Banknote Authentication Design Considerations

Machine Learning for Security and Performance in Production Settings

Alice Chang, Karthik Umashankar, Yu Chen

June 7, 2018

Abstract

Banknote authentication is a high-value task within the financial services industry. In this paper, we perform a validation study and probe into potential vulnerabilities of online machine learning models utilized for this task. In particular, we utilize the work of "Banknote Authentication" by Gillich and Lohweg[1]. The authors utilized an industry-standard technique of filtering the banknote grayscale image for viable regions of interest, and then performing a novel Shift Invariant Wavelet Transform for the model's feature engineering. We first validate Gillich and Lohweg's findings of 100% accuracy on holdout datasets using a multitude of models (k-Nearest Neighbors, Support Vector Machine, etc.), and then identify critical thresholds where sample size begins to negative impact the potential performance of a machine learning model once deployed into production.

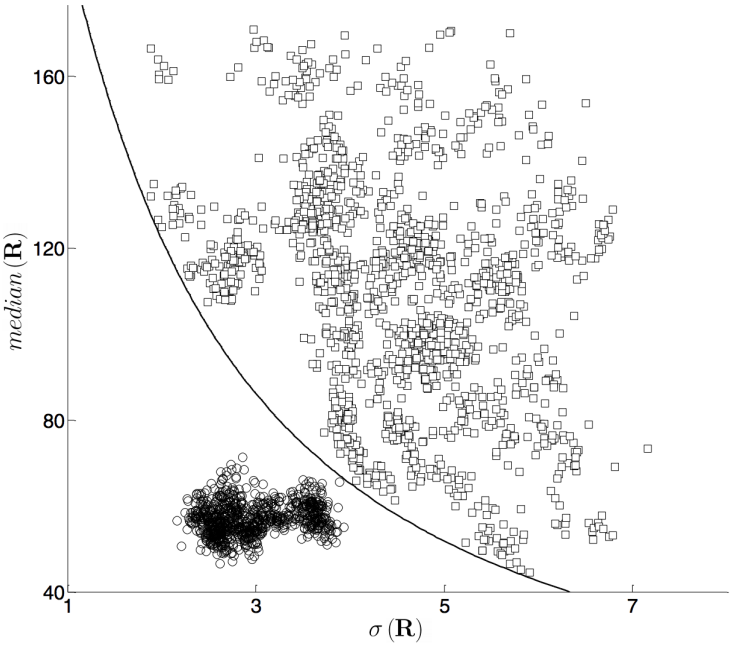
We conclude by investigating potential vulnerabilities an online machine learning system can encounter, specifically the Frog Boiling Attack. We outline a potential implementation of this type of penetration attack. In our experiments, depending on the classifier that was chosen, we were able to get the system to successfully mis-predict a forged data point as authentic in as few as 50 iterations. The complexity of the classifier didnt offer better protection against the attack. We saw that the simple k-Nearest-Neighbor performed better than the Support Vector Machine & Gaussian models.

We also highlight various potential countermeasures to mitigate its negative impact on a machine learning system in production.

1 Introduction

The Intaglio technique used within banknote printing allows for an unmatched sharpness in contrast through a feelable relief[2]. This technique forms a vital component of modern-day banknote authenticity verification. In the initial

Figure 1: A scatter plot from Gillich and Lohweg, 2010 showing the decision boundaries for selection of viable regions of interest. The circles are classified as appropriate ROIs while the squares are discarded for non-homogeneity or high mean grayscale values.



phase of authentication, **image digitization** occurs where the banknote's relief is captured through industry-grade cameras to produce an image grayscale that can be then undergo a feature transformation, such as the Wavelet Transform (WT). Prior to feature transformation, various **Regions of Interest (ROI)**. Regions with low mean and variance in grayscale values of the various blocks are selected:

2 Related Work

3 Experiments

3.1 Validation of Model Performances

In their original paper, Gillich and Lohweg assert that 100% accuracy can be achieved as a result of judicious Region of Interest selection and the Shift Invariant Transform. To validate, we performed basic data processing, scaling the dataset so that each feature column k had $\mu_k = 0$ and a $\sigma_k = 1$:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

df = pd.read_csv("data/dataset.csv", encoding="latin-1") # import the
    CSV dataset and save as a Pandas dataframe object

targets = df["class"].values # save the targets (Y) as a NumPy array of
    integers
features_df = df.loc[:, df.columns != "class"] # save everything else as
    the features (X)

features_names = features_df.columns.values # save the column names
features_matrix = features_df.values # convert X from a dataframe to a
    matrix (for our machine learning models)
scaled_features_matrix = StandardScaler().fit_transform(features_matrix)
    # scale the data so mean = 0, unit variance - important for many
    models
```

The cleaned `scaled_features_matrix` Numpy object can then be fed into a variety of different models:

```
logistic_regression = LogisticRegression()
logistic_regression.fit(X_train, y_train)
training_predictions = logistic_regression.predict(X_train)
test_predictions = logistic_regression.predict(X_test)

# check accuracy of Logistic Regression
number_correct = np.sum(training_predictions == y_train)
number_correct_test = np.sum(test_predictions == y_test)
```

3.2 Sample Size Stress Testing

3.3 Implementation of Frog Boiling Attack (FBA)

3.3.1 Overview

The Frog Boiling Attack is most effective during business contexts where online learning occurs (when data arrives as a stream and becomes ingested sequentially, with the fitted model updated after each new data point). The attack relies upon the concept of template drift[3], where the model's fitted "template" of an authentic banknote data sample is corrupted over time by a stream of compromised data points. The overall implementation, implemented over N steps, relies upon the update equation

$$F_{new} = F_{old} + (i-1)\delta(x) \tag{1}$$

where

$$\delta(x) = \frac{\bar{x}_{target} - \bar{x}_{original}}{N} \tag{2}$$

3.3.2 Experimental Procedure

In our experiments, we determined that the value of $\delta(x)$ is quite crucial in mounting a successful frog boiling attack. The idea behind the attack is to pass off forged banknotes as authentic, so, we should try to get the classifier to determine as many forged notes in the test data set as possible, as authentic. For this reason, for the remainder of this paper, when we refer to the authentic data set (A_{data}), it refers to the records in the training set that were classified as authentic, and when we refer to the forged data set (F_{data}), it refers to the records in the test set that were classified as forged.

In each iteration, we start off by calculating the mean (centroid) of all values in F_{data}, and find the point in A_{data} thats closest to the centroid, lets call it CAD (closest authentic data point). We then generate a new fake data point (FDP) by adding a tiny incremental value to the CAD. This incremental value is given by the formula

$$\frac{D_{CAD,TeF}}{N} \tag{3}$$

where $D_{CAD,TeF}$ is defined as the Euclidean distance between the closest authenticated data point and the center of the trained forged data.
We then add this new FDP back into A_{data}, manually assign it a classification of authentic, refit the model, and predict the classification of the FDP. If the classification is forged, we stop right there (as our attack is compromised). If the classification is authentic, we proceed to the next iteration.

With the above-mentioned approach, we were able to successfully "attack" the system into incorrectly predicting a forged banknote as authentic. We ran the algorithm against five different classifiers, and some of them displayed a stronger aversion to false detection than the others, but with enough iterations, we were able to crack them all eventually (with the exception of the Random Forest classifier). The tables below detail the results, with the highlighted row indicating where we saw a drop in the accuracy.

3.3.3 FBA Experimental Results

	Linear SVC					
	Training Set			Test Set		
	Count	Correctly Predicted	Accuracy	Count	Correctly Predicted	Accuracy
Initial Run	1097	1088	99.18%	275	271	98.55%
After 50 FBA iterations	1108	1096	98.92%	275	270	98.18%
After 100 FBA iterations	1123	1106	98.49%	275	268	97.45%
After 200 FBA iterations	1167	1150	98.54%	275	268	97.45%
After 500 FBA iterations	1363	1333	97.80%	275	266	96.73%

	Neural Network					
	Training Set			Test Set		
	Count	Correctly Predicted	Accuracy	Count	Correctly Predicted	Accuracy
Initial Run	1097	1097	100.00%	275	275	100.00%
After 50 FBA iterations	1112	1112	100.00%	275	275	100.00%
After 100 FBA iterations	1197	1197	100.00%	275	274	99.64%
After 200 FBA iterations	1297	1297	100.00%	275	274	99.64%
After 500 FBA iterations	1597	1597	100.00%	275	274	99.64%

	Gaussian					
	Training Set			Test Set		
	Count	Correctly Predicted	Accuracy	Count	Correctly Predicted	Accuracy
Initial Run	1097	1094	98.55%	275	273	99.27%
After 50 FBA iterations	1117	1111	99.46%	275	273	99.27%
After 100 FBA iterations	1154	1148	99.48%	275	270	98.18%
After 200 FBA iterations	1297	1286	99.15%	275	269	97.82%
After 500 FBA iterations	1597	1578	98.81%	275	267	97.09%

	KNN					
	Training Set			Test Set		
	Count	Correctly Predicted	Accuracy	Count	Correctly Predicted	Accuracy
Initial Run	1097	1096	99.91%	275	275	100.00%
After 50 FBA iterations	1147	1145	99.83%	275	274	99.64%
After 100 FBA iterations	1197	1193	99.67%	275	274	99.64%
After 200 FBA iterations	1297	1293	99.69%	275	274	99.64%
After 500 FBA iterations	1597	1593	99.75%	275	274	99.64%

	Random Forest					
	Training Set			Test Set		
	Count	Correctly Predicted	Accuracy	Count	Correctly Predicted	Accuracy
Initial Run	1097	1097	100.00%	275	274	99.64%
After 50 iterations	1106	1105	99.91%	275	274	99.64%
After 100 iterations	1144	1144	100.00%	275	274	99.64%
After 200 iterations	1135	1135	100.00%	275	275	100.00%
After 500 iterations	1219	1219	100.00%	275	275	100.00%
After 1000 iterations	1339	1339	100.00%	275	275	100.00%
After 2000 iterations	1458	1458	100.00%	275	275	100.00%
After 5000 iterations	2470	2470	100.00%	275	274	99.64%
After 10000 iterations	3859	3858	99.97%	275	275	100%

3.3.4 Analysis of FBA Experimental Results

- With the Neural Network and the KNN classifiers, since we drifted A_{data} towards the mean of F_{data}, post the attack, it was only the data point closest to the mean that was classified as authentic. For the attack to succeed on several data points, we need to run it multiple times, once each towards the forged datapoint we want the system to misclassify.
- With the Gaussian and the LinearSVC classifiers, with increasing iterations, we were able to classify more of F_{data} as authentic.
- The LinearSVC model chosen by the original researchers performed the worst, with as many as 9 mispredictions post 500 iterations of the attack.
- The Random Forest classifier really surprised us by almost always thwarting the attack at around the 30% mark of the number of iterations. The initial count for the number of data points in the A_{data} is 1097. For classifiers like KNN (that never detected the "corrupt" data points we were introducing, as forged) you can see that the end of n iterations, the number of data points in the A_{data} set had increased to $1097 + n$. Whereas in the case of Random Forest, even setting the iteration count to as high as 10000 only resulted in the classifier detected the corrupt data point after 2700 iterations. Not only that, but its test and training accuracy seemed to get better the more iterations we ran.
- We were able to reproduce the attack by starting with the mean value of A_{data}, rather than choosing the point closest to F_{data}, but it required significantly higher number of iterations (20x) to achieve the same result.

3.3.5 FBA Countermeasures

- **Freezing Online Learning**: When discussing potential countermeasures to FBA, a tradeoff between responsiveness and security must be estab-

lished. An obvious, yet sometimes unrealistic, method of prevention is to fully disable online learning completely, and "freeze" the authentication classifier. In many businesses cases, disabling online learning may result in a significant decline in model performance, especially if the feature space is exceptionally dynamic (ie. consumer entertainment and purchasing behavior, which is often a function of seasonality and inherent "drift").

- **Autocorrelation Detection**: An "honest" user (one who is utilizing the business' services as intended) should display no correlation between her submitted banknote data points. No correlation between a user's data points over time equates to each banknote being submitted independently of the prior banknotes (which should be the case if no fraudulent activity is happening).

- **Batch Model Updates**: The crux of the Frog Boiling Attack is the assumption that the model will not be able to detect small "drift" towards a target template, capitalizing upon the inherent responsiveness of the model towards new data points. However, batching model updates increases the magnitude of "drift", with the hope being that the model is able to successfully detect larger "steps" if we perform model updates in batches. The pseudocode implementation for batching would be as follows:

```
for user in users:
   BATCH_SIZE = 30 # update after user has submitted 30 banknotes
   batched_datapoints = [] # create a cache of datapoints per user
   authenticated = receive_data(banknote_datapoint) # ingest
       datapoint when user submits banknote

   if banknote_datapoint is authenticated:
      batched_datapoints.append(banknote_datapoint)
   else:
      batched_datapoints = [] # if any authentication points fail,
          clear the cache

   if length(batched_datapoints) == BATCH_SIZE: # once cache hits
       batch size, update the global model
      update_model(batched_datapoints)
```

Within this particular implementation, a key hyperparameter to optimize is `BATCH_SIZE` This is the number of drift steps taken from the original template to the target template *without updates (ie. model refitting)* before the model returns a negative classification. For instance, in our implementation of frog boiling attack, disabling the online learning model refitting and thereby freezing the model, we receive the following log output:

```
Closest positive point: [ 0.13 0.01 -0.54 0.24]
Negative centroid: [ 0.65 0.40 -0.14 0.02]
```

```
Distance from negative centroid: 0.79
Prediction: [0]
Classified as negative during iteration 10
```

The simple k-Nearest Neighbors model will accept drift up to 10 steps away before it detects a fraudulent banknote datapoints. Thus, we'd likely set our `BATCH_SIZE` to a number significantly higher than 10. This number can be set by repeatedly probing the authentication model with online learning disabled to generate a distribution of **tolerance thresholds**, or the number of steps taken before the machine learning begins to classify data points as negative. Say we have taken 50 different probes of our model, and the following array of tolerance thresholds are returned:

```
[ 7., 4., 2., 9., 13., 9., 10., 9., 13., 8., 8., 14., 8.,
10., 3., 12., 9., 15., 12., 8., 8., 13., 10., 5., 8., 15.,
6., 14., 8., 8., 15., 6., 6., 6., 15., 5., 11., 10., 9.,
13., 10., 13., 7., 9., 8., 11., 15., 8., 10., 16.]
```

We can fit these results to a Gaussian distribution, and then find its inverse cumulative distribution function (ICDF) at an acceptable α level. In the context of banknote authentication, given the sensitive nature of financial assets at risk, a relatively stringent α can be selected of 0.001:

```python
from scipy.stats import norm
alpha = 0.001
fitted_mean = 10
fitted_standard_deviation=3
threshold = int(norm.ppf(1-alpha, loc=fitted_mean,
    scale=fitted_standard_deviation))
print(f"Batch size at alpha level {alpha} is {threshold}")
# Batch size at alpha level 0.001 is 17
```

Thus, a `BATCH_SIZE` of 17 is selected for production deployment to minimize the risk of online learning intrusion attacks like FBA.

3.3.6 FBA Optimizations

- **Markov Chain Monte Carlo / Metropolis-Hastings Algorithm**:

4 Future Work

References

[1] Eugen Gillich and Volker Lohweg. *Banknote Authentication*. inIT - Institute Industrial IT, Department of Electrical Engineering and Computer Science, Ostwestfalen-Lippe University of Applied Sciences November 2010

[2] Van Renesse, R.L.: Optical Document Security, 3rd edn., Artehouse Boston/London (2005)

[3] Z. Wang, A. Serwadda, K.Balagani, and Vir V. Phoha, "Transforming animals in a Cyber-Behavioral Biometric Menagerie with Frog-Boiling Attacks" in *The IEEE Fifth International Conference on Biometrics: Theory, Application and Systems (BTAS 2012)*, Washington DC, 2012