MACHINE LEARNING AND SECURITY

BOOK REVIEW

Tin Votan
15. Oktober 2019

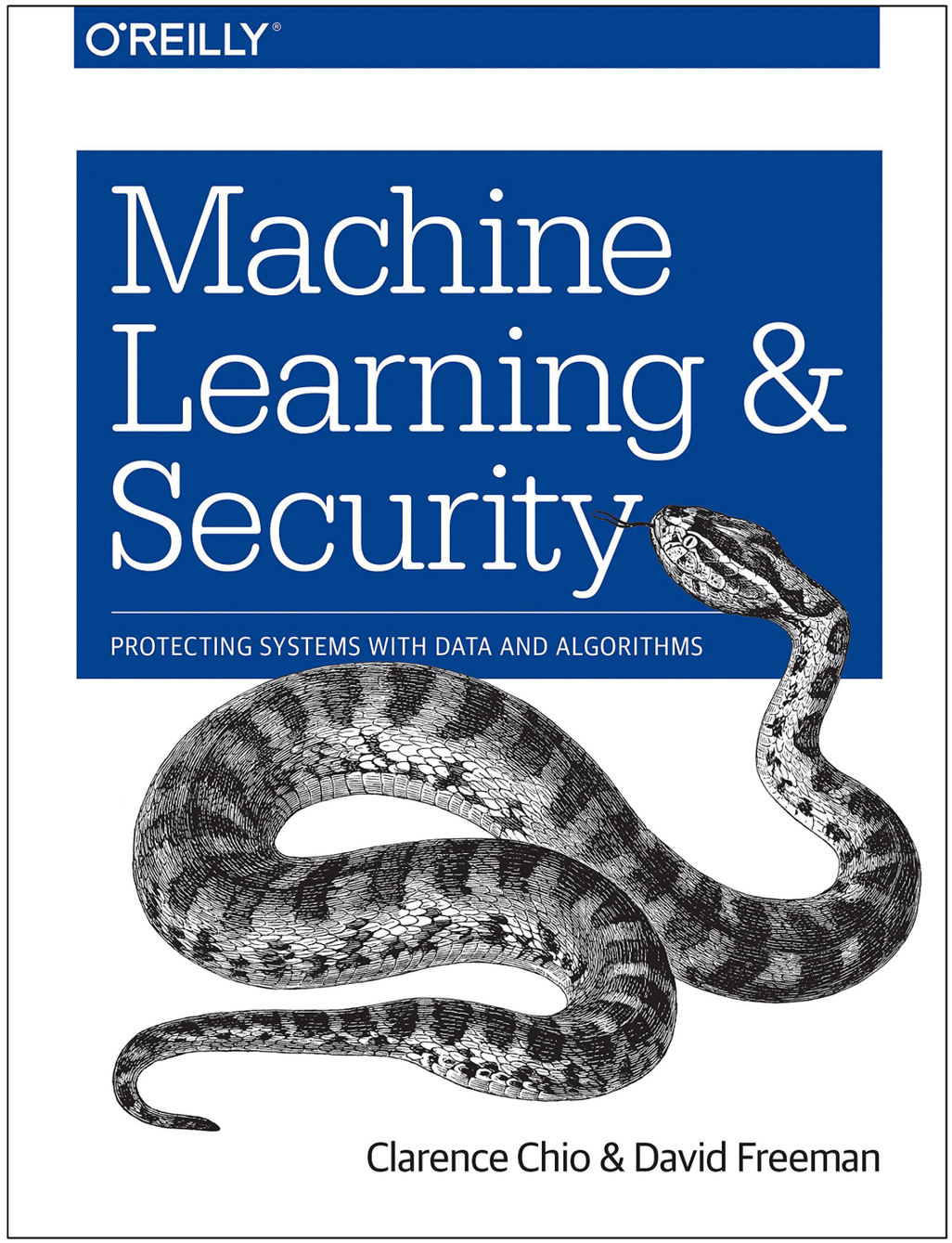
TIN VOTAN

Wirtschaftsingenieurwesen/Master (KIT)
Machine Learning Engineer (Freelancer)

- Start-up
- Machine Learning & Cyber-Security
- Python

Hobbies:
- Lesen, Sport, Kochen

Random facts:
- Zäpfchenzähne
- 18h-Harry-Potter-Marathon in Duisburg



Machine Learning and Security by Clarence Chio and David Freeman (O'Reilly). Copyright 2018 Clarence Chio and David Freeman, ISBN 978-1-491-97990-7.

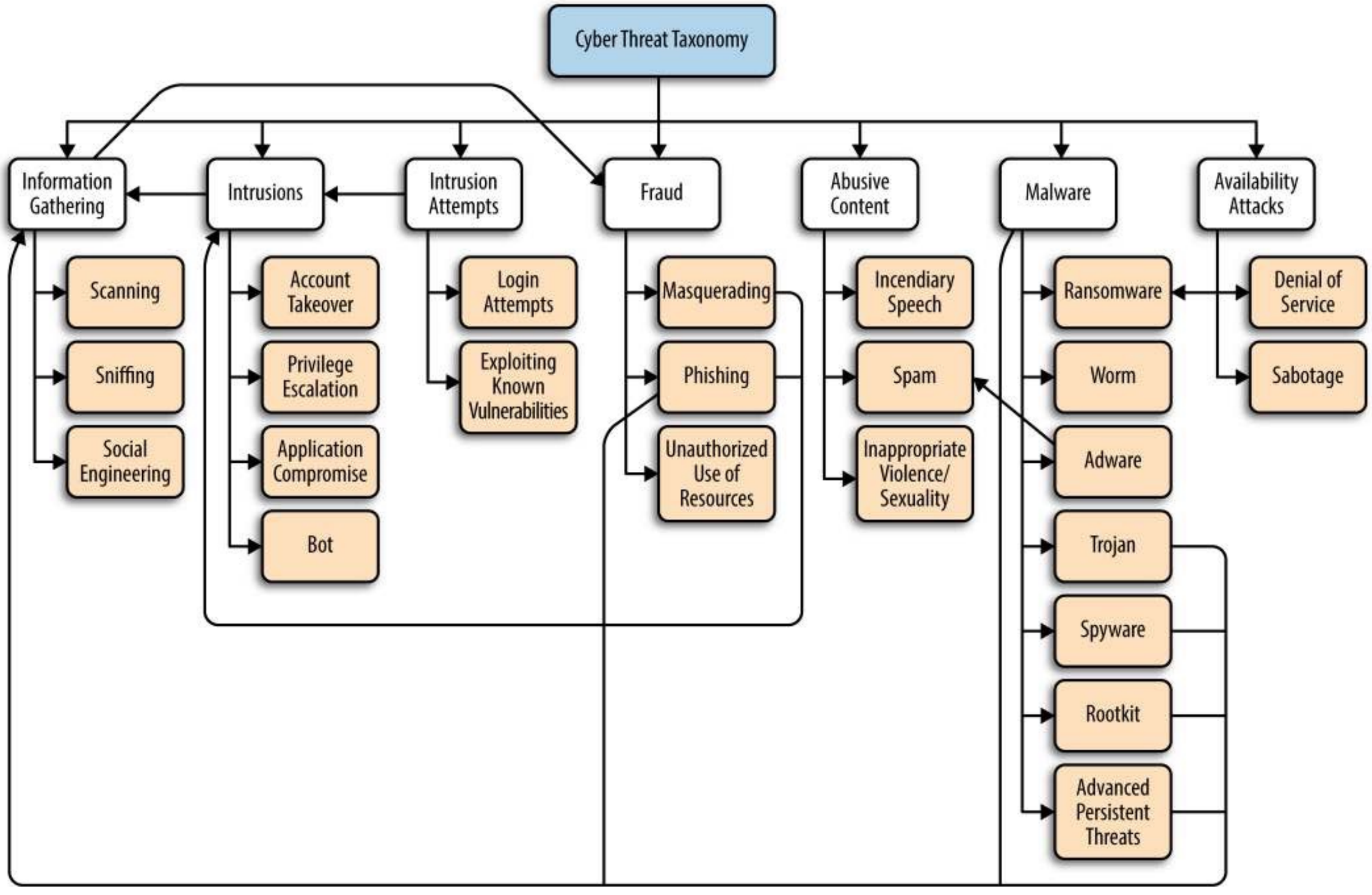
- 353 Seiten (effektiv)
- Englisch (keine deutsche Version)
- Praxisbasiert (mit Python-Code)
- ~ EUR 40,-

Grundlegende Kenntnisse in:

- Python
- Machine Learning
- ML/Data-Science Libraries (Scikit-Learn, Matplotlib, Numpy, Tensorflow, …)
- Jupyter Notebook

GitHub-Repository:
https://github.com/oreilly-mlsec/book-resources

Website:
https://mlsec.net



Cyber Threat Taxonomy tree (p.6)

ANOMALY DETECTION

MACHINE LEARNING USE CASES IN SECURITY

Pattern Recognition

Erkennung von (versteckten) Mustern in (großen) Datensätzen

- Spam Detection
- Malware Detection
- Botnet Detection

Anomaly Detection

Ausreißer in (normalen) Verhaltensmustern

- URL Detection
- Network Outlier Detection

FORMEN VON ANOMALY DETECTION

Novelty Detection

“involves learning a representation of ‘regular’ data using data that does not contain any outliers”

Outlier Detection

“involves learning from data that contains both regular data and outliers”

Anomaly Detection is closely coupled with the concept of time series analysis because an anomaly is often defined as a deviation from what is normal or expected, given what had been observed in the past.

KATEGORIEN VON ANOMALY DETECTION

Forecasting (Supervised machine learning)

Statistical metrics

Unsupervised machine learning

Goodness-of-fit tests

Density-based methods

ONE-CLASS SVM

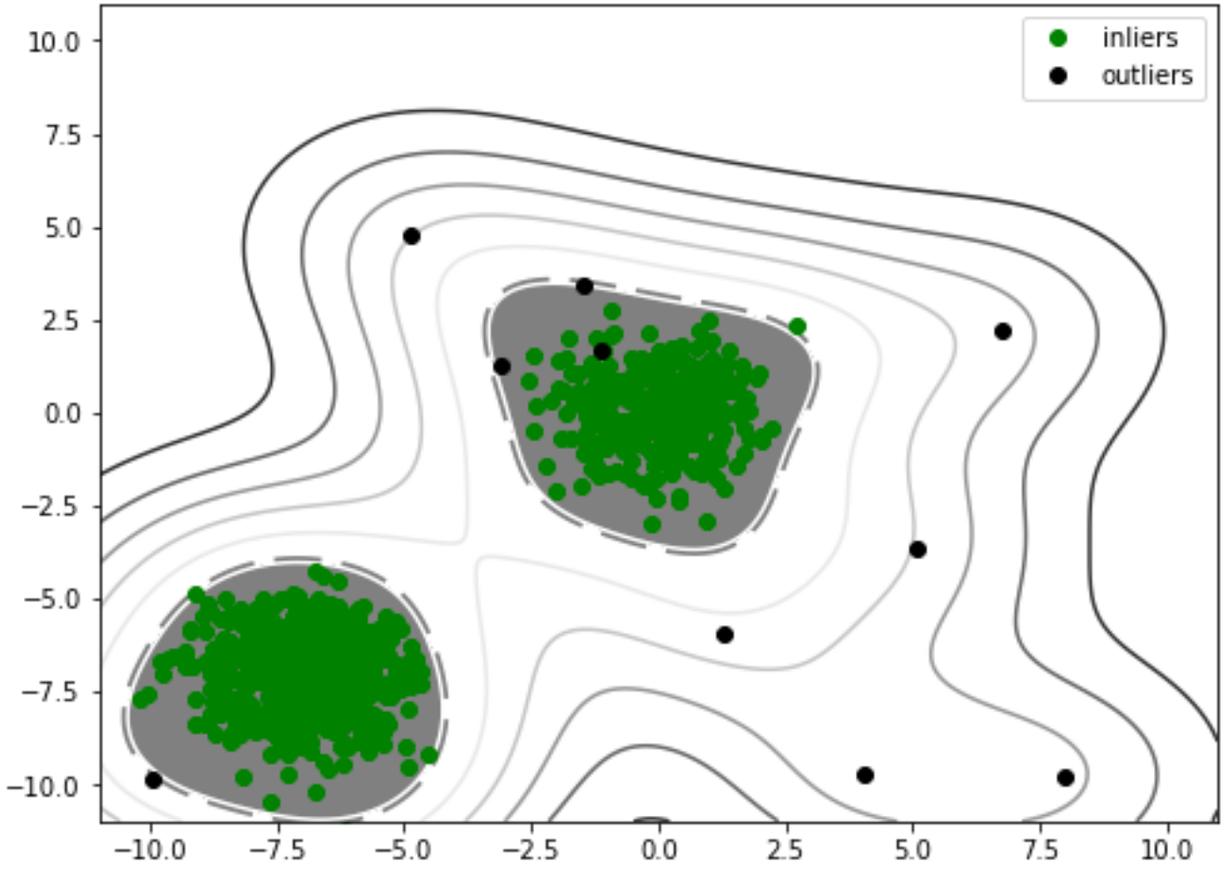
```
1  from sklearn import svm
2
3  classifier = svm.OneClassSVM(nu=0.99 * outlier_ratio + 0.01, kernel="rbf", gamma=0.1)
```

Parameter	Funktion
nu	Parameter steuert die obere Grenze auf den Anteil der Trainingsfehler und die untere Grenze auf den Anteil der Stützvektoren
kernel	“rbf” = Radial Basis Function
gamma	Optimierung des RBF-Kernel

Decision boundary for one-class SVM on bimodal synthetic data—trained using both outliers and inliers

```
x_0 = np.random.randn(num_inliers//3, num_dimensions)
x_1 = np.random.randn(num_inliers//3, num_dimensions) - 7
x_2 = np.random.randn(num_inliers//3, num_dimensions) - 7
x_rand = np.random.uniform(low=-10, high=10, size=(num_outliers, num_dimensions))

# Add outliers sampled from a random uniform distribution
x = np.r_[x_0, x_1, x_2, x_rand]

```

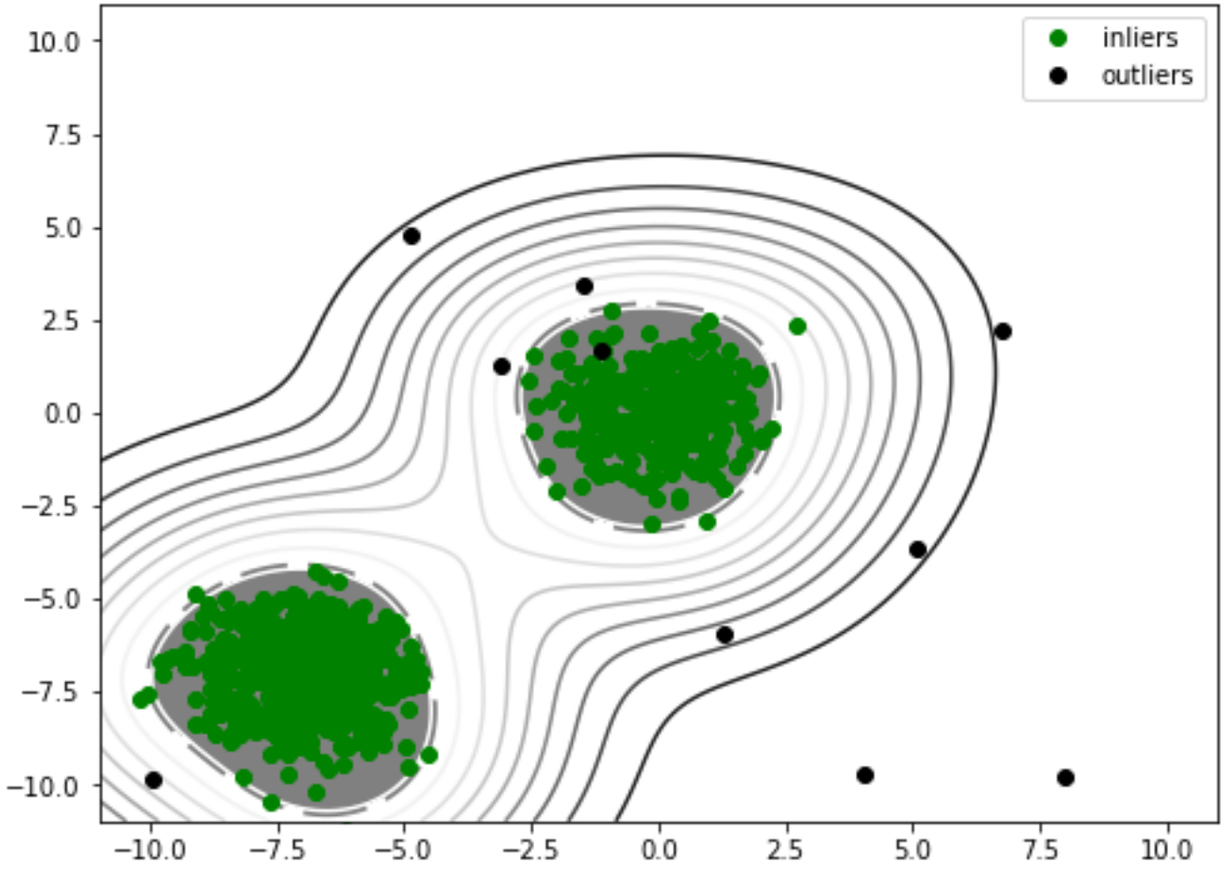
```
classifier = svm.OneClassSVM(nu=0.99 * outlier_ratio + 0.01, kernel="rbf", gamma=0.1)
classifier.fit(x)
```



Decision boundary for one-class SVM on bimodal synthetic data—trained using only inliers

```
x_0 = np.random.randn(num_inliers//3, num_dimensions)
x_1 = np.random.randn(num_inliers//3, num_dimensions) - 7
x_2 = np.random.randn(num_inliers//3, num_dimensions) - 7
x_rand = np.random.uniform(low=-10, high=10, size=(num_outliers, num_dimensions))
```

```
x = np.r_[x_0, x_1, x_2]
classifier = svm.OneClassSVM(nu=0.99 * outlier_ratio + 0.01, kernel="rbf", gamma=0.1)
classifier.fit(x)

```



ISOLATION FOREST

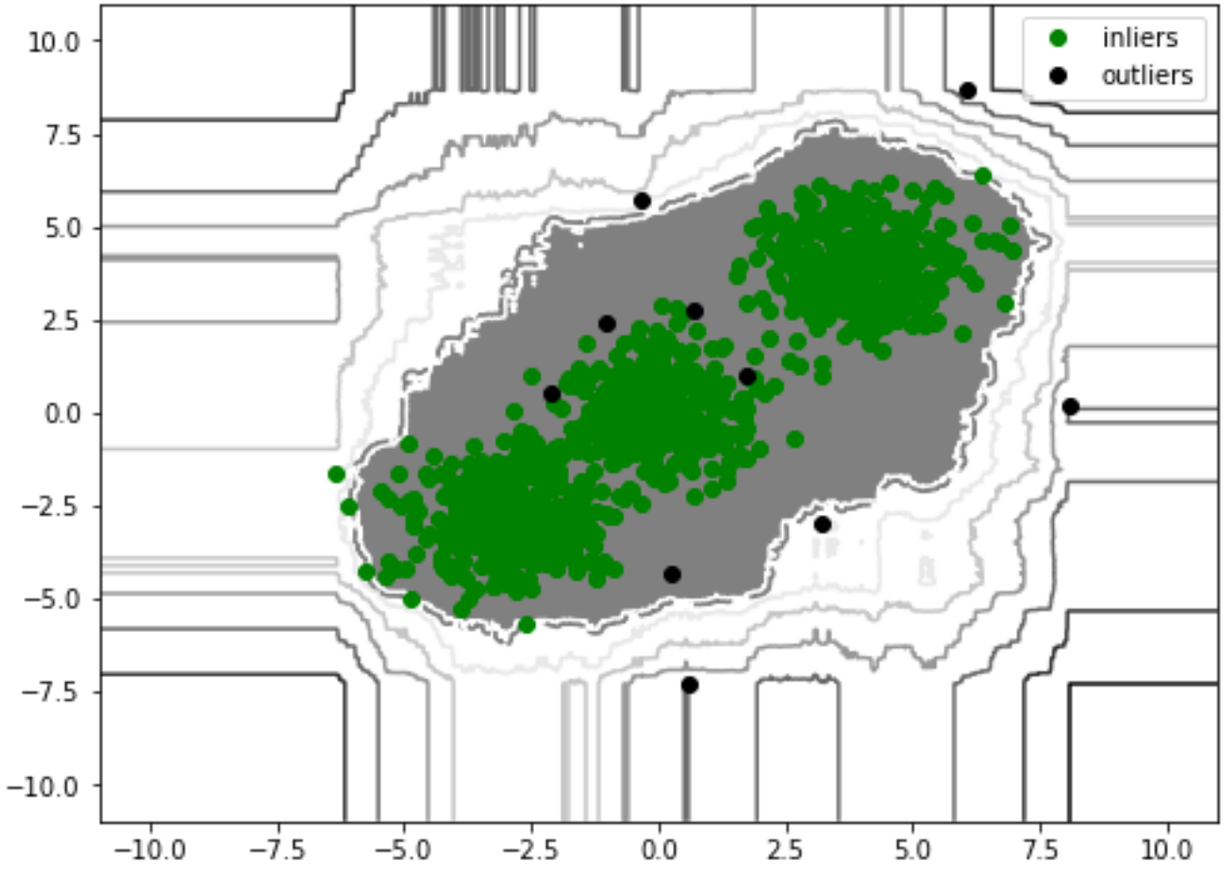
```
num_dimensions = 2
num_samples = 1000
outlier_ratio = 0.01
num_inliers = int(num_samples * (1-outlier_ratio))
num_outliers = num_samples - num_inliers
```

```
from sklearn.ensemble import IsolationForest

rng = np.random.RandomState(99)

classifier = IsolationForest(max_samples=num_samples, contamination=outlier_ratio, random_state=rng)
classifier.fit(x)
```

Parameter	Funktion
contamination	Definiert den Kontaminationsgrad; bei 0,01 gelten die kürzesten 1% der Wege als Anomalien

Decision boundary for isolation forest on synthetic non-Gaussian data

```
num_dimensions = 2
num_samples = 1000
outlier_ratio = 0.01
num_inliers = int(num_samples * (1-outlier_ratio))
num_outliers = num_samples - num_inliers
```



ADVERSARIAL MACHINE LEARNING

“**Adversarial machine learning** is the study of machine learning vulnerabilities in adversarial environments.” (S. 315)

Quelle: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.8156&rep=rep1&type=pdf

Explaining and Harnessing Adversarial Examples

Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy
Google Inc., Mountain View, CA
{goodfellow,shlens,szegedy}@google.com

Abstract

Several machine learning models, including neural networks, consistently misclassify *adversarial examples*—inputs formed by applying small but intentionally worst-case perturbations to examples from the dataset, such that the perturbed input results in the model outputting an incorrect answer with high confidence. Early attempts at explaining this phenomenon focused on nonlinearity and overfitting. We argue instead that the primary cause of neural networks' vulnerability to adversarial perturbation is their linear nature. This explanation is supported by new quantitative results while giving the first explanation of the most intriguing fact about them: their generalization across architectures and training sets. Moreover, this view yields a simple and fast method of generating adversarial examples. Using this approach to provide examples for adversarial training, we reduce the test set error of a maxout network on the MNIST dataset.

Practical Black-Box Attacks against Machine Learning

Nicolas Papernot
Pennsylvania State University
ngp5056@cse.psu.edu

Patrick McDaniel
Pennsylvania State University
mcdaniel@cse.psu.edu

Ian Goodfellow*
OpenAI
ian@openai.com

Somesh Jha
University of Wisconsin
jha@cs.wisc.edu

Z. Berkay Celik
Pennsylvania State University
zbc102@cse.psu.edu

Ananthram Swami
US Army Research Laboratory
ananthram.swami.civ@mail.mil

ABSTRACT

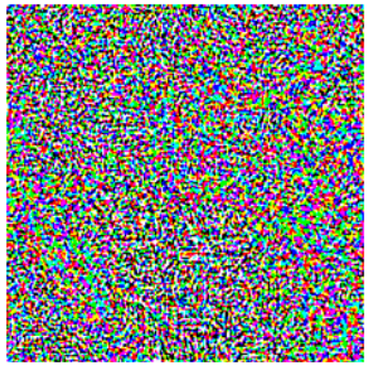
Machine learning (ML) models, e.g., deep neural networks (DNNs), are vulnerable to adversarial examples: malicious inputs modified to yield erroneous model outputs, while appearing unmodified to human observers. Potential attacks include having malicious content like malware identified as legitimate or controlling vehicle behavior. Yet, all existing adversarial example attacks require knowledge of either the model internals or its training data. We introduce the first practical demonstration of an attacker controlling a remotely hosted DNN with no such knowledge. Indeed, the only capability of our black-box adversary is to observe labels given

vulnerability of classifiers to integrity attacks. Such attacks are often instantiated by *adversarial examples*: legitimate inputs altered by adding small, often imperceptible, perturbations to force a learned classifier to misclassify the resulting adversarial inputs, while remaining correctly classified by a human observer. To illustrate, consider the following images, potentially consumed by an autonomous vehicle [13]:

Quellen:

- Practical Black-Box Attacks against Machine Learning (https://arxiv.org/abs/1602.02697)
- Explaining and harnessing adversarial examples (https://arxiv.org/abs/1412.6572)

$+ .007 \times$



$=$

$\boldsymbol{x}$

“panda”
57.7% confidence

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

“nematode”
8.2% confidence

$\boldsymbol{x} +$
$\epsilon \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
“gibbon”
99.3 % confidence

Quelle:

- Explaining and harnessing adversarial examples (https://arxiv.org/abs/1412.6572)

ATTACK MODEL

Influence	Specifity	Security violation
Causative attacks	Targeted end	Integrity attack
Exploratory attacks	Indiscriminiate end	Availability attack

Marco Barreno et al., “Can Machine Learning Be Secure?” Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security (2006): 16–25. (https://dl.acm.org/citation.cfm?doid=1128817.1128824)

MODEL POISONING / RED HERRING

MODEL POISONING / RED HERRING

- Angreifer verfügt über die Kontrolle einen Teil des Trainingsdatensatzes
- Einfluss wächst mit Anteil der Kontrolle über den Trainingsdatensatz
- beliebte Systeme (mit Machine Learning Applikationen) sind schwerer zu manipulieren => hoher Chaff* notwendig

---

“**Chaff**” (=Spreu) bezeichnet den Angriffsverkehr für die Poisoning (=Vergiftung) von ML-Modellen.

BEISPIEL: BINARY CLASSIFIER POISONING ATTACK

1. Zufälligen synthetischen Datensatz erzeugen

```
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=200, n_features=2, n_informative=2,
                           n_redundant=0, weights=[.5, .5], random_state=17)
```

2. Multilayer Perceptron setzen (MLP)

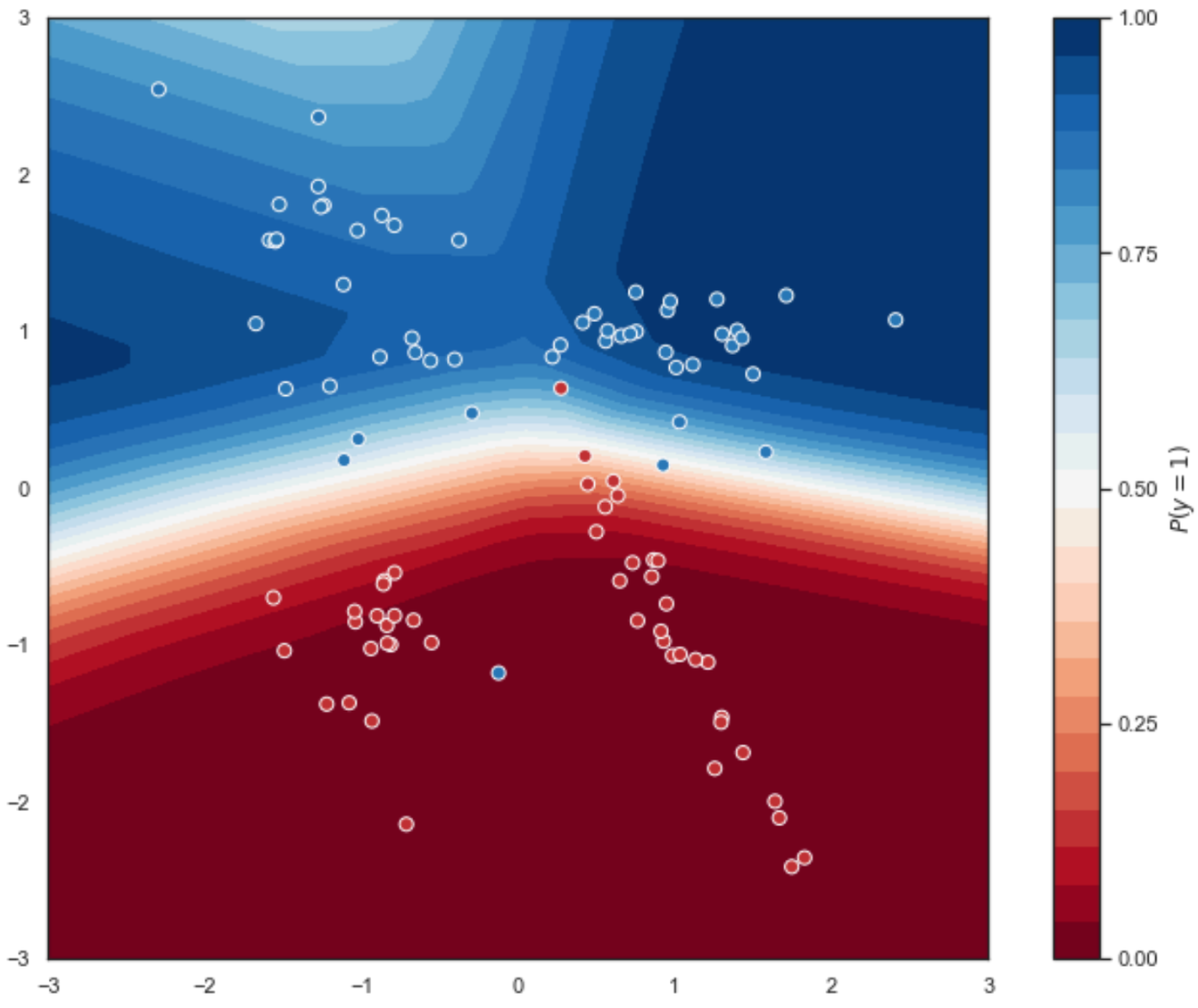
```
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(max_iter=600, random_state=123).fit(X[:100], y[:100])
```

3. Zweidimensionales Gitter von Punkten im Eingaberaum X, Y

```
xx, yy = np.mgrid[-3:3:.01, -3:3:.01]
grid = np.c_[xx.ravel(), yy.ravel()]
probs = clf.predict_proba(grid)[:, 1].reshape(xx.shape)
```

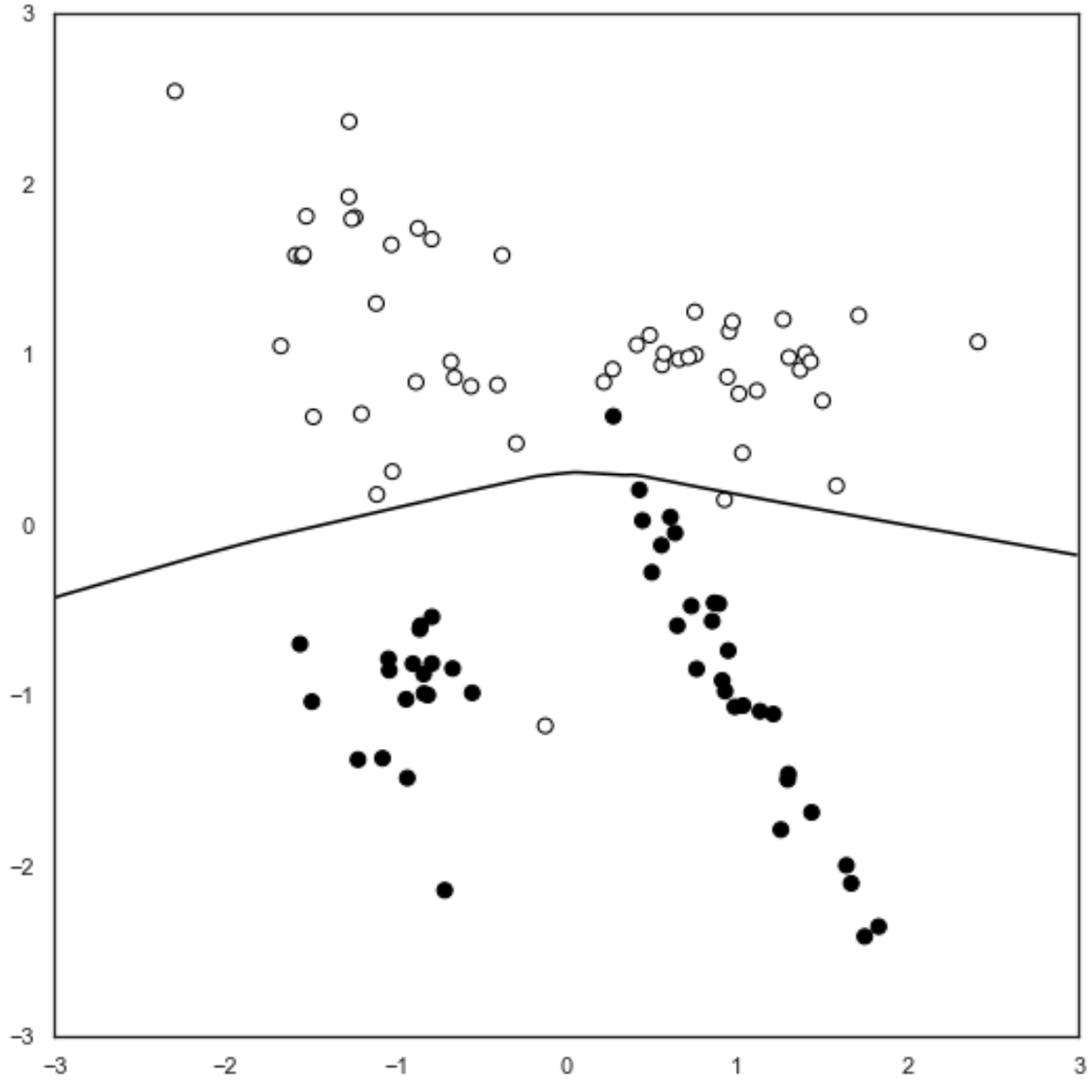
BEISPIEL: BINARY CLASSIFIER POISONING ATTACK

4. Konturdiagramm erzeugen und Testset plotten



BEISPIEL: BINARY CLASSIFIER POISONING ATTACK

5. Vertrauensschwellenwert (=Confidence Threshold) von 0,5 als MLP Classifier Decision Boundary



BEISPIEL: BINARY CLASSIFIER POISONING ATTACK

6. Poisoning Traffic: Intrusion von 5% Chaff in MLP Classifier

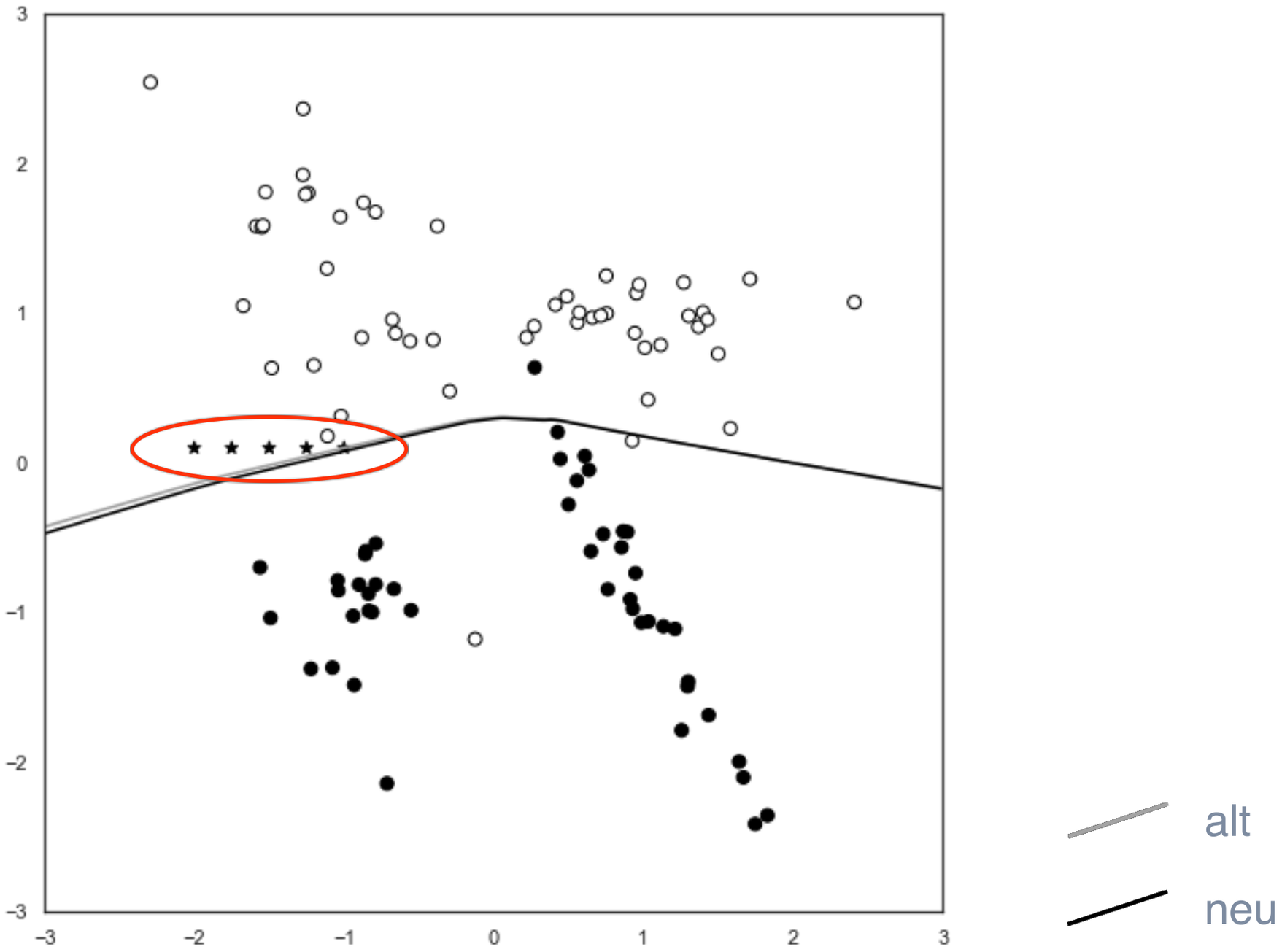
```
num_chaff = 5
chaff_X = np.array([np.linspace(-2, -1, num_chaff), np.linspace(0.1, 0.1, num_chaff)]).T
chaff_y = np.ones(num_chaff)
```

7. 1x Anpassen des MLP Classifiers an die 5 Chaffs

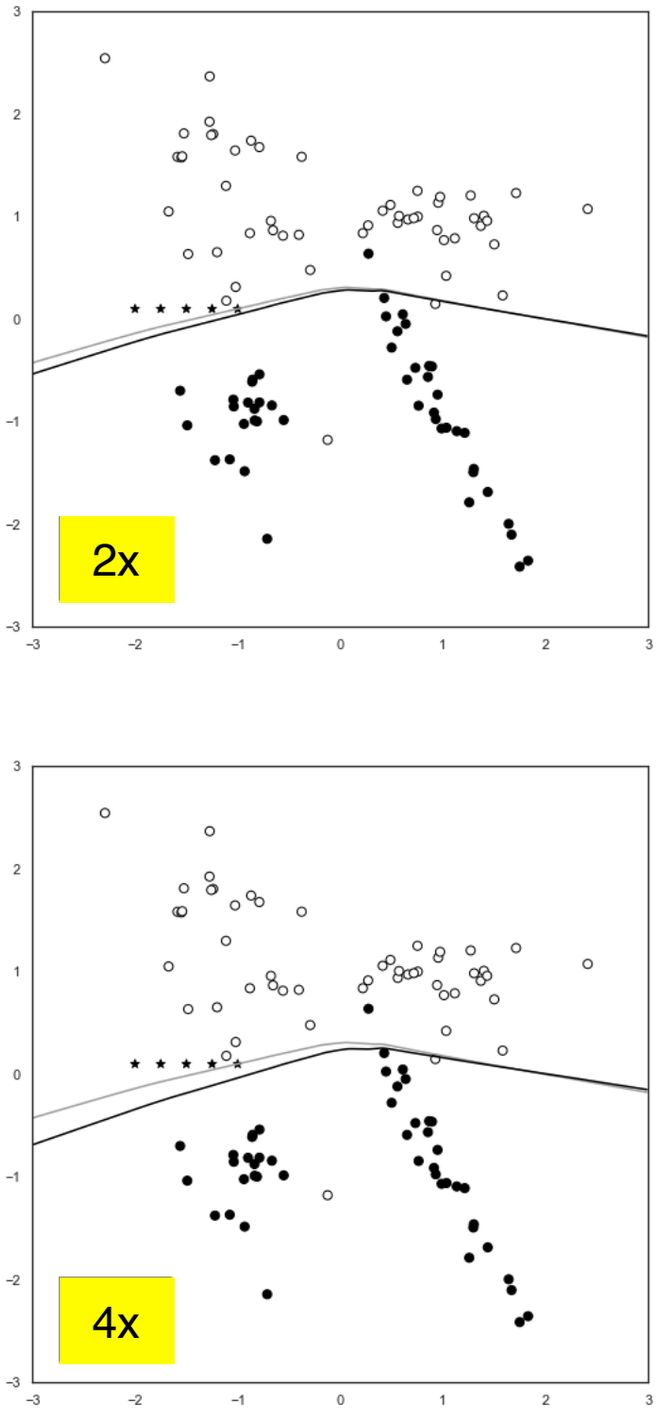
```
clf.partial_fit(chaff_X, chaff_y)
probs_poisoned = clf.predict_proba(grid)[:, 1].reshape(xx.shape)
plot_decision_boundary(X, y, probs, chaff_X, chaff_y, probs_poisoned)
```

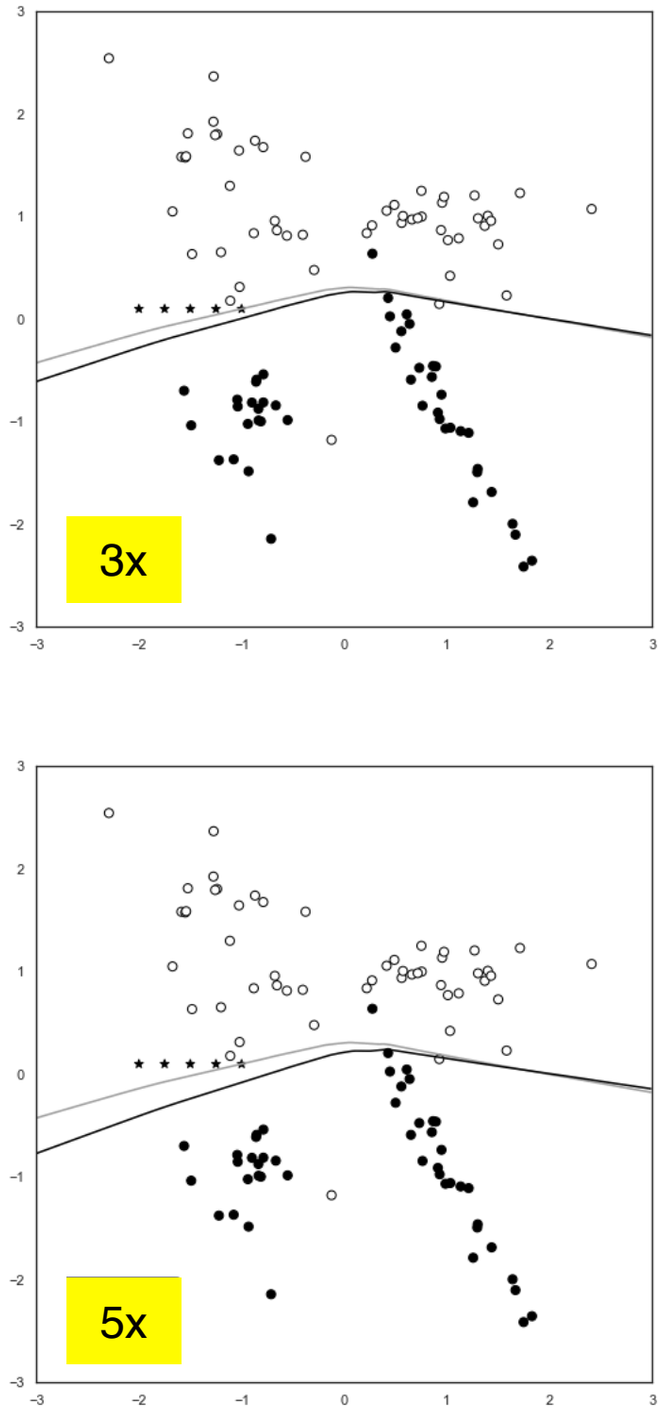
“**Chaff**” (=Spreu) bezeichnet den Angriffsverkehr für die Poisoning (=Vergiftung) von ML-Modellen.

BEISPIEL: BINARY CLASSIFIER POISONING ATTACK



BEISPIEL: BINARY CLASSIFIER POISONING ATTACK





alt

neu

Danke für die Aufmerksamkeit!

Repository für Jupyter Notebook und Slides:

github.com/tvotan/MLUGS_ml_and_security

linkedin.com/in/tinvotan/

@tin_votan