change the feature value to be the maximum value on the opposite side of the mean. For example, if a feature has a range of [-2,3] and a mean of 1 in the training data then during testing we corrupt a feature with value -1 by changing it to 3. It is an interesting research question to understand and model the different types of feature drift that real problems experience.

The eight datasets we used are described in Table 1. All our voting techniques use random forest as the basic classifier. We also tried support vector machines and neural networks, and while our techniques worked on all algorithms, we found random forest was the best choice for presenting results. Our techniques actually worked better on the other algorithms, but random forest was the fastest to train and test and gave good non-corrupted performance on all the problems we tested. We were somewhat worried that random forest would already have strong resistance to corruption given that it already does majority vote and involves selecting a subset of features, but it is important to remember that random forest randomly selects features at every node in the tree. Therefore random forest is not a random subspace method, and, for large trees, it is likely that the tree uses most of the features.

We used the scikit-learn Python library (Pedregosa et al., 2011) for all our code and used the standard cross validation library for parameter selection where the number of random features select was from $\{\sqrt{n}, 2\sqrt{n}, n\}$ and the number of trees was selected from $\{10, 20, 50, 100\}$. For all experiments, we used a 80/20 train/test split. The only exception is that we set 100,000 as the maximum number of instances for training or testing. For all datasets, we combined and permuted the existing data. This was to ensure that all the data was iid before the adversary corrupted the instances. For the data dependent bounds, the results for the random subspace method are the expected bounds give that the 500 hypotheses are sampled from all possible subsets. On all experiments we also report the error rate of predicting with the majority label. This is a good minimum baseline classifier and is unaffected by feature corruption.

Given the large number of experiments and independent variables, we limit the presented results to a single number of hypotheses for both the fixed split technique and the random subspace technique. For the fixed split algorithm we picked the best result from $h = \{3, 5, 7, 9, 11, 13, 15, 21, 31\}$. For feature corruption, we ran experiments from 0 to 35 corrupt features. We sometimes stopped the experiments early if error-rate degraded to majority label baseline. For the random subspace, given the expense of running the algorithm, we only used a single set of parameters. We used $h = 500$ since, as explained in Section 5, that gives good bounds on finding a representative sample of hypotheses, and we set k based on the $\frac{n}{k}$ value that gave the best results for the fixed partition algorithm.[5] Other values of h and k give qualitatively similar results when the features are significantly corrupted.

[4] The bounds can be improved slightly by switching to sampling without replacement (Bardente & Maillard, 2015).

[5] When focused on a single problem with sufficient validation data, we recommend testing more parameters.

Table 1. Datasets used in corruption experiments.

Data set	Features	Label	|Labels|	|Train|	|Test|	Access
UNO	1024	Device Mode	11	24413	6104	non-public
Pi	1024	Device Mode	3	6067	1517	non-public
Smart	512	Device Mode	4	10000	2000	non-public
Character Font	409	Italic Arial	2	20989	5248	UCI repository
IoT Botnet	115	ACK Attack	2	100000	42591	UCI repository
UJIIndoorLoc	520	Building Floors	5	16838	4210	UCI repository
US Census Data (1990)	68	Marital Status	5	100000	100000	UCI repository
CoverType	54	Forest Covers	7	100000	100000	UCI repository

6.1. Electromagnetic Side Channel Data

We are currently working on a project that determines the computational mode of a device based on its unintended electromagnetic (EM) emissions (Alexander et al., 2018). The goal of this project is to determine if unauthorized code is running on the device. While we do not have space to give all the details on this problem, we have captured EM data of three devices while they execute authorized code. The data is captured using an antenna and a software defined radio sampling at 25 MHz at a specified central frequency. We then processed that data using the fast Fourier transform into either 512 or 1024 frequency bins and use those energy levels as our features. Through various techniques, we have labeled this data into device modes.

One convenient property of this data is the presence of harmonics where information is correlated over the frequency spectrum. This along with strong differences between the device modes, makes the learning problem fairly easy for standard machine learning techniques. The difficulties arise when the system is used to make label prediction at a different time and/or location. Changing, intermittent EM noise can be present and can corrupt different features over a sequence of predictions. This was our motivation to develop these techniques. While this EM noise problem is not a worst case adversary, it has properties that make it difficult to analyze with traditional train/test assumptions.

We ran experiments for three devices: an Arduino UNO, a Raspberry Pi, and a power grid smart meter. It was difficult to do controlled experiments with real environmental noise, so we used the adversarial noise model explained at the start of this section.

Figure 2 gives the result for an Arduino UNO running a simple program consisting of loops of NOP statements. A simple loop that repeats a specific number of clock cycles causes a repeated behavior that is picked up at a specific frequency as an amplitude modulation of the CPU clock. As can be seen, the random forest algorithm eventually gets close to half the predictions wrong. The fixed-split algorithm does much better, but the best results are for the random subspace method. We also give graphs of the worst case adversarial bounds. It is not surprising that these bounds are much worse since they assume the adversary can maximize the errors by controlling the prediction of any corrupt hypothesis. Still the bounds are somewhat positive given that the $\frac{n}{k}$ ratio is roughly 13. The voting hypotheses must have very high accuracy to be able to tolerate corruption of almost half the hypotheses.

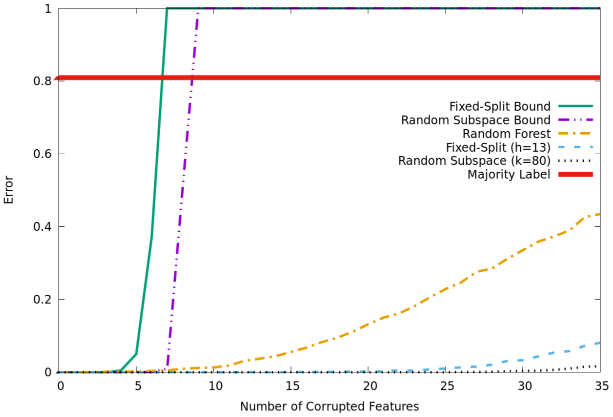

Figure 2. UNO experiments.

Figure 3 tells a similar story. It is based on a Raspberry Pi running Linux with a simple program that loops over SHA, string search, and sleep. While hard to see, the random forest algorithm is doing slightly better at the start; however it quickly decays as $l > 10$. Again the best performance, as the corruption increases, is the random subspace algorithm.

The Smart Meter experiments are based on unmodified code running on the device. We are unsure why the result are so much worse for random forest; perhaps it is related to the large label skew in this problem. However, both fixed-split and random subspace are largely resistant to the corruption with the random subspace algorithm having the lowest error.

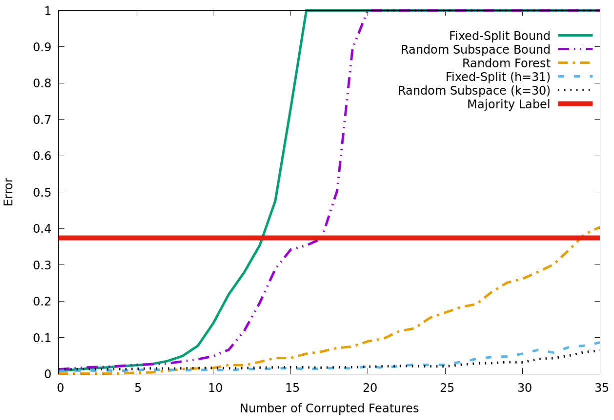

Figure 3. Raspberry Pi experiments.

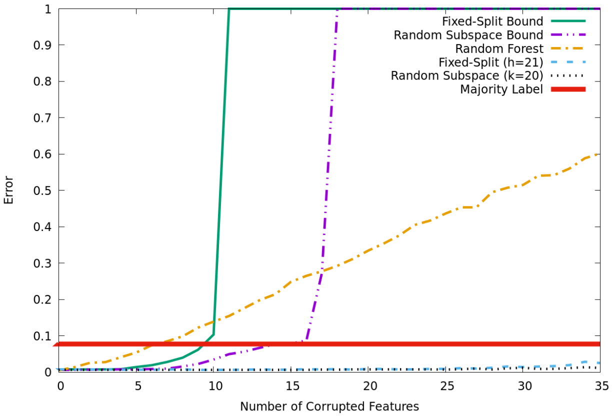

Figure 4. Smart meter experiments. h=21 k=20

6.2. UCI Data

We selected five UCI datasets (Dheeru & Karra Taniskidou, 2018), described in Table 1, from UCI by sorting based on number of instances and choosing problems that fit certain criteria.[6] In particular, we selected classification problems but avoided any problems with less than fifty features or that required extensive feature processing. We also avoid problems that contained features that were clearly a function of some set of original features. The motivation for our problem is that the basic features are independently susceptible to noise, and having features that are functionally related to each other would break that assumption and spread the corruption.[7] We did not select problems based on their performance with the random subspace method as part of our goal was to document a somewhat random sample of problems.

Our first UCI problem tries to determine if a bit-mapped Arial font character is italic. The results here are similar to the previous section with the exception that all the classifiers do not start at zero error. As can be seen in Figure 5, the error increase for both subspace algorithms is very slow. At $l = 35$ both algorithms are doing much better than random forest with a slight advantage to the random subspace method.

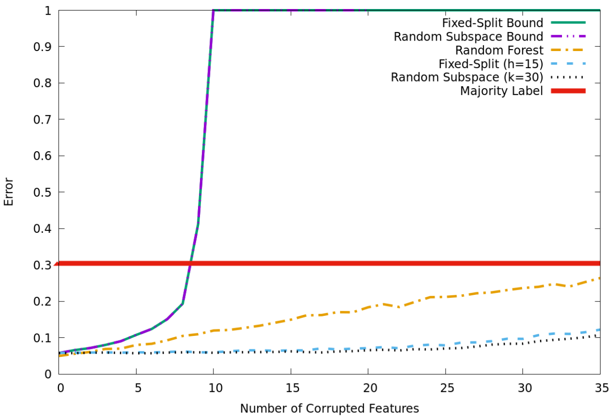

Figure 5. Font experiments.

In Figure 6, we give the results for the binary label problem of determining whether a botnet attack is occurring (Meidan et al., 2018). Again, the results are positive with the subspace algorithms tolerating roughly twice as many corrupt features before it starts to make a significant number of errors. After $l = 20$, both subspace algorithms start to rapidly decay. While it is possible that increasing h and reducing k could decrease the error rate with these large amounts of corruption, the algorithms are already at the point of having only six features per hypotheses which seems extreme. It is likely this problem has a large amount of relevant feature redundancy.

The results in Figure 7 are interesting and are based on predicting the floor the user is on in a building based on data collected from Wi-Fi access points (Torres-Sospedra et al., 2014). The adversary only has a minimal effect on all the algorithms. We are unsure if this is caused by the feature sparsity of this problem combined with our choice of adversary. We plan to study the issue of instance sparsity in the future. However, even in this case, we do see a decrease in error rate for the random subspace algorithm.

Our next data set is based on Census data. Here we defined the label based on the marital status since it has five labels values and reasonable class balance. As can be seen in Figure 8, we cannot tolerate as many corrupt features, but

[6] Our initial intent was to choose problems with lots of instances to validate different ways to split features, but the computational costs have limited us to simpler experiments.

[7] We suggest that any derived, functionally related features should always be placed in the same voting hypothesis to avoid spreading the corruption.

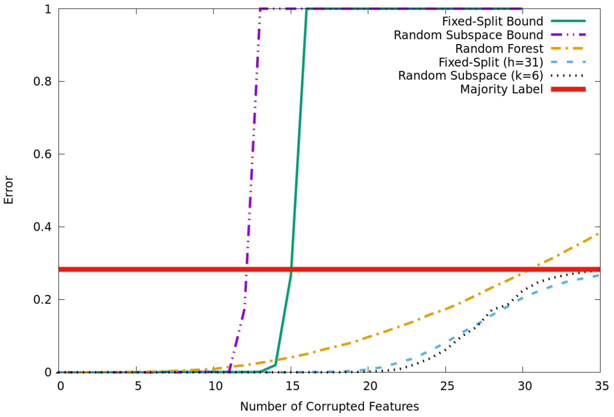

Figure 6. Botnet experiments.

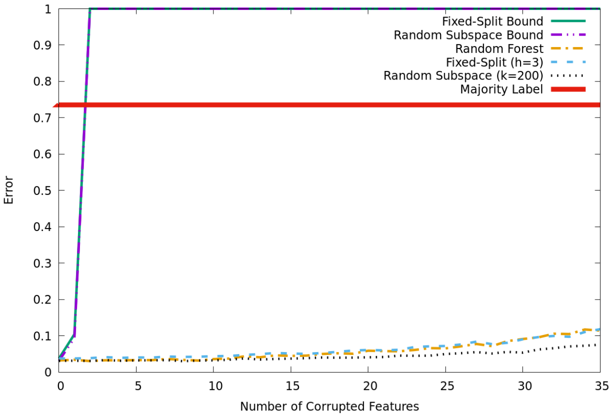

Figure 7. UJI Indoor Location experiments.

we also have significantly fewer total features. Also we suffer some loss of accuracy in the non-corrupt case, but as soon as the corruption starts, the subspace techniques have lower error rates than random forest. In addition, we need to use $h = 3$ as the non-corrupt error rate rises quickly as h increases. This shows the difficulty of the non-corrupt form of this problem for the subspace techniques. Though not shown in the plot, if we new the corruption was $l = 5$ then $h = 7$ would lower the error rate slightly to 0.278. The optimal parameter choice will depend on the amount of corruption. This is a subject for future research as explained in Section 8.

In Figure 9, the learning problem labels different types of forest cover. The results are similar to the previous Census data; however, in this case the problem is even more difficult to learn. Still, we do see improvement for $l = 1$ and $l = 2$. For higher values of l all the algorithms are doing worse than majority label prediction algorithm.

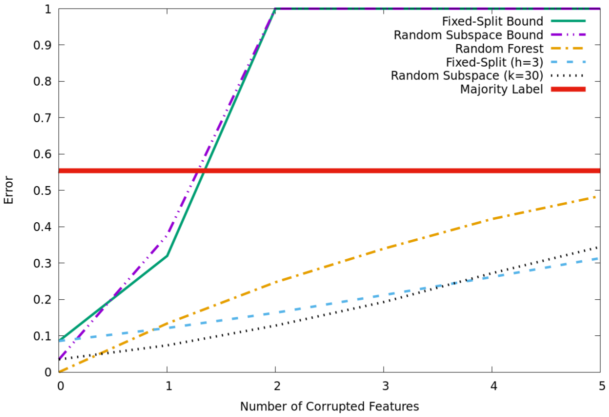

Figure 8. Census experiments.

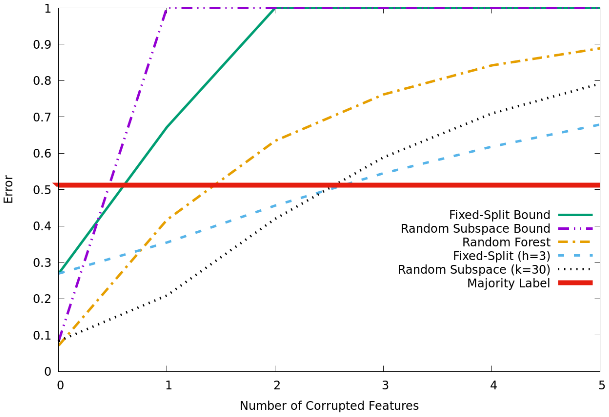

Figure 9. Cover experiments.

7. Difficult Target Functions

An alternative way to interpret our results is a way to quantify what types of learning problems cannot be solved by random subspace methods. For example, a conjunction with no redundant attributes will need at least half the majority vote hypotheses to have every relevant variable. Our voting algorithms are designed to make sure any l features are missing in more than half of the hypotheses. For a conjunction with three terms, the chance that all three will appear in more than half the hypotheses will be small even when k is fairly large. For example, with $n = 1000$ one would need to select $k > 793$ features for the random subspace method to have a 0.5 chance of working even without corruption. It is interesting how well many of our UCI experiments in Section 6.2 perform with much smaller k values. While not definitive, this suggests there are large amounts of relevant feature redundancy in many practical problems.

This is related to the fact that a sufficiently strong adversary can make learning impossible by moving each instance to a part of the space with a different label. It also suggest algorithms that attempt to minimize the number of features are susceptible to these types of adversaries since they might learn functions that remove redundancy.

8. Future Work

We believe the topics of this paper have a rich range of possible extensions. We are currently looking at ways to keep the strong adversarial guarantees of the fixed-split algorithm while increasing the flexibility in creating hypotheses. One approach we have developed proves the same bound as the fixed-split algorithm but with the ability to generate a large number of hypotheses that each have k features. This combines the flexibility of using any k value, and the improved variance reduction of more bagging from the k-subset algorithm with the strong adversarial guarantees of the fixed-split algorithm. The two restrictions are that the number of features n must be prime and the number of hypotheses must be bn for some integer b.[8] These are not significant issues since we anticipate this technique will be most useful for smaller values of n, and we can always add dummy features to increase the n value to the next prime number. Given the prime number theorem, on average we should need to add approximately $\frac{\ln(n)}{2}$ features (Granville, 1995).[9] Our proof for this techniques is based on a combinatorial method of constructing all different necklaces (i.e., decision rules) consisting of the same number of beads (i.e., features) of 2 colors (i.e., presence or absence of a feature in a decision rule). In the future, we plan to do experiments with this technique to see if it improves performance over the fixed-split algorithm.

We are currently working on formalizing a more general adversary. Assume f is the classification function learned by the algorithm and let d be the distance function for a metric space on the set of instances. Define $M(f, x, y, l) = 1$ if there exists a $\hat{x}$ such that $d(x, \hat{x}) \leq l$ and $f(\hat{x}) \neq y$; otherwise $M(f, x, y, l) = 0$. This function captures the property of whether an adversary can change the label by corrupting an instance. With suitable measure theoretic assumptions, given a distribution $P(x, y)$, the worst case error with corruption is $E(f, l) = \int M(f, x, y, l) dP(x, y)$. Letting $z_i = M(f, x_i, y_i, l)$ creates a sequence of $\{0, 1\}$ iid variables, and one can use standard concentration results to create bounds for empirical risk minimization when selecting a function f from a finite set F (Vapnik, 1999). One can interpret these definitions as related to the hyperplane margin. If $f(x) = y$ then the margin is $\delta(x) = \min_{\{\hat{x} \in X | f(\hat{x}) \neq y\}} d(x, \hat{x})$; otherwise $\delta(x) = -\min_{\{\hat{x} \in X | f(\hat{x}) = y\}} d(x, \hat{x})$. Many of the intuitions of margins carry over to these definitions. In particular, there is a natural trade-off when optimizing the margin. You can select a bigger margin which is good against the adversary, but this will force the algorithm to predict more non-corrupt instances incorrectly.

While these random subspace techniques seem ideal for problems with large number of features, we still need to consider the impact of sparse instances. If only a small fraction of the features are non-zero then parameters must be chosen to ensure that a majority of the basic voting hypotheses do not have all zero features. At a minimum, this suggests that the number of features is not as important as the distribution of "active" features. We plan to research this issue including running experiments since sparse instance problems are common in many setting such as text.

Another issue is how to set the parameters when one does not know how many features the adversary will corrupt. Using an algorithm that can tolerate high l values generally has the trade-off of degrading performance for small l values. We are currently looking into using a vote to combine several random subspace algorithms to create a technique that is more robust for a range of l values. Ideally it should be able to adapt if l changes over the course of an experiment.

A related topic is to create an algorithm that can learn when the adversary is limited to corrupting a fixed set of features. In this case, the adversary can pick l features at the start of testing and only corrupt those features. A straightforward strategy for the algorithm is to try to identify those features/hypotheses that are corrupt and remove them from the prediction procedure. While this problem is useful in its own right, we also believe it will be a useful tool for solving the previous issue of optimizing for an unknown l value when dealing with the more general adversary.

9. Conclusion

This paper presents a new analysis of the random subspace method and shows that with appropriate parameters it can tolerate arbitrary corruption of a limited number of features. While the amount of corruption that can be tolerated depends on unknown details of the problem, we give a statistic that can be used to estimate the worst case performance of the algorithm using uncorrupted test data. This is similar to the traditional test bound used in iid supervised learning but allows us to extend that framework to handle adversarial changes in the instances. While adversaries are not typically encountered in learning problems, the proofs also apply to other situations that include various types of instance distribution drift. We give experiments to show these algorithms perform well on a range of realistic problems including five UCI datasets and three new datasets based on electromagnetic side channel information.

[8] bn has a maximum value of $C(n, k)$.

[9] When learning we can just remove the dummy features.

Acknowledgments

The authors are grateful for the support of DARPA I2O and AFRL via contract number FA8650-16-C-7623 and the LADS program. The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

Alexander, S., Agrawal, H., Chen, R., Hollingsworth, J., Hung, C., Izmailov, R., Koshy, J., Liberti, J., Mesterharm, C., Morman, J., Panagos, T., Pucci, M., Sebuktekin, I., and Tsang, S. Casper: an efficient approach to detect anomalous code execution from unintended electronic device emissions. In *Cyber Sensing 2018*, 2018.

Bardente, R. and Maillard, O.-A. Concentration inequalities for sampling without replacement. *Bernoulli*, 21(3):1361–1385, 09 2015. doi: 10.3150/14-BEJ605.

Bifet, A., Frank, E., Holmes, G., and Pfahringer, B. Ensembles of restricted hoeffding trees. *ACM Transactions on Intelligent Systems and Technology*, 3, 2012.

Biggio, B., Fumera, G., and Roli, F. Multiple classifier systems for robust classifier design in adversarial environments. *International Journal of Machine Learning and Cybernetics*, 1(1-4):27–41, 2010.

Dheeru, D. and Karra Taniskidou, E. UCI machine learning repository, 2018. URL http://archive.ics.uci.edu/ml.

Granville, A. Harald cramer and the distribution of prime numbers. *Scandinavian Actuarial Journal*, 1:12–28, 1995.

Herbster, M. and Warmuth, M. K. Tracking the best linear predictor. *Machine Learning*, 1:281–309, 2001.

Ho, T. K. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, Aug 1998. ISSN 0162-8828. doi: 10.1109/34.709601.

Littlestone, N. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

Littlestone, N. and Warmuth, M. K. The weighted majority algorithm. *Information and Computation*, 108:212–261, 1994.

Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Shabtai, A., Breitenbacher, D., , and Elovici, Y. N-baiotnetwork-based detection of iot botnet attacks using deep autoencoders. *IEEE Pervasive Computing*, 17:12–22, 2018.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Quionero-Candela, J., Sugiyama, M., Schwaighofer, A., and Lawrence, N. D. *Dataset Shift in Machine Learning*. The MIT Press, Cambridge, Massachusetts, 2009.

Sugiyama, M. and Kawanabe, M. *Machine Learning in Non-Stationary Environments: Introduction to Covariate Shift Adaptation*. The MIT Press, Cambridge, Massachusetts, 2012.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., and Fergus, R. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.

Torres-Sospedra, J., Montoliu, R., Martnez-Us, A., Avariento, J., J. Arnau, T., Benedito-Bordonau, M., and Huerta, J. Ujiindoorloc: A new multi-building and multi-floor database for wlan fingerprint-based indoor localization problems. In *International Conference on Indoor Positioning and Indoor Navigation*, 10 2014. doi: 10.1109/IPIN.2014.7275492.

Vapnik, V. *The Nature of Statistical Learning Theory*. Springer Science and Business Media, 1999.

Wald, A. *Sequential Analysis*. John Wiley and Sons, Cambridge, Massachusetts, 1947.