Proposals for Model Security:

Fair and private models, whitehat and forensic model debugging, and common sense

June 19, 2019

Like many others I've known for sometime that machine learning models could pose security risks. A recent flourish of posts and papers has outlined the broader topic, listed attack vectors and vulnerabilities, and started to propose defensive solutions. Before we get started, it's important to say I am no security expert, but I have been following the areas of machine learning debugging, explanations, fairness, interpretability, and privacy very closely and it's many of these techniques that I think can be applied to defend predictive modeling systems. The aim of this post is to brainstorm specifically on potential security problems and solutions in the context of popular, traditional predictive modeling systems, like linear and tree-based models trained on static datasets. In hopes of furthering discussion between *actual* security experts and practitioners in the applied machine learning community (like me), this post will put forward some not unreasonable scenarios for attack vectors on a typical machine learning system at a typical organization and propose tentative defensive solutions.

1 Data Poisoning Attacks

Data poisoning refers to someone systematically changing your training data to manipulate your model's outputs. (Data poisoning attacks have also been called "causative" attacks.) To poison data, an attacker must have access to some or all of your training data, and at many companies, many different employees, consultants, and contractors have just that – and with little oversight. It's also possible a malicious external actor could acquire unauthorized access to some or all of your training data and poison it. A very direct kind of data poisoning attack might involve altering the labels of a training dataset so that whatever the commercial application of your model is, the attacker could dependably benefit from that model's predictions. For example, by altering labels so that your model learns to award large loans, large discounts, or small insurance premiums to people like themselves. (Forcing your model to make a false prediction for the attacker's benefit is sometimes called a violation of your model's "integrity".) It's also possible that a malicious actor could use data poisoning to train your model to intentionally discriminate against a group of people, never awarding them the big loan, big discount, or low premiums they rightfully deserve. This is like a denial-of-service (DOS) attack on your model itself. (Forcing your model to make a false prediction to hurt others is sometimes called a violation of your model's "availability".) While it might be easiest to think of data poisoning attack as changing the values in the existing rows of a dataset, data poisoning can also be done by adding seemingly harmless or superfluous columns onto a dataset. Values in these columns could then trigger altered model predictions.

Potential Solution(s):

- **Disparate impact analysis**: Many banks already undertake disparate impact analysis for fair lending purposes to determine if your model is treating different types of people in an unfair manner. Many other organizations do not seem to care. Disparate impact analysis could potentially discover intentional discrimination in model predictions. There are several great open source tools for disparate impact analysis, such as aequitas, themis, and AIF360.
- **Fair or private models**: Models such as learning fair representations (LFR) and private aggregation of teacher ensembles (PATE) try focus less on individual demographic traits to make predictions, and these models may be less susceptible to discriminatory data poisoning attacks.

- **Reject on Negative Impact (RONI)**: RONI is a technique that removes rows of data from the training dataset that decrease prediction accuracy. See further 8 for more information on RONI.
- **Residual analysis**: Look for strange, prominent patterns in the residuals of your model predictions, especially on employees, consultants, or contractors.
- **Self-reflection**: Score your models on your employees, consultants, and contractors and look for anomalously beneficial predictions.

Disparate impact analysis, residual analysis, and self-reflection can be conducted at training time and as part of real-time model monitoring activities.

2 Watermark Attacks

Watermarking is a term borrowed from the deep learning security literature that often refers to putting special pixels into an image to trigger a desired outcome from your model. It seems possible to do the same with customer or transactional data. An employee, consultant, contractor, or malicious external actor with access to the production code that makes real-time predictions could change that code to recognize a strange or unlikely combination of input variable values to trigger a desired prediction outcome. Like data poisoning, watermark attacks can be used to attack your model's integrity or availability. For instance, to attack your model's integrity, a malicious insider could easily insert a payload into your model's production scoring code that recognizes the combination of age of 0 and 99 years at an address to trigger some kind of positive prediction outcome for themselves or their associates. To deny model availability, an attacker could insert an artificial, discriminatory rule into your model's scoring code that prevents your model from producing positive outcomes for a certain group of people.

Potential Solution(s):

- **Disparate impact analysis**: see section 1.
- **Version control**: Production model scoring code should be managed and version-controlled just like any other mission-critical software.
- **Data integrity constraints**: Many databases don't allow for strange or unrealistic combinations of input variables and this could potentially thwart watermarking attacks. Applying data integrity constraints on live, incoming data streams could have the same benefits.
- **Outlier detection**: Autoencoders are a common fraud detection model that can identify input data that is strange or unlike other input data, but in complex ways. Autoencoders could potentially catch any watermarks used to trigger malicious mechanisms.

Disparate impact analysis, data integrity constraints, and outlier detection can be used at training time and as part of real-time model monitoring activities.

3 Inversion by Surrogate Models

Inversion basically refers to getting unauthorized information out of your model – as opposed to putting information into your model. Inversion can also be an example of a "exploratory reverse-engineering" attack. If an attacker can receive many predictions from your model API or other endpoint (website, app etc.), they can train their own surrogate (i.e. a "simulation") model, of your predictive model. An attacker could conceivably train a surrogate or simulation model between the inputs they used to generate the received predictions and the received predictions themselves. Depending on the number of predictions they can receive, the surrogate model could become quite an accurate simulation of your model. Once the surrogate model is trained, then the attacker has a sandbox from which to plan impersonation (i.e. "mimicry") or adversarial example attacks against your model's integrity or the potential ability to start reconstructing aspects of your potentially sensitive training data. These types of surrogate models can also be trained using external data sources that can be somehow matched to your predictions, as ProPublica famously did with the proprietary COMPAS recidivism model.

Solution(s):

- **Authorized access**: Require additional authentication (e.g. 2FA) to receive a prediction.
- **Throttle predictions**: Restrict high numbers of rapid predictions from single users; consider artificially throttling prediction latency.
- **Whitehat surrogate models**: As a whitehat hacking exercise, train your own surrogate models between your inputs and the predictions of your production model and see ...
  - The accuracy bounds of different types of surrogate models; try to understand the extent to which a surrogate model can really be used to learn unfavorable knowledge about your model.
  - The types of data trends that can be learned from your surrogate model, like linear trends represented by linear model coefficients.
  - The types of segments or demographic distributions that can be learned by analyzing the number of individuals assigned to certain surrogate decision tree nodes.
  - The rules that can be learned from a surrogate decision tree, for example how to reliably impersonate an individual that would receive a beneficial prediction.

4 Membership Inference by Surrogate Model

In an attack similar to inversion, also carried out by surrogate models, a malicious actor can determine whether a given person or product is in your model's training data. Called a *membership inference attack*, this hack is executed with two layers of surrogate models. First an attacker passes data into a public prediction API or other endpoint, receives predictions back, and then trains a surrogate model or models between the passed data and the predictions. Once a surrogate model or models has been trained to replicate your model, the attacker then trains a second layer classifier that can differentiate between data that was used to train the surrogate(s) and data that was not used to train the surrogate(s). When this second model is used to attack your model, it can give a solid indication as to whether any given row (*or rows*) of data is in your training data. Membership in a training data set can be sensitive when the model and data are related to undesirable outcomes such as bankruptcy or disease, or desirable outcomes like high income or net worth. Moreover, if the relationship between a single row and the target of your model can be easily generalized by an attacker, such as an obvious relationship between race, gender, age and some undesirable outcome, this attack can violate the privacy of an entire group of people. Frighteningly, when carried out to it's fullest extent, a membership inference attack could also allow a malicious actor, with access only to a public prediction API or other model endpoint, to reconstruct portions of a sensitive or valuable dataset.

Solution(s):

- **Authorized access**: see section 3.
- **Disparate impact analysis**: see section 1.
- **Fair or private models**: see section 1.
- **Monitor for development data**: Monitor for data that is within a certain range of any individual used to develop the model. Real-time scoring of rows that are extremely similar or identical to data used in training, validation, or testing should be recorded and investigated.
- **Throttle predictions**: see section 3.
- **Whitehat surrogate models**: see section 3.

5 Adversarial Example Attacks

A motivated attacker could conceivably learn, say by trial and error (i.e. "exploration"), surrogate model inversion, or by social engineering, how to game your model to receive their desired prediction outcome or to avoid an undesirable prediction. Carrying out an attack by specifically engineering a row of data for such a purpose is referred to as an adversarial example attack, and sometimes also as a "exploratory integrity" attack. An attacker could use an adversarial example attack to grant themselves a large loan or a low insurance premium, or to avoid a long prison sentence based on a high criminal risk score. Some people might call using adversarial examples to avoid an undesirable outcome from your model prediction "evasion".

Solution(s):

- **Activation analysis**: Activation analysis requires benchmarking internal mechanisms of your predictive models, such as the average activation of neurons in your neural network or the proportion of observations assigned to each leaf node in your random forest, then comparing that information against your model's behavior on incoming, real-world data streams. As one of my colleagues put it, "this is like seeing one leaf node in a random forest correspond to 0.1% of the training data but hit for 75% of the production scoring rows in an hour". Patterns like this could be evidence of an adversarial example attack.
- **Authorized access**: see section 3.
- **Benchmark models**: Use a highly-transparent benchmark model when scoring new data in addition to your more complex model. Linear models could be seen as harder to hack because their mechanisms are so transparent. When scoring new data compare your new fancy machine learning model against an older, trusted linear model or a linear model trained on a trusted data source and pipeline. If the difference between your more complex and opaque machine learning model and your linear model is too great, fall back to predictions of your linear model or send the row of data for manual processing. Also record the incident, because it could be an adversarial example attack.
- **Throttle predictions**: see section 3.
- **Whitehat sensitivity analysis**: Use sensitivity analysis to conduct your own exploratory attacks to understand what variable values (or combinations) can cause large swings in predictions and screen for these values or combinations of values when scoring new data.
- **Whitehat surrogate models** see section 3.

Activation analysis and benchmark models can be used at training time and as part of real-time model monitoring activities.

6 Impersonation

A motivated attacker can learn, say again by trial and error, surrogate model inversion, or by social engineering, what type of input or individual receives a desired prediction outcome and then impersonate this input or individual to receive their desired prediction outcome for your model. (Impersonation attacks are sometimes also known as "mimicry" attacks.) Like an adversarial example attack, an impersonation attack involves artificially changing the input data values to your model. Unlike an adversarial example attack, where a potentially random combination of input data values could be used to trick your model, impersonation implies using the information associated with another modeled entity (i.e. convict, customer, financial transaction, patient etc.) to receive the prediction your model associates with that type of individual. For example an attacker could learn what characteristics your model associates with giving large discounts, like comp-ing a room at a casino for a big spender, and then falsify their information to receive the same discount. They could also share their strategy with others, potentially leading to large losses for your company.

If your are using a two stage model, be aware of an attack vector often known as an "allergy" attack, where a malicious actor would impersonate a normal row of input data for the first stage of your model to conduct an attack on the second stage of your model.

Solution(s):

- **Activation analysis**: see section 5
- **Authorized access**: see section 3.
- **Screen for duplicates**: At scoring time track the number of similar records your model is exposed to, potentially in a reduced-dimensional space using autoencoders, multi-dimensional scaling (MDS), or similar dimension reduction techniques. If too many similar rows are encountered during some time-span, take corrective action.
- **Security-aware features**: Keep a feature in your pipeline, say `num_similar_queries`, that may be useless when your model is first trained or deployed, but could be populated at scoring time or during future model retrainings, that could make your model or your pipeline security-aware. For instance if at scoring time `num_similar_queries` is greater than zero, the scoring request could be sent for human oversight. In the future, when you retrain your model, you could teach it to give input data rows with high `num_similar_queries` values negative prediction outcomes.

Activation analysis, screening for duplicates, and security-aware features can be used at training time and as part of real-time model monitoring activities.

7 General Concerns

Several common machine learning usage patterns also have more general security concerns.

Blackboxes and Unnecessary Complexity: Although recent developments in interpretable models and model explanations have provided the opportunity to use accurate and also transparent nonlinear classifiers and regressors, many machine workflows are still centered around more traditional blackbox models. Such blackbox models are only one type of often unnecessary complexity in a typical commercial machine learning workflow. Other examples of potentially harmful complexity could be overly-exotic feature engineering or large numbers of package dependencies. Such complexity can be problematic for at least two reasons.

- A dedicated, motivated attacker can, over time, learn more about your overly complex blackbox modeling system than you or your team knows about your own model. (Especially in today's overheated and turnover-prone data "science" market.) To do so, they can use many newly available model-agnostic explanation techniques and old-school sensitivity analysis, among many other more common hacking tools. This knowledge imbalance can potentially be exploited to conduct the attacks described in sections 1 – 6 or for other yet unknown types of attacks.
- Machine learning in the research and development environment is highly dependent on a diverse ecosystem of open source software packages. Some of these packages have many, many contributors and users. Some are highly specific and only meaningful to a small number of researchers or practitioners. It's well understood that many packages are maintained by brilliant statisticians and machine learning researchers whose primary focus is mathematics or algorithms, not software engineering, and certainly not security. It's not uncommon for a machine learning pipeline to be dependent on dozens or even hundreds of external packages, anyone of which could be hacked to conceal an attack payload.

Distributed Systems and Models: For better or for worse, we are in the age of big data, and many organizations are now using distributed data processing and machine learning systems. Distributed computing can provide a broad attack surface for a malicious internal or external actor in the context of machine learning. Data could be poisoned on only one or a few worker nodes of a large distributed data storage or processing system. A back door for watermarking could be coded into just one model of a large ensemble. Instead of debugging one simple dataset or model, now practitioners must examine data or models distributed across large computing clusters.

Large-scale Distributed Denial of Service (DDOS) attacks: If a predictive modeling service is crucial to your organization's mission, ensure you have atleast considered more conventional distributed denial of service attacks, where attackers hit the public-facing prediction service with an incredibly high volume of requests to delay or stop predictions for legitimate users.

7.1 General Solutions

Several older and newer general best practices can be deployed to lessen your security vulnerabilities and to generally increase fairness, accountability, transparency, and trust in machine learning predictive systems.

Authorized access and prediction throttling: Standard safe-guards such as additional authentication and throttling may be highly effective at stymieing a number of the attack vectors described in sections 1–6.

Benchmark models: An older or trusted linear modeling pipeline, or other highly transparent predictor, can be used as a benchmark model from which to measure whether a prediction was manipulated by any number of means, such as data poisoning, watermark attacks, or adversarial example attacks. If the difference between your trusted model's prediction and your more complex and opaque model's predictions are too large, record these instances, refer them to human analysts, or take other appropriate forensic or remediation steps. Of course serious precautions should be taken to ensure your benchmark model and pipeline remains secure and unchanged from it's original, trusted state.

Interpretable, Fair, or Private Models: The techniques now exist, e.g. monotonic GBMs (M-GBM), scalable Bayesian rule lists (SBRL), eXplainable Neural Networks (XNN), that could allow you to have both accuracy and interpretability. These accurate and interpretable models are easier to document and debug than classic machine learning black boxes. Newer types of fair and private models, e.g. LFR, PATE, can also be trained to essentially care less about outward visible, demographic characteristics that can be easily observed, socially-engineered into an adversarial example attack, or impersonated. When considering creating a new machine learning workflow in the future, think about basing it on lower-risk, interpretable, private or fair models that are more easily debugged and potentially robust to changes in an individual entity's characteristics.

Model Debugging for Security: The newer field of model debugging is focused on discovering errors in machine learning model mechanisms and predictions and remediating those errors. Debugging tools such a surrogate models, residual analysis, and sensitivity analysis can be used in whitehat exercises to understand your own vulnerabilities or for forensic exercises to find any potential attacks that may have occurred or be occurring.

Model Documentation and Explanation Techniques: Model documentation is a risk-mitigation strategy that has been used for decades in banking. It allows knowledge about complex modeling systems to be preserved and transferred as teams of model owners change over time. Model documentation has been traditionally applied to highly-transparent linear models, but with the advent of powerful, accurate explanatory tools such as tree SHAP and derivative-based local feature contributions for neural networks, now even pre-existing black box model workflows can be atleast somewhat explained, debugged, and documented. Documentation should obviously now include all security goals, including any known, remediated, or anticipated security vulnerabilities.

Model Monitoring and Management Explicitly for Security: Most serious practitioners understand most models are trained on static snapshots of reality represented by training data and their prediction accuracy degrades in real-time as present realities drift away from the information captured in the training data. Today most model monitoring is aimed at discovering this drift in input variable distributions that will eventually lead to accuracy decay. Model monitoring should now likely be designed to monitor for the attacks described in sections 1 – 6 and any other potential threats your whitehat model debugging exercises uncover. (While not directly related to security, my opinion is models should also be evaluated for disparate impact in real-time as well.) Along with model documentation all modeling artifacts, source code, and associated metadata need to be managed, versioned, and audited for security like the valuable commercial assets they are.

Random data detection: Because random data could be used to generate surrogate models, rules for generating adversaries, or for boundary checks, autoencoder models could be used to detect random data in real-time and prevent the issuance of predictions on random data.

Security-aware features: Features, rules, and pre- or post-processing steps can be included in your models or pipelines that are security-aware, such as the number of similar rows seen by the model, whether the current row represents an employee, contractor, or consultant, or whether the values in the current row are similar to those found in whitehat adversarial example attacks. These features may or may not be useful when a model is first trained, but keeping a placeholder for them when scoring new data or when retraining future iterations of your model may come in very handy one day.

Systemic anomaly detection: Train an autoencoder–based anomaly detection meta-model on your entire predictive modeling system's statistics: number of predictions in some time period, latency, CPU, memory, and disk loads, and everything else you can get your hands on, and then closely monitor this meta-model for anomalies. An anomaly could tip you off that something was generally not right in your predictive modeling system. Subsequent investigation or specific mechanisms would be needed to trace down the exact problem.

8 References and Further Reading

A lot of the contemporary academic literature focuses on adaptive learning, deep learning, and encryption. However, I don't know many practitioners who are actually doing these things yet. So in addition to recently published articles and blogs, I found papers from the 1990s and early 2000s about network intrusion, virus detection, spam filtering, and related topics to be really helpful resources as well. If you'd like to learn more about the fascinating subject of securing machine learning models, here are the main references – past and present – that I used for this post. I'd recommend them for further reading too.

Barreno, Marco, et al. "The Security of Machine Learning." *Machine Learning* 81.2 (2010): 121-148. URL: `https://people.eecs.berkeley.edu/~adj/publications/paper-files/SecML-MLJ2010.pdf`

Kumar, Ajitesh. "Security Attacks: Analysis of Machine Learning Models." *DZone* (2018). URL: `https://dzone.com/articles/security-attacks-analysis-of-machine-learning-mode`

Lorica, Ben and Loukides, Mike. "You created a machine learning application. Now make sure its secure". O'Reilly Ideas. URL: `https://www.oreilly.com/ideas/you-created-a-machine-learning-application-now-make-sure-its-secure`

Papernot, Nicolas. "A Marauder's Map of Security and Privacy in Machine Learning: An overview of current and future research directions for making machine learning secure and private. " *Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security.* ACM, 2018. URL: `https://arxiv.org/pdf/1811.01134.pdf`

Shokri, Reza, et al. "Membership Inference Attacks Against Machine Learning Models." 2017 IEEE Symposium on Security and Privacy (SP). IEEE, 2017. URL: `https://www.cs.cornell.edu/~shmat/shmat_oak17.pdf`

Conclusion

I care deeply about the science and practice of machine learning, and I'm now concerned a terrible machine learning hack, combined with credible concerns about privacy violations and algorithmic discrimination, could increase burgeoning public skepticism about machine learning and AI. We should all remember there have been AI winters in the not-so-distant past. Security vulnerabilities, privacy violations, and algorithmic discrimination could all potentially combine to lead to decreased funding for machine learning research or draconian over-regulation of the field. Let's start discussing and addressing these important problems sooner rather than later.

Acknowledgements

Thanks to Prashant Shuklabaidya and Tom Kraljevic for their insightful comments and suggestions.