Steganalysis of multiple overlapping images

Dan-Andrei Gheorghe, supervised by Andrew D. Ker

May 20, 2016

Abstract

One goal of steganalysis is to recover additional information pertaining to the existence of a hidden message in a digital image. A likely situation in practice (given that there is no cost associated anymore with taking multiple pictures) is for the steganographer to have used images taken with the same camera under the same conditions, of the same scene. We explore in laboratory conditions whether such a situation aids us in building a better model via linear regression that can predict the size of the payload hidden inside such images. Previous work for classification, rather than regression, has shown that a technique knows as calibration improves prediction with overlapping images and now we try to extends these results. We reason this is possible, by exploring a new hypothesis, that steganographic features can be represented as the sum of components corresponding to multiple noise sources (camera, scene, time noises and stego signal), some of which are shared by overlapping images and cancelled via calibration. When we explore this, we see that calibration reduces one source of noise, but increases another and our expectation is that it will lead to a net gain. We will find this is the case.

Contents

Chapter 1

Introduction

1.1 Steganography

Steganography is the art of hiding information [10] inside apparently innocent media (for example: images, videos, music, text), while **steganalysis** is the attempt to discover the existence of such information with a certain degree of reliability. It is well known that hiding is at its safest when the medium used consists of independent parts, yet it often happens in practice for the steganographer's covers to contain overlapping parts. Can we use this fact to make better estimates about the payload of a digital image? That is the question which this project will try to answer.

Steganography is typically illustrated by Simmons Prisoners' Problem [15], in which Alice and Bob, two separated prisoners, discuss an escape plan through a medium monitored by a Warden. The goal of this kind of communication is to ensure its **Secrecy**, so that the Warden remains unaware of the existence of hidden information, since then he would be able to take measures against it.

Here is some common terminology used in this subject: the message which Alice wants to send is known as the **payload**, the possible media that she can use are regarded as the **cover objects**, while one where a payload has already been hidden is a **stego object**. Alice will be considered the steganographer. Usually, Alice and Bob would have shared a secret key beforehand, of which the Warden is unaware, but it will be seen later that this will not matter for the project.

Because of their wide usage, we will only focus on digital media as cover objects, and more specifically on RAW images, since they have a huge capacity for embedded payload, without the complex enough structure of music or videos files for steganography to become unwieldy. A random payload assumption will be assumed throughout this report: the payload is a random sequence of bits indistinguishable from random coin flips, and since this property will be maintained by encryption, compression or any other operations that are applied before the message is sent, there is no need to care about the existence of a secret key (we say that we simulate embedding).

1.2 Steganalysis

Steganalysis represents the other side of the problem, that of the Warden, or the steganalyst, who is the enemy of the steganographer. Usually, the goal is to identify the objects with payload hidden inside them, while reducing the rate of false positives (cover objects considered stego) and that of false negatives (stego objects considered covers). There can be a active Warden, one who will tamper with the media objects, or a passive one, who can only inspect them, but our focus will only be on the latter.

As an adaptation of the Kerckhoffs' Principle, which can be summarized by "Assume the enemy knows the system", we consider that the Warden is aware of the exact embedding algorithms that are used, not an entirely unlikely situation in practice. Furthermore, he does not know anything about the secret key (to maintain the random payload assumption), nor about the payload.

There are two ways in which the Warden could analyse the objects he encounters: the first one involves building a **classifier**, which tries to detects whether the objects are covers or stego; the second one involves a **regressor**, which will try to estimate the amount of payload hidden inside the objects, usually as a ratio of the maximum possible payload size. In the case of the latter, 0 or negative values can be interpreted as no payload being embedded.

1.3 Steganalysis of Overlapping Images

In this specific case of Steganalysis, **Overlapping Images** either have been used as covers or the Warden is capable of creating them. This allows us to make stronger assumptions then previously, namely that the Warden has access to the exact same camera (or the same model) used for taking the covers and that he is capable of developing his own training data. He also has access to the test objects and is aware of the camera characteristics used, so he will now be able to recreate them.

In this project, machine learning will be used to gain additional data about the stego images, namely we will attempt to train regressors capable of predicting the amount of hidden payload in each stego image. Since building an exact model of images is usually impossible (hence why structural steganalysis, which attempts exactly that, is considered infeasible), machine learning applied in steganalysis is always done using their features, multidimensional vectors that describe their general characteristics using statistical properties, filters, projections. Similar endeavours regarding overlapping images have been undertaken in the past, mostly with classifiers for JPEGs [1] and for RAW images [17]. We will be comparing how a regressor trained to detect the difference in payload of two overlapping images would fare against one that estimates the payload itself and try to improve the performance of the former.

Chapter 2

Background

2.1 Overlapping Images

A realistic situation that will occur in practice is for the steganographer to use a library of digital images taken with the same camera, possibly multiple ones of the same object, and embed in part of them. In this case we encounter overlapping images, where significant parts of the content will be similar (hence the overlap). This might happen because the camera settings will

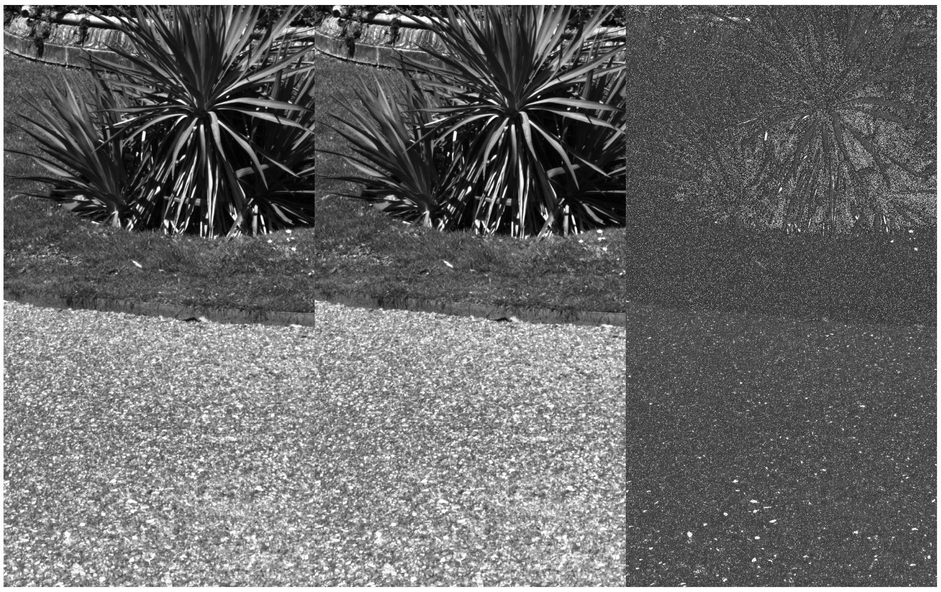


Figure 2.1: Two Overlapping Images and their Pixel-wise Difference

be reused, and taking multiple captures of the same scene is not a problem any more as storage capacity for digital media is nearly limitless. As for the photographer, it is likely for him to have retakes in order to select the best photo later and multiple photos (with significant overlapping content) for a panorama.

Even so, it is important to note that overlapping images need not to have identical content, due to camera orientation, changes over time in the scenery, and of physical phenomenons happening when capturing the scene (we can see this in figure 2.1).

Given this scenario, we will examine whether and how can we use the overlapping images in order to improve the steganalysis for lossless images.

2.2 Embedding Algorithms

2.2.1 LSBM - Least Significant Bit Matching

LSB Matching (also known as ± 1-embedding) is a steganographic algorithm that embeds the payload using the least significant bits of each pixel in the cover image, making changes so that each of them will be equal (modulo 2) to the corresponding bit in the payload. Whenever they do not match, it will randomly increment or decrement the value of the pixel, both with equal probability, to avoid structural attacks (which its predecessor, LSB Replacement, suffered from). A first description of this was done by Toby Sharp in 2001 [14]. An implementation of the algorithm would translate in the following formula:

$$s = \begin{cases} c, \text{ if } c = m (\text{mod } 2), \\ 1, \text{ otherwise and } c = 0, \\ 254, \text{ otherwise and } c = 254, \\ c \pm 1, \text{ otherwise, randomly with 0.5 probability.} \end{cases}$$

When embedding a random payload of size p, we would only need to change approximately $p/2$ pixels (since the others would have the correct bit). Then the formula that simulates LSBM embedding with a random payload of size p would be:

$$s = \begin{cases} c, \text{ with probability } 1 - p/2; \\ 1, \text{ with probability } p/2, \text{ if } c = 0; \\ 254, \text{ with probability } p/2, \text{ if } c = 255; \\ c + 1, \text{ with probability } p/4, \text{ otherwise;} \\ c - 1, \text{ with probability } p/4, \text{ otherwise.} \end{cases}$$

Implemented as a Python Script for simulating embedding on pgm files, this would translate to:

```
import numpy
import os
import io
import sys

...
p=numpy.random.randint(1,100)
p=p*0.01

f=io.open(inputFilename,'rb')
g=io.open(outputFilename,'wb')

g.write(f.read(16))
pixels=bytearray(f.read())
p=p/2 #and this is the true rate of change
i=0
pixelsLen=len(pixels)
while(i<pixelsLen):
    r=numpy.random.random()
    if(r<p/2):
        if(pixels[i]==0):
            pixels[i]=pixels[i]+1
        else:
            pixels[i]=pixels[i]-1
    elif(r>1-p/2):
        if(pixels[i]==255):
            pixels[i]=pixels[i]-1
        else:
            pixels[i]=pixels[i]+1
    i+=1
g.write(bytes(pixels))

f.close()
g.close()
```

2.2.2 HUGO - Highly Undetectable SteGO

HUGO is a steganographic algorithm for spatial-domain digital images designed by Tomáš Pevný, Tomáš Filler, and Patrick Bas, which uses high dimensional models to perform embedding [12]. It takes the cover, computes the distortion value at every pixel and assigns the probabilities for each pixel to change based on it. It also supports model correction, which is an attempt to estimate the distortion after every change. It is important to note that the distortion function used has been designed with SPAM (Subtractive Pixel Adjacency Matrix) features in mind, being resistant to steganalysis based on them. The payload itself is encoded with Syndrome Trellis Codes.

Even while being Highly Undetectable, it can be defeated just by using a different set of features for the steganalysis, as the BOSS contest has proved [4] (additionally, both SRM [3], which are similar to the ones that won the contest, and PSRM [6] are effective against HUGO).

2.3 Features

Machine learning techniques usually require features that represent the data that needs to be analysed, so in order for us to make use of them in our steganalysis experiments, we need to have **features** that describe every image of our data set. The ones that are going to be used in this case are the **PSRM** (Projected Spatial Rich Model) features, described in [6]. They attempt to model the noise of each image, by obtaining the noise residuals (usually defined as the difference between pixel and predicted value), projecting them randomly on neighbourhoods and then computing the first-order histogram. Sadly, their are somehow impractical, mainly because of their high calculation time (20 minutes per Mpix image).

In consequence, we use GPU-PSRM [9], an implementation that changes the features slightly to make them more efficient to compute, which then exploits the power and parallelism of GPU Hardware in order to reduce the extraction time. There will be only 20 kernels per residual, giving a 4680-dimensional feature vector, which will be reasonably useful for the regressors (a lower dimensionality makes it faster to train them, without any significant drops in the accuracy). The code for their extraction has been run by the supervisor on clusters of Oxford University's Advanced Research Computing (ARC) facility, since how the features are computed is not in the scope of the project.

2.4 Hypothesis

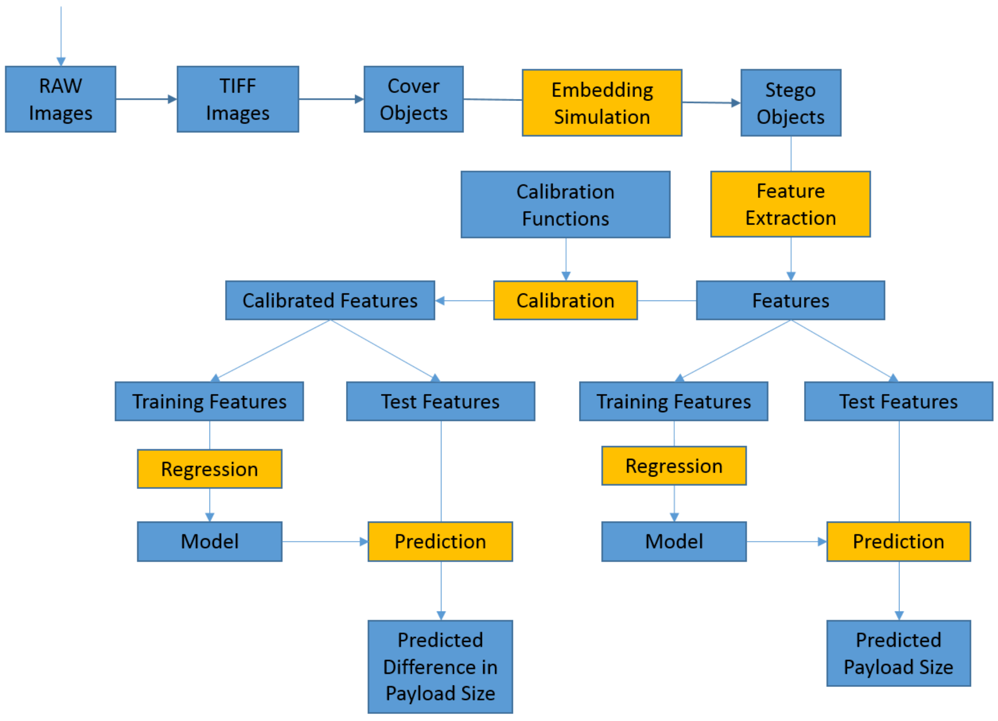
It is very common in steganalysis to think that every image contains noise (the image noise or cover content) and possibly a signal (which confusingly is called stego noise) and, in order to solve the problem (classification or regression), some processing is done to suppress the former and enhance the latter. Now, the cover content can be described in terms of three noise sources: a camera noise (its ”fingerprint” actually, caused by the model and subtle, but unique differences in lenses and the capture mechanisms; also, the settings of the camera may slightly alter it), a scene (or space) noise (caused by the position and orientation of the camera, which identifies the scene taken) and a time noise (mainly because very few scenes are actually static, light conditions varying with the passing of time and movements may occur in the scene). We also consider the signal to be only caused by the embedding algorithms that the steganographer will use.

The purpose of features is to suppress the image noise (which is why many developments in steganalysis have focused on developing better filters to be used in their computation and to design high-dimensional ones so that they become more efficient). Calibration is used alongside them in order to further enhance the stego noise and it has been used successfully on both JPEGs [1] and RAWs [17] to better solve classification.

Now we hypothesise that calibration which uses overlapping images as reference objects will not only improve the signal, but it will also reduce camera and scene noise albeit with some increases in time noise. Since those two represent the most significant natural components of the cover content, we should be able to use that in building better regressors for lossless images that will more accurately detect the steganographic signal. Therefore, when we perform regression on overlapping images, we expect the estimates of payload difference between two images (from the same scene) to be more accurate than absolute estimates of payload size. In our experiments, we will measure this accuracy in terms of bias, absolute error and mean square error.

Chapter 3

Experimental Design



3.1 Data Set

A Canon PowerShot G16 was used for taking the photos that make the data set for our experiments. The camera has been placed on a tripod in 103 different positions, with a certain orientation (which we will refer to as a scene). For every scene, the focal distance, zoom, and the ISO (light sensitivity) are constant, the only variable being the light exposure, chosen to avoid having too dark or bright areas in the picture (the latter make embedding artificially more detectable). Multiple pictures have been taken per scene with the aid of a timer and only 500 pictures were kept for the data set. Thanks to fast SD memory cards, a photo could be taken every second and each scene was done in about 10-15 minutes. Obtaining the entire data set (which amounts to 50k photos occupying 1.5TBs) required 20 hours of continuously taking pictures over a time span of several weeks.

After curating the data, the pictures had to be converted from CR2 (the proprietary Canon Raw Format) to TIFF (Tagged Image File Format) using RawTherapee, and then to PGM (Portable GrayMap, a grayscale lossless format part of the Netpbm project) using ImageMagick. We need the multiple conversions, because the implementations for the LSBM and HUGO algorithms simulate embedding only in PGM files. The only difference between uncompressed grayscale and colour images is the number of components per pixel (brightness for the former and red, green, and blue for the latter), the steganography and steganalysis being the same, therefore the results could be extended to colour formats (but it is beyond our scope). Each image has been divided in 10 slices of resolution 812 X 1518 in order to increase the size of our data set.

The embedding simulation in the covers using LSBM was done with a personal Python implementation of the algorithm, while the HUGO embedding simulation uses Binghampton University's DDE Lab's C++ implementation [2]. In both cases, a random payload has been chosen (for LSBM, values between 0 and 1, for HUGO, between 0 and 0.4).

To perform the experiments themselves, only the PSRM features and the payload corresponding to each slice are necessary and, for ease of access, they have been stored as matrices and vectors in the NumPy array format (this being the Python library which we will use for array and matrix computations).

3.2 Regression

Prediction is the problem of modelling the relationship between a dependent variable y and one or more independent variables $x_1, x_2, ...x_n$, while **regression** is an approach that given sample (or training) data tries to build a model. This model would then later be used on a different set of (test) data, in order to make predictions about the dependent variable. We need to proceed in this way, because, in general, we can not actually observe the true model and have no other way of measuring the results.

This is useful in steganalysis, since we set the dependant to be the relative payload size (or a function of it) for an image, and we then model it in terms of the corresponding features (which will act as the independent variables). We measure the payload in terms of bits per pixels (bpp) and, so that the results are more accurate, all images will be of the same resolution.

In the case of linear regression (which is the one we use in our experimental design), we assume this dependence can be captured by a linear model, as following:

$$y = \bar{\beta} \cdot \bar{x} + \epsilon,$$

where $\bar{x} = [1, x_1, x_2, ...x_n]$ is the vector of independent variables (note that we have added a 1 to capture the relationship when y is independent from the variables), $\bar{\beta}$ is the vector of parameters that describes the linear dependence, and ϵ is a random variable that represents the error term. For this to be useful in practice, $E[\epsilon] \simeq 0$ is needed (otherwise, we are dealing with a biased estimator), meaning that the predictor we get will be the line

$$\hat{y} = \hat{\beta} \cdot \bar{x}.$$

As for how we obtain the parameters $\bar{\beta}$ via regression on training data, the exact approaches we will consider are Ordinary Least Squares and Ridge Regression, while the NumPy Linear Regressor will be used for comparison.

3.2.1 Ordinary Least Squares Regression

In the case of a least-squares procedure, when fitting the line through a set of m data points ($\{(\bar{x}_1, y_1), (\bar{x}_2, y_2), ...(\bar{x}_m, y_m)\}$) the goal is to minimise the differences between perceived and actual values. In **Ordinary Least Squares**, this is done by minimising the sum of squared errors (the vertical deviations from the fitted line) [16], which is:

$$SSE = \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{m} (y_i - \beta_0 - \beta_1 x_{i,1} - ... - \beta_n x_{i,n})^2.$$

A minimum is attained when the derivatives in respect to each β_k is set to zero, which will give us the **normal equations** [13]:

$$m\beta_0 + m\beta_1 \sum_{i=1}^{m} x_{i,1} + ... + \beta_n \sum_{[i=1]}^{m} x_{i,n} = \sum_{i=1}^{m} y_i,$$

$$\beta_0 \sum_{i=1}^{m} x_{i,k} + \beta_1 \sum_{i=1}^{m} x_{i,1} x_{i,k} + ... + \beta_n \sum_{i=1}^{m} x_{i,k} x_{i,p-1} = \sum_{i=1}^{m} y_i x_{i,k}, k \in \overline{1, n}.$$

Since this is a rather complex problem, we will use linear algebra to approach it. We let $\mathbf{X}$ be the $m \times (n+1)$ matrix

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{m,1} & x_{m,2} & \cdots & x_{m,n} \end{bmatrix},$$

and similarly we define $\mathbf{Y}$, $\hat{\mathbf{Y}}$ using the actual, and respectively predicted values. This means that given the parameters $\hat{\beta}$, we compute the predicted values using the formula: $\hat{\mathbf{Y}} = \mathbf{X}\hat{\beta}$. Therefore, we can now write the normal equations in matrix form [13]:

$$\mathbf{X}^T\mathbf{X}\hat{\beta} = \mathbf{X}^T\mathbf{Y},$$

with solution if $\mathbf{X}^T\mathbf{X}$ invertible:

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}.$$

Coincidently, this gives us the **Best Linear Unbiased Estimator** (BLUE) of $\hat{\beta}$, according to the Gauss-Markov theorem, should the error terms for y_i (ϵ_i) be uncorrelated ($cov(\epsilon_i, \epsilon_j) = 0$ with mean 0 ($E[\epsilon_i] = 0$) and have the same finite variance ($V(\epsilon_i) = \sigma^2 < \infty$) [5].

3.2.2 Ridge Regression

Since Ordinary Least Squares chooses the best parameter values ($\bar{\beta}$) for modelling the training data, it will overfit when given noisy data and perform badly when predicting the test data. Additionally, we have no guarantee that the matrix we invert ($\mathbf{X}^T\mathbf{X}$) is well-conditioned, which causes greater errors when computing the parameters.

This is why, in the case of **Ridge Regression** [11], we try instead to minimize:

$$\sum_{i=1}^{m}(y_i - (\bar{\beta}^T x_i)) + m\lambda\bar{\beta}^T\bar{\beta} = SSE + m\lambda\bar{\beta}^T\bar{\beta},$$

for a fixed $\lambda \geq 0$. The corresponding solution will be obtained from the equation:

$$\beta = (\lambda I_{n+1} + \mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}.$$

The purpose of Ridge Regression (also known as Penalized Least Squares) is to perform regularization on the model, to make it simpler by punishing complexities, which we measure as the 2-norm ($||\bar{\beta}|| = \sum_{i=0}^{n}(\beta_i)^2$) multiplied by a constant, λ, the complexity penalty. Choosing the penalty is therefore important, since higher values will make the parameters not model the training data properly, while lower ones would make it indistinguishable from Ordinary Least Squares. One simple, quite practical method would be to try random values for λ and just choose the one that gives statistically better results. Because of time constraints and the fact that we are dealing with unsigned integer features (meaning very large $\mathbf{X}$ matrices), we have used the heuristic:

$$\lambda = 10^{-6} \max_{x_{i,j} \in \mathbf{X}^T\mathbf{X}} (x_{i,j}),$$

which, in practice, has introduced an appropriate ridge.

In general, this method works reasonably well for almost matrix, but it has one main drawback: when the matrix to be inverted is already non-singular it does manage to introduce an error. This is acceptable especially when we do not have enough training data (why OLSR would fail), although if it is not the case we should reconsider the approach.

The code listing below is a personal implementation of Ridge Regression as a Python Script:

```
import numpy
...
ilambda = pow(10,6)
...
def trainRidgeRegressor(experiment):
  mx=numpy.matrix(numpy.load('.../Training/xmatrix.npy'),
       numpy.float64)
  my=numpy.matrix(numpy.load('.../Training/ymatrix.npy'),
       numpy.float64)

  mxt=mx.getT()
  mxtx=mxt*mx
  mxty=mxt*my

  lambda=mxtx.max()/ilambda
  mr=mxtx+lambda*numpy.matrix(
         numpy.identity(mxtx.shape[0]),numpy.float64)

  betas=numpy.linalg.solve(mr,mxty)
  numpy.save('.../Results/betas_ridge.npy',betas)

det testRidgeRegressor(experiment):
  mx=numpy.matrix(numpy.load('.../Test/xmatrix.npy'),
       numpy.float64)
  my=numpy.matrix(numpy.load('.../Test/ymatrix.npy'),
       numpy.float64)
  betas=numpy.matrix(
          numpy.load('.../Results/betas_ridge.npy'),
          numpy.float64)

  y_=mx*betas
  numpy.save('.../Results/y_ridge',y_)

  ...#make use of y and y_ to compute errors and so on

def runExperiment(experiment):
  ...
  trainRidgeRegressor(experiment)
  testRidgeRegressor(experiment)
  ...
```

3.2.3 NumPy Regression

Although this is not a canonical implementation of regression, we refer to the linear regressor that the **NumPy library** provides as a useful black box to compare it with the previous methods. It computes the minimum-norm solution to least squares with methods from the C implementation of the LAPACK library (Linear Algebra PACKage, originally developed for Fortran), which use the Singular Value Decomposition of the matrix that needs inverting (in our case $\mathbf{X}^T\mathbf{X}$). The code handles better in some cases ill-conditioned matrices and usually runs faster than our implementations, yet it can be outperformed by a ridge regressor under certain situations.

3.3 Calibration

Calibration is often used with the purpose of improving the accuracy of many steganalysis methods. It usually involves manipulation of the stego object in an attempt to recover information about the corresponding cover, to build a reference object (which is then used in the steganalysis).

There are two main calibration techniques: cropping and re-compressing, which is commonly used on JPEGs (and possibly similar lossy formats), and using a library of overlapping images. We will focus only on the latter, since it is more adequate for a lossless format and as shown in [17] it makes classification better. We will try to see whether the same statement holds for a regressor.

Before describing how we will make use of the overlapping images, it is very important to remember that the reference objects can themselves be stego, therefore we can not assume that we will have covers to refer to. As such, we can assume that we know for a subset of images the embedded payload size and we will use that to predict for two images the **difference in payload**. To make this feasible, rather than using the features directly, we will use a function applied to two of them. As for these functions, we have:

- the difference function $\kappa(\bar{x}, \bar{y}) = \bar{x} - \bar{y}$, the motivation being that this way we can "naively" predict the difference in payload;
- the concatenation function $\kappa(\bar{x}, \bar{y}) = \bar{x}||\bar{y}$, because a more complex vector might capture the relation for linear regression;
- the concatenation with difference function $\kappa(\bar{x}, \bar{y}) = \bar{x}||\bar{y}||(\bar{x} - \bar{y})$, a combination of the two above, which is also the most effective of the three when used for classification [17] against both LSBM and HUGO.

It is important to remark that the hypothesis space corresponding for each calibration method is a superset for the previous one, so, in theory, it should be possible to obtain the best model from the most complex one (and have no need for the others). Sadly, this does not work in practice, since we would need infinite training data (and we have only a finite amount). Additionally, the simpler hypothesis have a significant advantage, that of lower dimensionality, meaning regression will be easier and faster.

As for why calibration will make the regression more effective, we will refer to our hypothesis. A vector of features ($\bar{x}$) can be described by the sum of camera noise ($\bar{c}$), space noise ($\bar{s}$), time noise ($\bar{t}$) and steganographic signal (or payload, $\bar{p}$), giving us the formula:

$$\bar{x} = \bar{c} + \bar{s} + \bar{t} + \bar{p}.$$

We assume that each component can be represented by independent random variables (allowing us to later use the difference of variances $Var(A - B) = Var(A) + Var(B)$) and we remark that since the same camera is always used, then $\bar{c}$ should be a constant with variance $Var(\bar{c}) = 0$. If we consider 2 vectors ($\bar{x}_1$ and $\bar{x}_2$) of features belonging to overlapping images, then we have $\bar{s}_1 = \bar{s}_2$, therefore the following is true when they are calibrated with a difference function:

$$\kappa(\bar{x}_1, \bar{x}_2) = \bar{c} + \bar{s}_1 + \bar{t}_1 + \bar{p}_1 - \bar{c} - \bar{s}_2 - \bar{t}_2 - \bar{p}_2 = (\bar{t}_1 - \bar{t}_2) + (\bar{p}_1 - \bar{p}_2),$$

meaning that the camera and space noise have been cancelled out. The efficiency of our regressions is also reflected in the variance of predicted payload, since $Var(y - \hat{y}) = \frac{SSE}{m-n}$ and the Sum of Squared Errors is what we optimize. So we should have better predictions when the variance is lower. When our regressor builds the model without calibration, the variance of predicted payload would therefore be:

$$\begin{aligned} Var(\hat{y}) = Var(\hat{\beta}\bar{x}) &= \hat{\beta}^T(Cov(\bar{s}) + Cov(\bar{t}) + Cov(\bar{p}))\hat{\beta} \\ &= \hat{\beta}^T Cov(\bar{s})\hat{\beta} + \hat{\beta}^T Cov(\bar{t})\hat{\beta} + \hat{\beta}^T Cov(\bar{p})\hat{\beta}, \end{aligned}$$

while if we used calibration, we would get:

$$Var(\hat{y}_1 - \hat{y}_2) = Var(\hat{\beta}(\bar{x}_1 - \bar{x}_2)) = 2\hat{\beta}^T Cov(\bar{t})\hat{\beta} + 2\hat{\beta}^T Cov(\bar{p}))\hat{\beta}.$$

Under our hypothesis, this means that the regression will be better in practice if and only if the variance caused by time noise and steganographic content is smaller than the space noise:

$$\hat{\beta}^T Cov(\bar{s})\hat{\beta} > \hat{\beta}^T Cov(\bar{t})\hat{\beta} + \hat{\beta}^T Cov(\bar{p})\hat{\beta}.$$

In summary, when we change from a regression of absolute payload to one of difference and when given overlapping images, we double the time and stego noise, while cancelling the space and camera noise. Therefore, this method will work if the doubled components are smaller than the ones that get cancelled.

3.4 Training and Testing Sets

For a easier explanation of the defined experiments, we consider the following notations for representing sets of data:

- $c^I_{S,L}$ for cover objects with indices in I, from slices in L of scenes in S;
- $s^I_{S,L}$ for stego objects with indices in I, from slices in L of scenes in S;

for arbitrary subsets $S \subseteq \mathcal{S}$, $L \subseteq \mathcal{L}$, $I \subseteq \mathcal{I}$ where $\mathcal{S}$ is the set of all scenes, $\mathcal{L}$ is the set of all slices and $\mathcal{I}$ is the set of all indices of images.

The different classes of experiments we are going to run are:

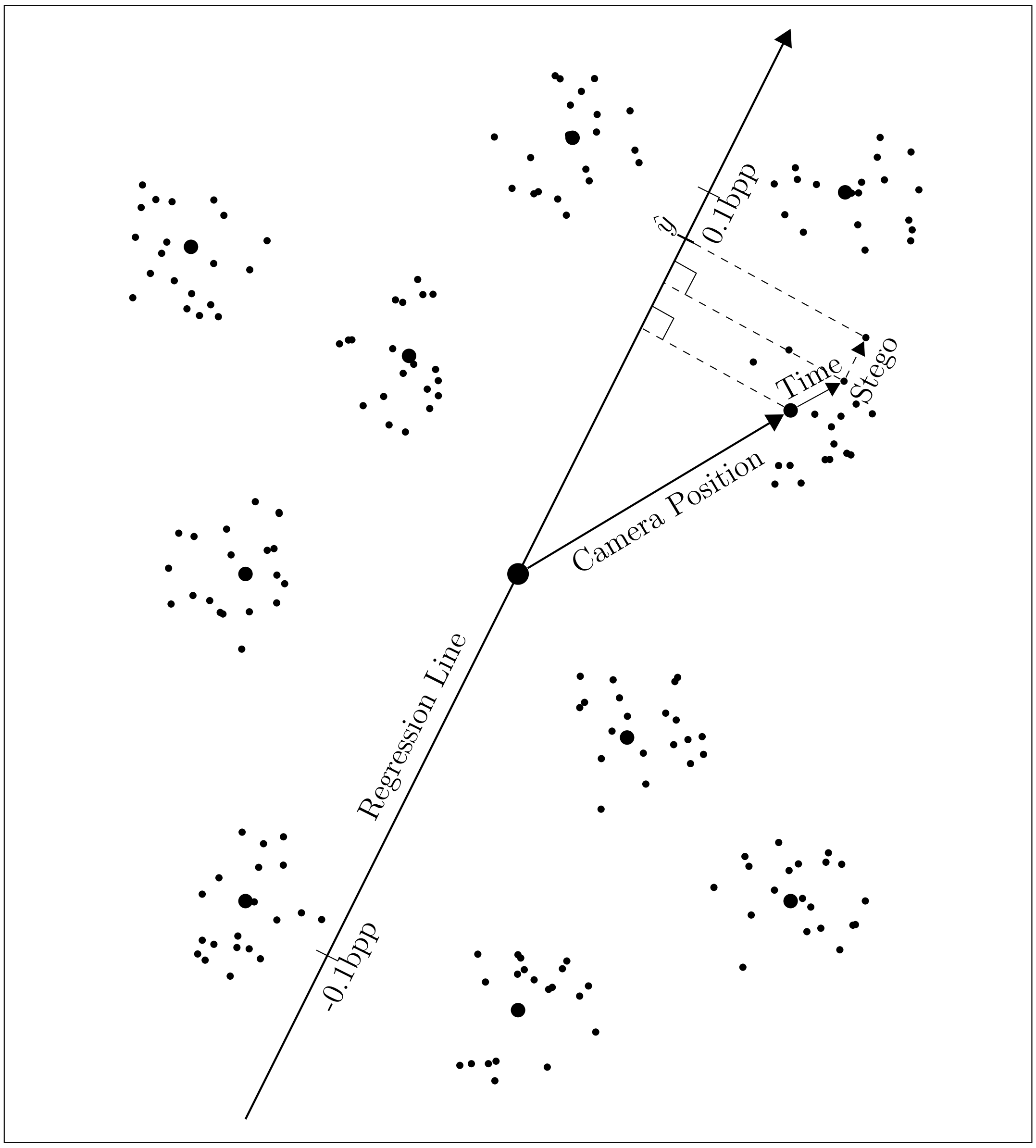


Figure 3.1: Decomposition of features into the sum of noises generated by the Camera Position ($\hat{c} + \hat{s}$), Time ($\hat{t}$) and Steganographic content ($\hat{s}$) and their respective projections on the regression line

- Single Slice Experiments, with Training Dataset $s_{s,l}^{I}$ and Test Dataset $s_{s,l}^{\bar{I}}$, where $I \subset \mathcal{I}$ chosen randomly 10 times for every $s \in \mathcal{S}$, $l \in \mathcal{L}$ such that $|I| = |\mathcal{I}|/2$ (250 training/test features);

- Single Scene Experiments, with Training Dataset $s_{s,L}^{\mathcal{I}}$ and Test Dataset $s_{s,\bar{L}}^{\mathcal{I}}$ where $L \subset \mathcal{L}$ chosen randomly 10 times for every $s \in \mathcal{S}$ such that $|L| = 5$ (2500 training/test features);

- Multiple Scenes Experiments, with Training Dataset $s_{s,\mathcal{L}}^{\mathcal{I}}$ and Test Dataset $s_{S,\mathcal{L}}^{\mathcal{I}}$ where $S \subset (\mathcal{S} - \{s\})$ chosen randomly for every $s \in \mathcal{S}$ such that $|S| = 5$ (25000 training / 5000 test features).

Having these different classes of experiments will allow us to see the performance of regression in different situations, making it easier to draw conclusions afterwards. Also, for evaluating the different experiments, we consider 3 measures (defined for $\epsilon = \hat{y} - y$), namely:

- $E[\epsilon]$, which represents the bias of the predictor (a high value should never arise here);

- $E[|\epsilon|]$, which is the expected average error;

- $\sqrt{E[\epsilon^2]}$, which is the square root of the variance (also representing the Mean Square Error).

We consider the model built by regression to be better, when all these 3 values are closer to 0.

3.5 Implementation

All of the code needed to implement the regression algorithms and run the experiments can be written using scripts. This is why, I have chosen Python as the main programming language, since it is a high-level, dynamic, scripting language with a large number of libraries developed for almost all purposes, which is exactly what was wanted. Matlab would have been a viable alternative, because most of the time, matrix operations had to be performed. Instead we used the NumPy library, a package for scientific computing that provides a multidimensional array object of homogeneous data types and powerful methods for manipulating them. It maintains Python's simplicity, while at the same time offering a better performance by executing optimized pre-compiled C code. This way the main problem of Matlab, namely the slow execution of code, can be avoided while maintaining the required capabilities.

While we can improve the performance by using any faster C-like language, it would not have been as simple to write and it would have required more work for implementing the needed methods.

Before being able to run experiments, the data needed to be processed and converted into the right formats, which was been done using scripts that either called the required commands (for TIFF and PGM conversion and for the HUGO embedding) or implemented them (the case of LSBM embedding and storing the features in NumPy array format). For TIFF conversion, we used RawTherapee, a raw image processing program. It is an obvious choice, because it is capable to understand the CR2 proprietary format, has command line support (meaning that it can be easily used in our scripts), and performs the processing in a much more timely manner than the provided Canon software (0.1s instead of 10s), avoiding a serious bottleneck. PGM conversion was done with ImageMagick (a software suite that handles all the other formats) for very similar reasons. Both were needed because we could not implement a direct conversion, since neither program could handle both formats (RawTherapee offers no PGM support, while ImageMagick is able to partially read CR2 files, but would not perform demosaicing, causing bad conversions). The LSBM embedding was easy to implement from scratch, although this is not the case with HUGO, where using the official implementation as a black box was needed, because of practical concerns.

For a better expression of the data, plotting it was necessary and, to do so, we used the PyPlot module from the Matplotlib library [7]. This made the task considerably easier, given the fact that it can handle the large number of points we needed to plot (which are in the order of thousands, and just as many plots). The library also has a good enough integration with the NumPy array, displaying it properly without the need for reconstructing it.

The experiments themselves have been performed on a server with 20 3.1GHz Intel CPU cores and appropriate memory (96 GB RAM) and storage facilities. In total, approximately 1.5 years of CPU time and 7.9 TB of storage have been necessary for organising and executing them. More details regarding time and storage requirements are in the table.

Process	Unit	Time/Unit	Storage/Unit	Core Time	Total Storage
Photography	Image	1.5s	14.1MB	21.45h	818GB
TIFF Conversion	Image	0.076s	32.9MB	1.086h	1.5TB
PGM Conversion and Slicing	Image	12s	11.75MB	7.152d	1.2TB
LSBM Embedding	Slice	0.5s	1204KB	2.98d	1.2TB
HUGO Embedding	Slice	5s	1204KB	28d	1.2TB
Feature Extraction	Scene	-	180MB	-	53.581GB
Running Regression	-	-	-	1.31y	1.9TB

Table 3.1: Time and Storage Usage

Chapter 4

Results

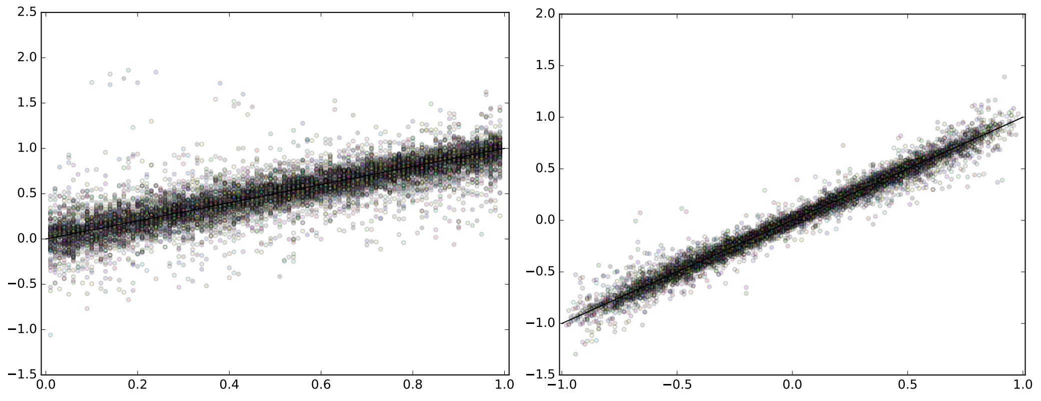


Figure 4.1: Scatter Plots of Actual Values (x-axis) and Predicted Values (y-axis) pairs from 10000 randomly selected samples from Multiple Scene Experiments from a Absolute payload size Ridge Regression Model (left) and Difference in payload size Ridge Regression Model (right)

4.1 LSBM

4.1.1 Confirmation of Hypothesis

We observe that the results of regression applied to our simulation of LSBM embedding (illustrated in the tables 4.1, 4.2, and 4.3) do confirm our hypothesis, since calibration does overall lead to reduced errors and improved estimates.

The experiment class does have an effect on the error measures we obtain: in general, the best predictions are done in Single Slice Experiments, while the highest errors happen in the Single Scene ones, with the Multiple Scene Class being somewhere in between. This situation is to be expected given the size of the training data available for each of them, which influences the quality of models we build.

We see that it also changes the efficiency of calibration, which works well for Single and Multiple Scenes, but not that great for Single Slices (compare the columns in 4.1). This is due to calibration halving the number of training samples that we model and possibly increasing the dimensionality of features. This introduces more inaccuracies when the dimension of features (4680) is already higher than the dataset size (500, respectively 250, without and with calibration for Single Slices), which will make the already under-determined equation even more difficult the solve.

On the other hand, when given enough data, the benefits of calibration will accommodate for the loss in training samples, which happens for both Single and Multiple Scene Experiments. We can see this by comparing the columns in 4.2 and 4.3: the average error and variance are lower when calibration is used compared to when we avoid it. This is because the larger data sets, even when reduced, still allow the matrices which need inverting to be somehow well-conditioned.

There are greater effects on prediction caused by the regression algorithm, which is why we are going to explore their results in more detail.

Ordinary Least Square Regression behaves almost as expected, having higher average errors, variances and even bias. In the case of Single Scene Experiments, the bias ($E[\epsilon] = 1.119$) is higher than the maximum payload size ($1bpp$), meaning that the model it built is of no practical use. When we compare it against the other algorithms, its errors a few orders of magnitude greater and this also holds for Single Slices, indifferent to whether or not calibration was used.

This is because the method is very vulnerable to outliers, which can cause significant changes in the regression line. We are also dealing with many cases when the matrix to be inverted is almost singular, due to low distances between features of overlapping images in their own space, making their measurement (which is what OLSR does) pointless. As such, when the models built by OLSR are then used to make predictions even for well-behaved data, the errors accumulate. There is only one exception, that of Multiple Scene Experiments, when it behaves as well as the NumPy regressor. We attribute this to the many available training samples, which makes modelling easier and allows calibration to work properly.

Compared to OLSR, we obtain better results when using either Ridge or NumPy Regression. They also show similar performance regarding the calibration method used: the lowest errors are always for the difference function, followed by concatenation with difference, and then plain concatenation. The results of concatenation can be blamed on the increased dimensionality of features, although it will not explain why using the difference when building the model is better than not (whether or not we combine it with concatenation). It is very likely that the stego signal is amplified by the difference (as we have hypothesised), while the major noise sources are cancelled. Interestingly, NumPy Regression models have lower errors and variances when no calibration or just difference have been used, while Ridge models are better when using the concatenation functions.

We can also explain, why avoiding calibration gives the best results for Single Slice Experiments (and why it works differently in the other classes). In the case of Single Slices, the camera and space noise is always the same, so there is little noise to be cancelled, which is not true for the others (since we do have captures from multiple slices and even different scenes). Because we do not have too much noise to cancel, it will certainly be smaller than the one which we double via calibration. This is what introduces the errors in our model for the Single Slices.

4.1.2 Illustration

In the three tables, we have the biases, expected average error, and variances. Another way to illustrate this data is in the eight figures we have included.

In the first four (in Figures 4.2 and 4.3), within each ten rows we have a scene, each of them being a slice and every point a capture. They show the errors (by the position of dots), average bias of a slice (marked with crosses), and average bias of the scene (shown with a line). This data is given only for 10 randomly selected scenes out of the 103, when Ridge Regression has been performed on the Multiple Scene Experiments.

Without any calibration (Figure 4.2), we can see that the bias varies within the scene and even within the slices in their respective scene. This illustrates that every slice capture different camera and scene noises, even within the same scene. This is because they represent different tenths of the original picture (which is why the camera noise can also vary, since it may not be uniformly introduced by the sensor) and our features manage to isolate their respective textures.

The other three diagrams (difference function in Figure 4.2, and concatenation without and with difference in Figure 4.3) also illustrate why our hypothesis is true. The bias gets cancelled along with the camera and space noise when calibration is applied to our features, which we believe to be much greater than the time noise which we double). It also shows the lower errors of our predictions, since they are now much closer to the zero-error line.

The remaining four diagrams (in Figures 4.4, 4.5) show the variance (not its square root) of our predictions for each slice (marked with a cross) within their respective scene (the y-coordinate) and the average variance, plotted on a logarithmic scale. These have been drawn for the same experiment class and regressor. We see that no matter what calibration we have used, the variance gets reduced by up to an order of magnitude (illustrated through the left movement of the crosses in the other figures compared to the first one).

Regressor		x	$x-y$	$x\|\|y$	$x\|\|y\|\|(x-y)$
OLS	$E[\epsilon]$	-3.983×10^{-2}	1.129×10^{-1}	8.851×10^{-2}	-2.374×10^{-1}
	$E[\|\epsilon\|]$	2.016	37.191	1.55	18.897
	$\sqrt{E[\epsilon^2]}$	34.261	754.108	41.739	233.736
Ridge	$E[\epsilon]$	1.072×10^{-6}	1.615×10^{-5}	1.296×10^{-5}	-1.220×10^{-6}
	$E[\|\epsilon\|]$	6.647×10^{-3}	9.425×10^{-3}	1.143×10^{-2}	1.054×10^{-2}
	$\sqrt{E[\epsilon^2]}$	1.703×10^{-2}	2.304×10^{-2}	3.161×10^{-2}	2.966×10^{-2}
NumPy	$E[\epsilon]$	-1.629×10^{-7}	1.602×10^{-5}	6.180×10^{-6}	2.533×10^{-6}
	$E[\|\epsilon\|]$	6.108×10^{-3}	9.432×10^{-3}	1.097×10^{-2}	1.038×10^{-2}
	$\sqrt{E[\epsilon^2]}$	1.701×10^{-2}	2.304×10^{-2}	3.120×10^{-2}	2.935×10^{-2}

Table 4.1: LSBM Single Slice Experiments

Regressor		x	$x-y$	$x\|\|y$	$x\|\|y\|\|(x-y)$
OLS	$E[\epsilon]$	1.119	-4.615×10^{-1}	2.399	4.478×10^{-1}
	$E[\|\epsilon\|]$	4.994	50.454	15.053	28.273
	$\sqrt{E[\epsilon^2]}$	27.43	760.524	240.826	254.113
Ridge	$E[\epsilon]$	1.47×10^{-2}	6.379×10^{-5}	2.522×10^{-4}	6.465×10^{-4}
	$E[\|\epsilon\|]$	1.371×10^{-1}	4.113×10^{-2}	5.921×10^{-2}	5.276×10^{-2}
	$\sqrt{E[\epsilon^2]}$	2.288×10^{-1}	7.967×10^{-2}	1.131×10^{-1}	9.807×10^{-2}
NumPy	$E[\epsilon]$	1.815×10^{-2}	6.160×10^{-5}	7.859×10^{-4}	9.495×10^{-4}
	$E[\|\epsilon\|]$	1.091×10^{-1}	4.111×10^{-2}	6.3625×10^{-2}	5.624×10^{-2}
	$\sqrt{E[\epsilon^2]}$	1.783×10^{-1}	7.936×10^{-2}	1.185×10^{-1}	1.048×10^{-1}

Table 4.2: LSBM Single Scene Experiments

Regressor		x	$x-y$	$x\|\|y$	$x\|\|y\|\|(x-y)$
OLS	$E[\epsilon]$	-7.524×10^{-3}	-6.745×10^{-5}	-5.205×10^{-3}	3.939×10^{-3}
	$E[\|\epsilon\|]$	1.009×10^{-1}	3.652×10^{-2}	6.951×10^{-2}	7.272×10^{-2}
	$\sqrt{E[\epsilon^2]}$	1.484×10^{-1}	5.508×10^{-2}	1.025×10^{-1}	1.060×10^{-1}
Ridge	$E[\epsilon]$	-5.132×10^{-3}	-1.567×10^{-5}	3.750×10^{-4}	3.396×10^{-4}
	$E[\|\epsilon\|]$	1.188×10^{-1}	4.008×10^{-2}	5.189×10^{-2}	3.860×10^{-2}
	$\sqrt{E[\epsilon^2]}$	1.752×10^{-1}	6.318×10^{-2}	8.003×10^{-2}	7.468×10^{-2}
NumPy	$E[\epsilon]$	-7.524×10^{-3}	-6.745×10^{-5}	-5.206×10^{-3}	4.382×10^{-3}
	$E[\|\epsilon\|]$	1.009×10^{-1}	3.652×10^{-2}	6.951×10^{-2}	7.520×10^{-2}
	$\sqrt{E[\epsilon^2]}$	1.484×10^{-1}	5.508×10^{-2}	1.025×10^{-1}	1.093×10^{-1}

Table 4.3: LSBM Multiple Scene Experiments

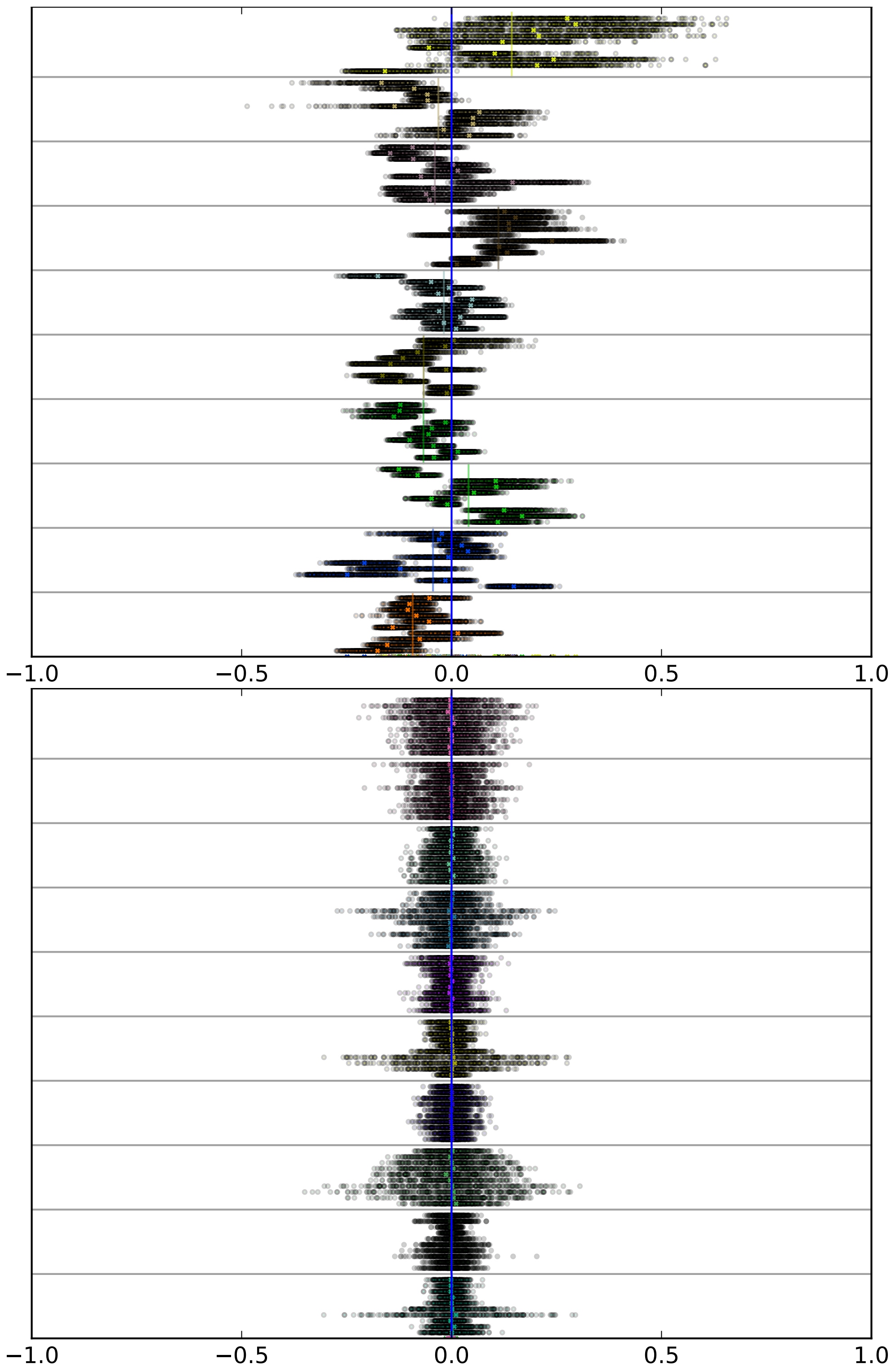


Figure 4.2: LSBM Bias and Error Plot for regression on Multiple Scene Experiments with no calibration (upper) or with difference (lower)

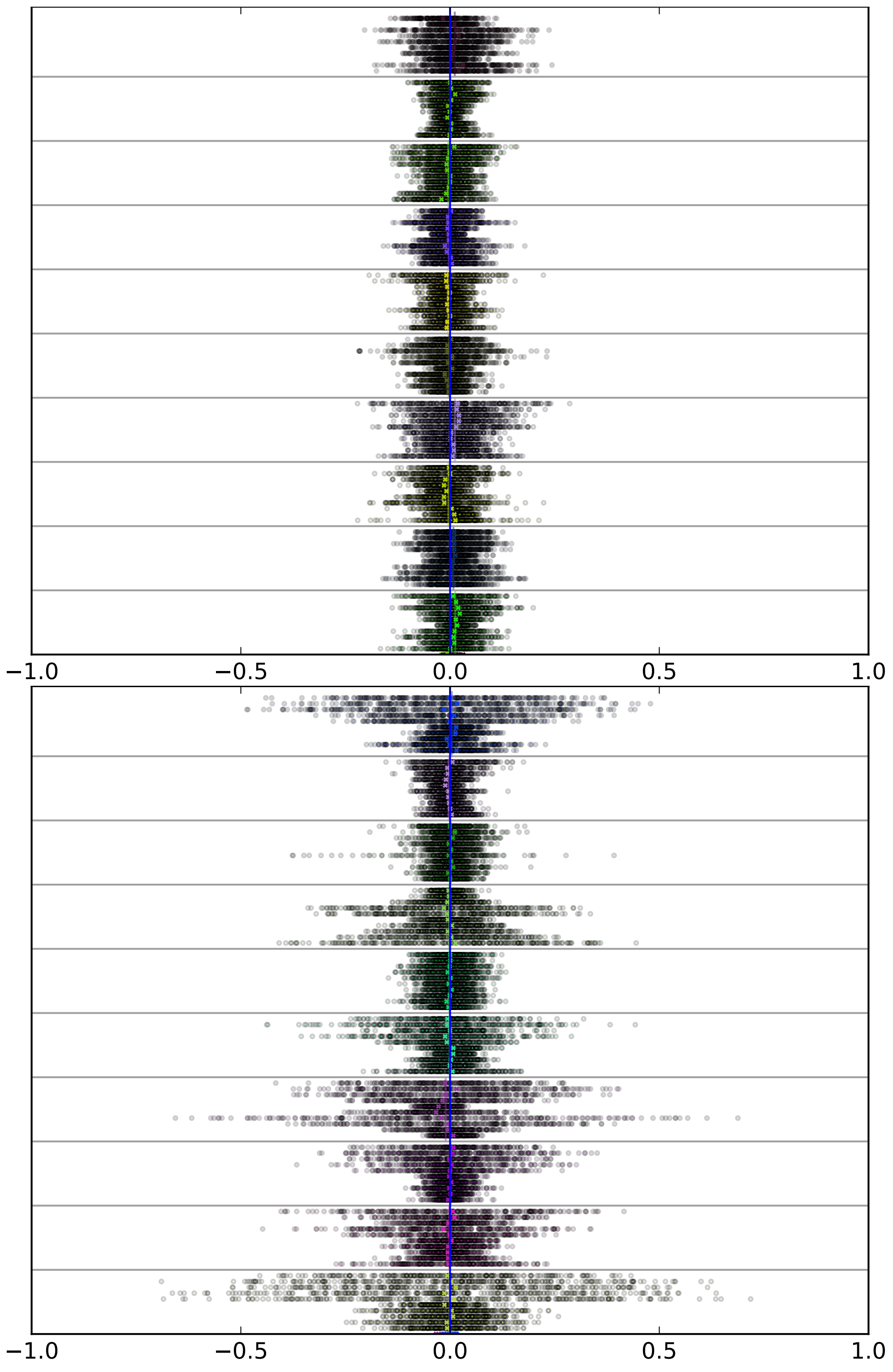
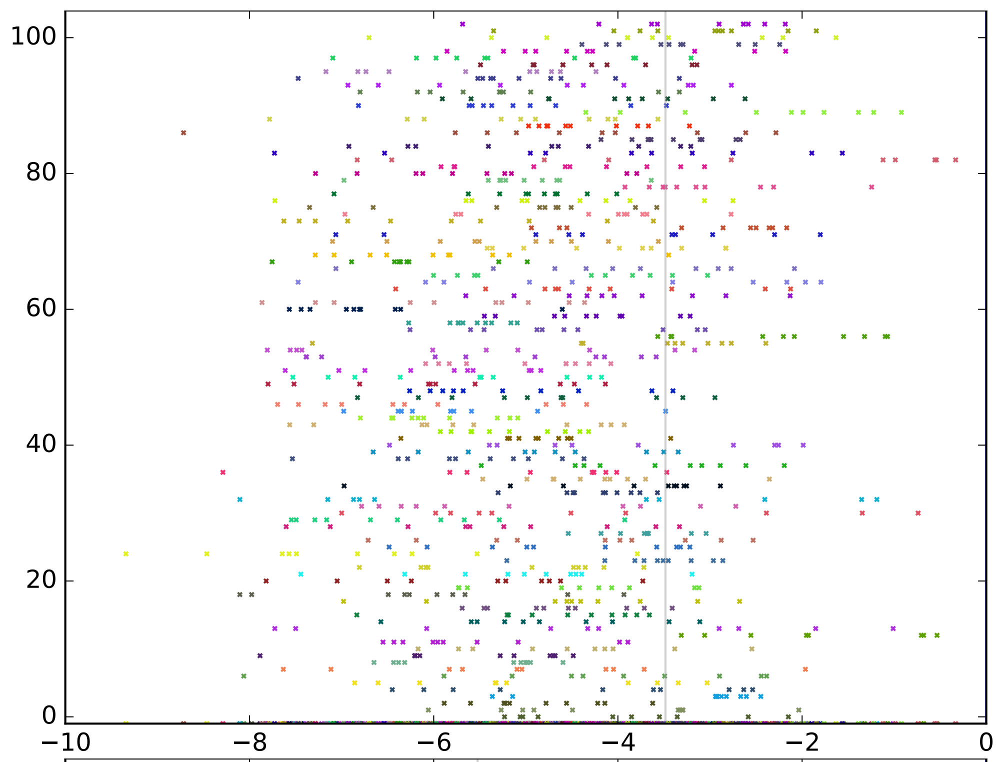


Figure 4.3: LSBM Bias and Error Plot for regression on Multiple Scene Experiments for concatenation without (upper) or with difference (lower)



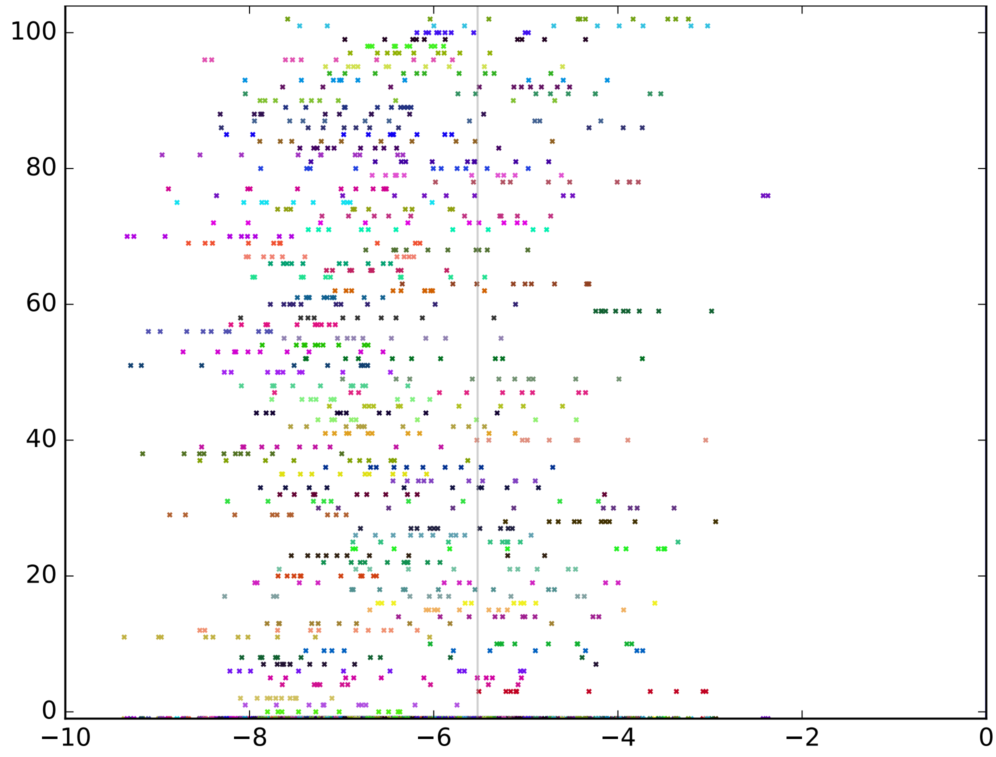
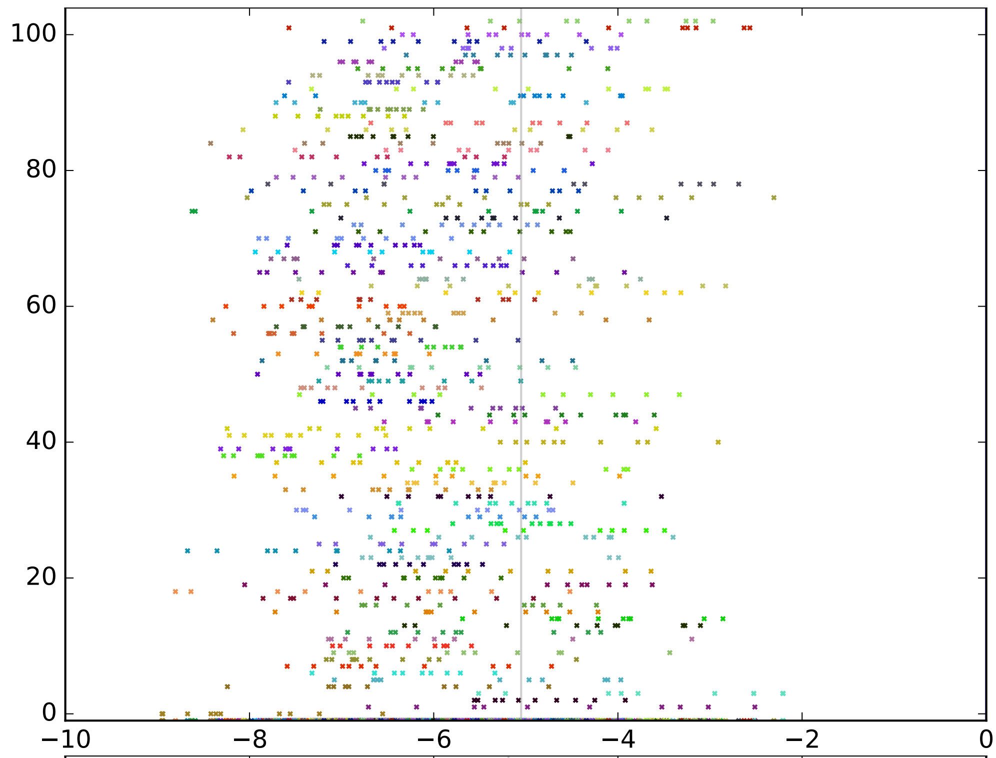


Figure 4.4: LSBM Variance Plot for regression on Multiple Scene Experiments with no calibration (upper) or with difference (lower)



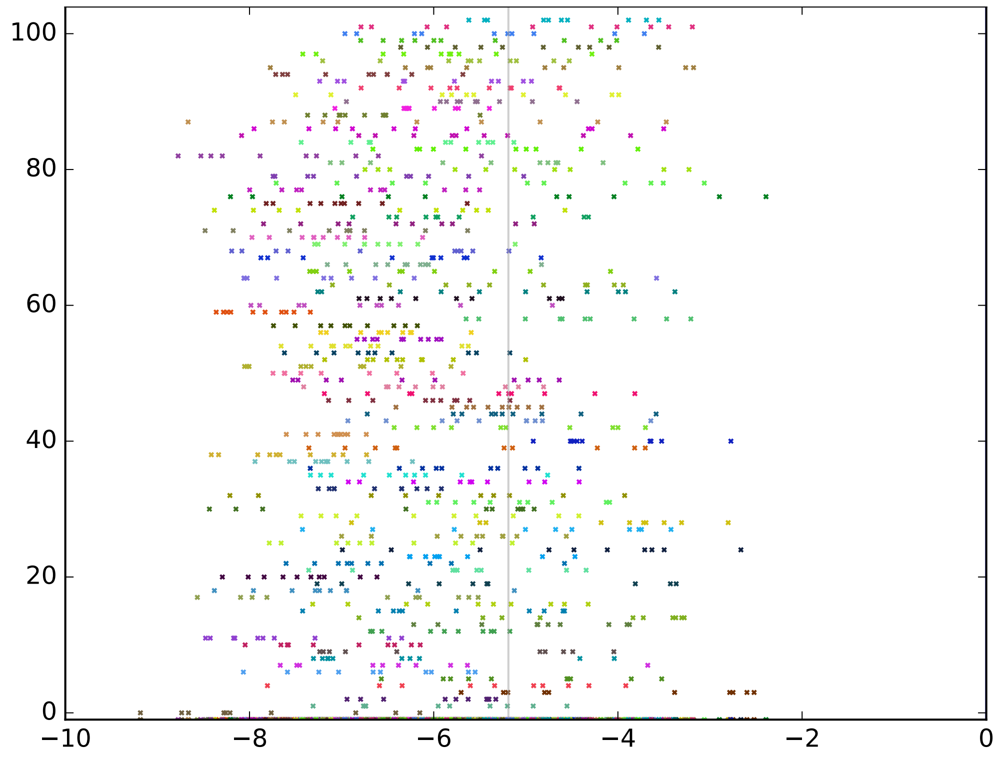


Figure 4.5: LSBM Variance Plot for regression on Multiple Scene Experiments for concatenation without (upper) or with difference (lower)

4.2 HUGO

4.2.1 Confirmation of Hypothesis

If HUGO is a better embedding algorithm than LSBM, it will be harder to estimate the payload sizes, which will translate into higher biases, errors, and variances. We can see that this is actually the case, by observing the tables 4.4, 4.5, and 4.6, which have greater values then the ones for LSBM. This is because it has been designed to minimize the distortion it causes in the images, which will be captured to a smaller degree by our features.

Even so, the results from HUGO experiments show a similar story to the ones obtained from LSBM, with some differences.

OLSR is still on par with the NumPy one for Multiple Scene Experiments (see table 4.6), yet it has incredible overall biases for Single Scene (1.119, when maximum payload size is now 0.4) and errors. When errors are that greater than the range of values, the predictor becomes useless.

Surprisingly, the best performance is almost consistently attained by Ridge Regression, especially when more than one slice is involved in the experiments. This is most likely because the matrices are less well-conditioned compared to LSBM.

Excluding these minor differences, the results behave similar to the ones for LSBM, keeping their tendencies: reduction of bias, errors and variances when calibrating features, except for Single Slice Experiments, and the difference function is still the best calibrator. As such, the same arguments hold and these results again confirm our hypothesis.

4.2.2 Illustration

The three tables contain the biases, average errors and variances that we have discussed for HUGO. This data is also illustrated with the help of eight plots.

The first four of the diagrams (Figures 4.6, 4.7) show prediction errors and average biases within slices and scenes, for 10 randomly selected scenes. This allows us to see when calibration is not used, the errors and biases are higher. We do not observe it as clearly in the table, since we encounter positive and negative biases just as often, and they average to values near 0. We also see that the biases not only vary with the scene, but also with the slice, which get cancelled with calibration.

The other four diagrams (Figures 4.8, 4.9) show the variances (and their average) of the prediction errors resulting from the same experiments and regressor as above. We still see that the variance is higher if no calibration function is applied to the features.

Regressor		x	$x-y$	$x\|\|y$	$x\|\|y\|\|(x-y)$
OLS	$E[\epsilon]$	-9.593×10^{-2}	1.527	4.545×10^{-2}	4.699×10^{-1}
	$E[\|\epsilon\|]$	4.011	19.320	4.085	22.842
	$\sqrt{E[\epsilon^2]}$	130.074	923.811	92.792	960.250
Ridge	$E[\epsilon]$	2.508×10^{-5}	7.727×10^{-5}	7.211×10^{-5}	-9.198×10^{-5}
	$E[\|\epsilon\|]$	2.950×10^{-2}	4.101×10^{-2}	4.877×10^{-2}	4.513×10^{-2}
	$\sqrt{E[\epsilon^2]}$	4.801×10^{-2}	7.242×10^{-2}	9.245×10^{-2}	7.409×10^{-2}
NumPy	$E[\epsilon]$	-1.643×10^{-5}	7.687×10^{-5}	7.133×10^{-5}	-9.178×10^{-5}
	$E[\|\epsilon\|]$	2.640×10^{-2}	4.101×10^{-2}	4.733×10^{-2}	4.460×10^{-2}
	$\sqrt{E[\epsilon^2]}$	4.746×10^{-2}	7.244×10^{-2}	9.406×10^{-2}	7.334×10^{-2}

Table 4.4: HUGO Single Slice Experiments

Regressor		x	$x-y$	$x\|\|y$	$x\|\|y\|\|(x-y)$
OLS	$E[\epsilon]$	2.668	-1.544×10^{-1}	3.080×10^{-2}	-9.014×10^{-1}
	$E[\|\epsilon\|]$	24.311	15.472	37.287	11.771
	$\sqrt{E[\epsilon^2]}$	200.503	138.079	293.421	41.089
Ridge	$E[\epsilon]$	3.907×10^{-2}	-1.859×10^{-4}	3.104×10^{-3}	-8.141×10^{-4}
	$E[\|\epsilon\|]$	3.487×10^{-1}	7.462×10^{-2}	1.250×10^{-1}	1.158×10^{-1}
	$\sqrt{E[\epsilon^2]}$	5.209×10^{-1}	1.466×10^{-1}	2.169×10^{-1}	2.089×10^{-1}
NumPy	$E[\epsilon]$	5.431×10^{-2}	-1.792×10^{-4}	1.612×10^{-2}	-3.589×10^{-3}
	$E[\|\epsilon\|]$	3.467×10^{-1}	7.662×10^{-2}	1.957×10^{-1}	1.793×10^{-1}
	$\sqrt{E[\epsilon^2]}$	5.708×10^{-1}	1.496×10^{-1}	3.430×10^{-1}	3.319×10^{-1}

Table 4.5: HUGO Single Scene Experiments

Regressor		x	$x - y$	$x\|\|y$	$x\|\|y\|\|(x-y)$
OLS	$E[\epsilon]$	-4.159×10^{-3}	4.936×10^{-5}	3.005×10^{-2}	-8.445×10^{-3}
	$E[\|\epsilon\|]$	2.461×10^{-1}	6.885×10^{-2}	2.059×10^{-1}	1.902×10^{-1}
	$\sqrt{E[\epsilon^2]}$	3.490×10^{-1}	9.810×10^{-2}	3.036×10^{-1}	2.708×10^{-1}
Ridge	$E[\epsilon]$	-5.191×10^{-3}	-1.447×10^{-4}	2.179×10^{-3}	-1.174×10^{-3}
	$E[\|\epsilon\|]$	1.591×10^{-1}	6.318×10^{-2}	8.315×10^{-2}	7.714×10^{-2}
	$\sqrt{E[\epsilon^2]}$	2.101×10^{-1}	9.168×10^{-2}	1.193×10^{-1}	1.101×10^{-1}
NumPy	$E[\epsilon]$	-4.159×10^{-3}	4.936×10^{-5}	3.005×10^{-2}	-8.117×10^{-3}
	$E[\|\epsilon\|]$	2.461×10^{-1}	6.885×10^{-2}	2.059×10^{-1}	1.893×10^{-1}
	$\sqrt{E[\epsilon^2]}$	3.491×10^{-1}	9.810×10^{-2}	3.036×10^{-1}	2.665×10^{-1}

Table 4.6: HUGO Multiple Scene Experiments

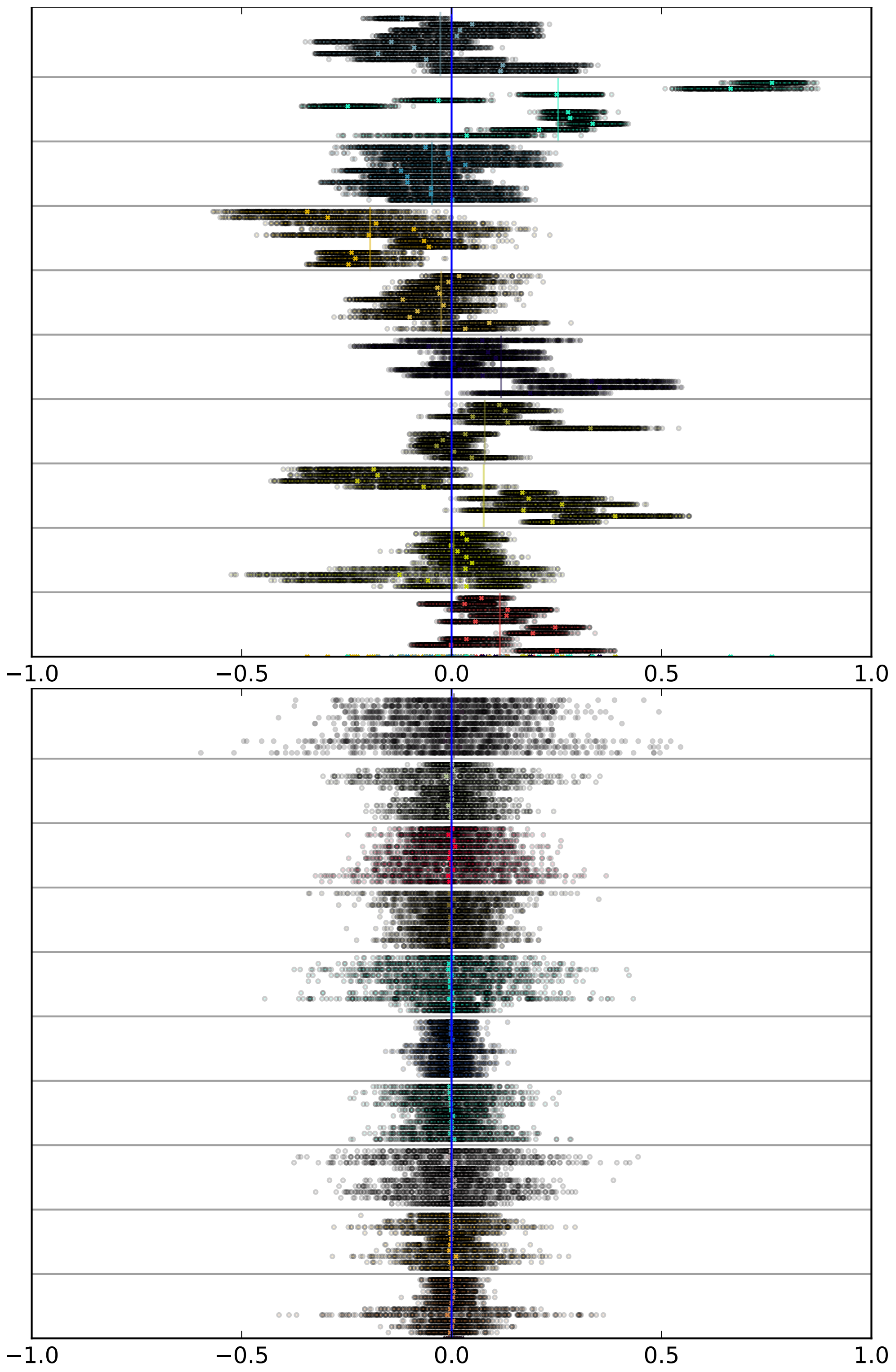
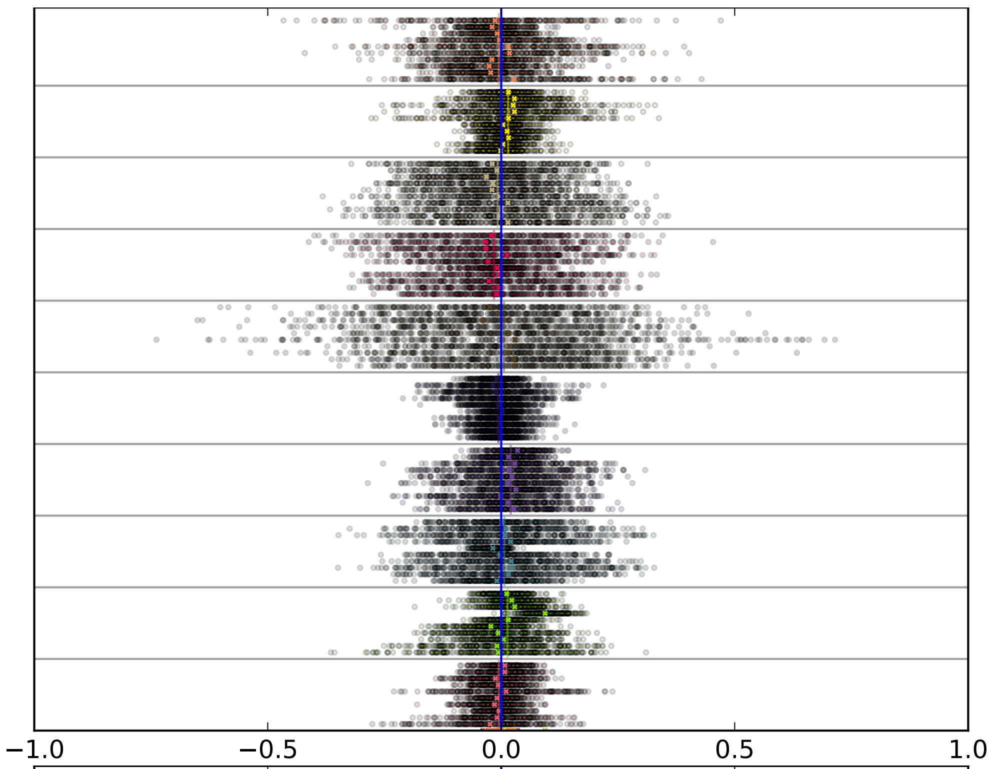


Figure 4.6: HUGO Bias and Error Plot for regression on Multiple Scene Experiments with no calibration (upper) or with difference (lower)



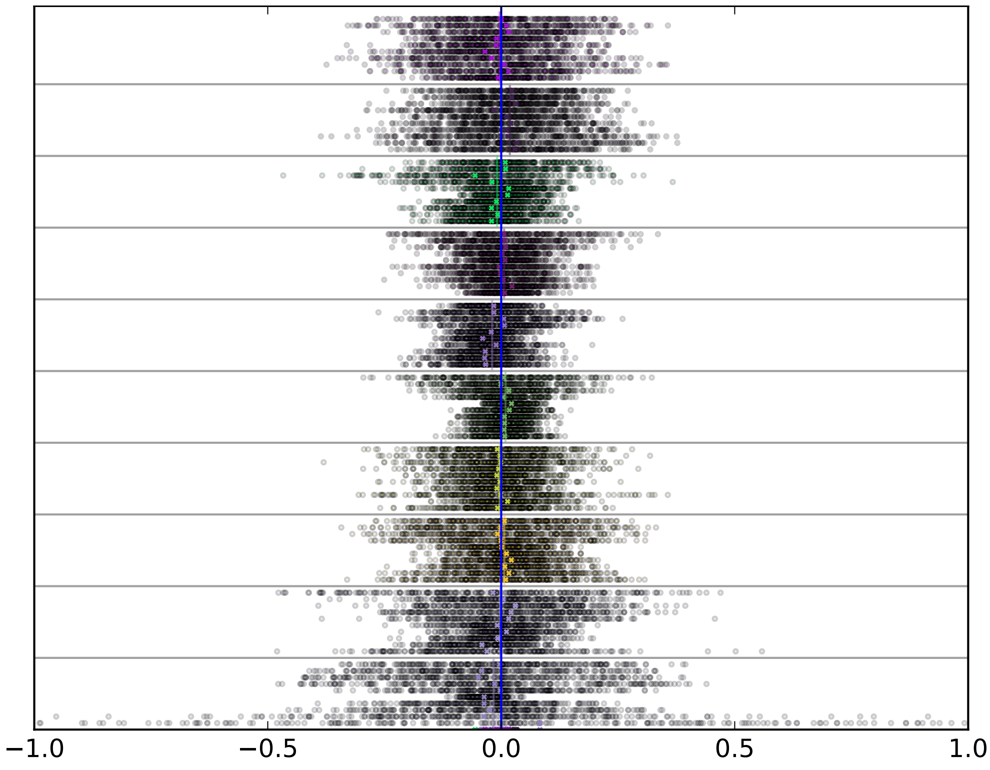


Figure 4.7: HUGO Bias and Error Plot for regression on Multiple Scene Experiments for concatenation without (upper) or with difference (lower)

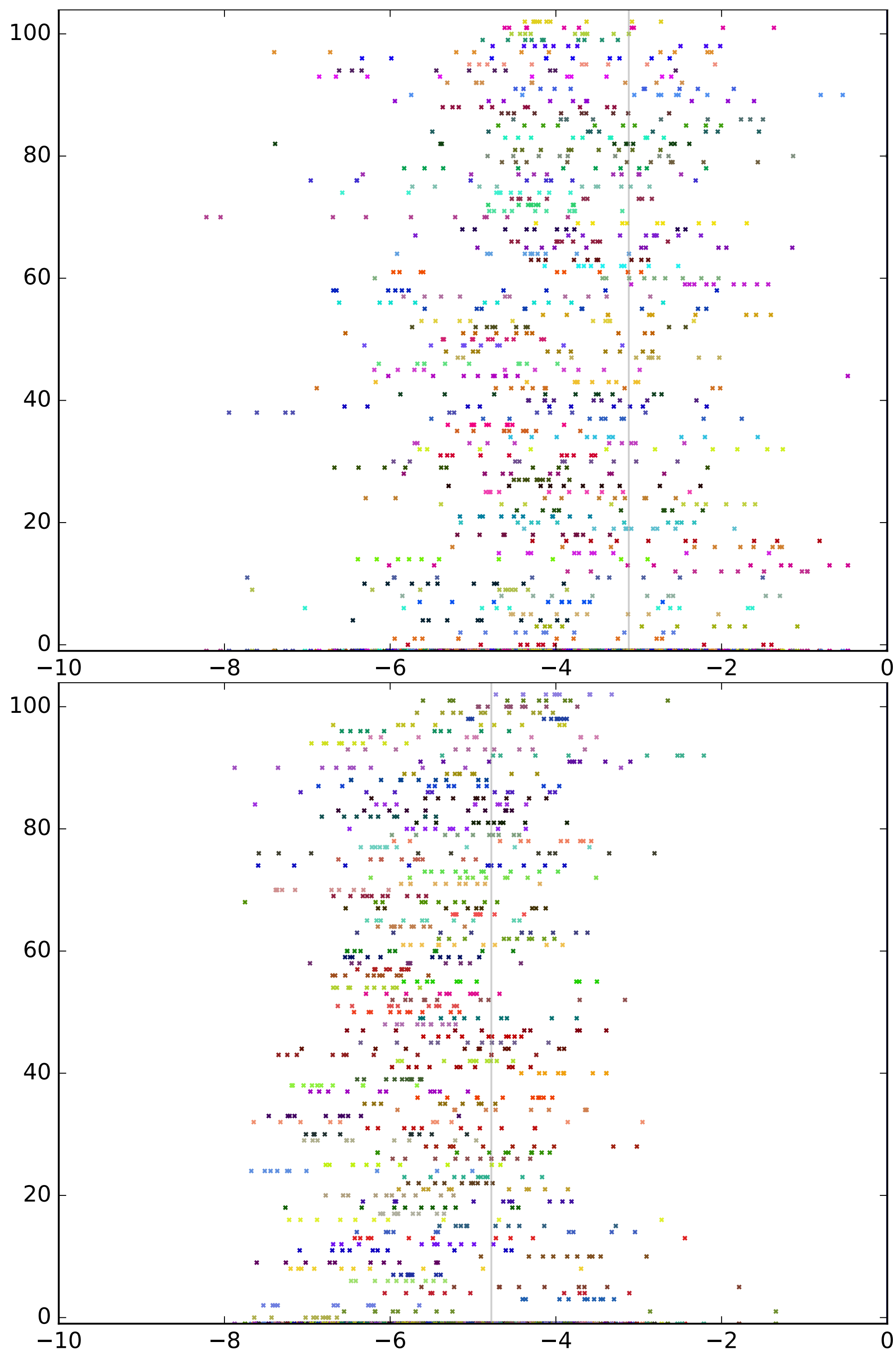
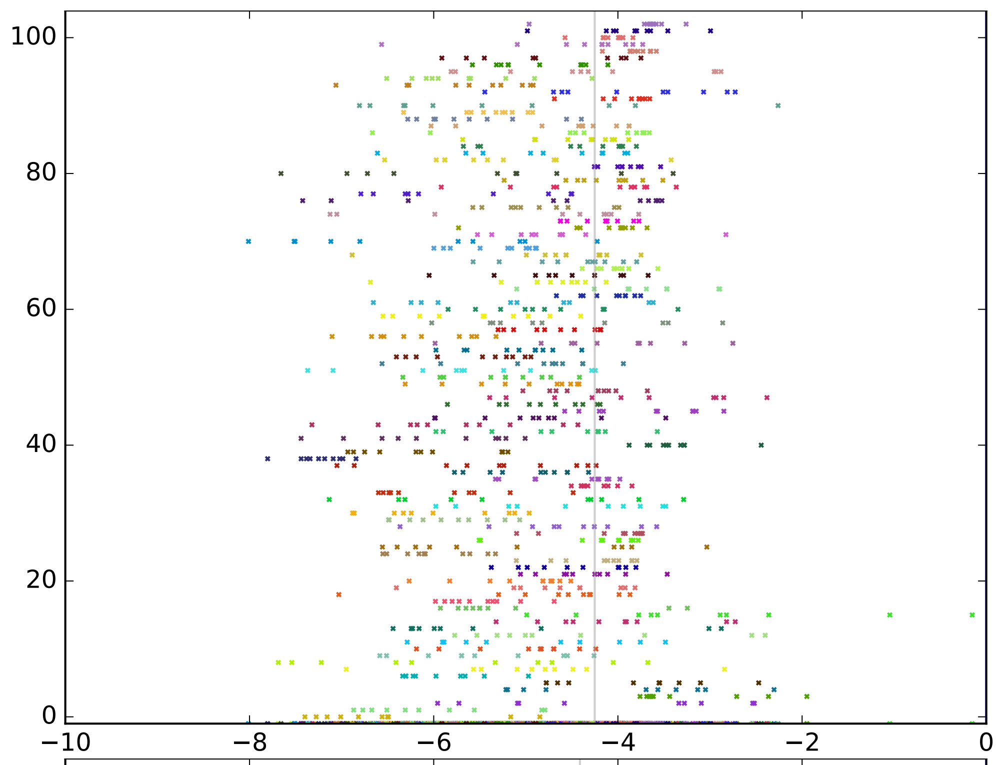


Figure 4.8: HUGO Variance Plot for regression on Multiple Scene Experiments with no calibration (upper) or with difference (lower)



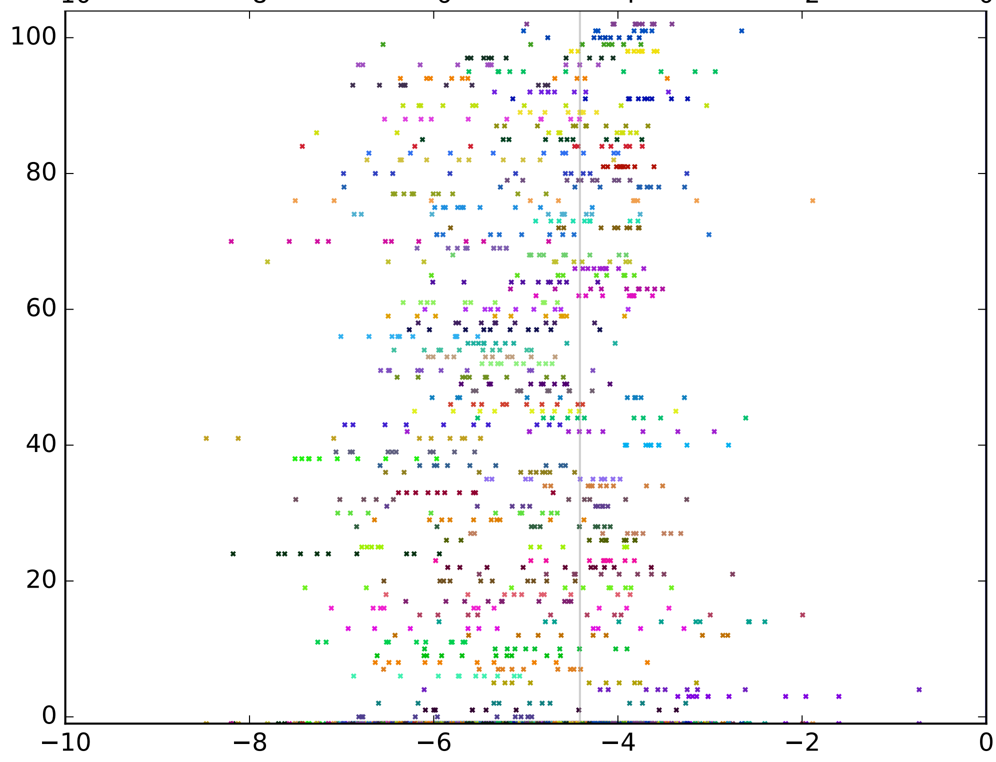


Figure 4.9: HUGO Variance Plot for regression on Multiple Scene Experiments for concatenation without (upper) or with difference (lower)

Chapter 5

Conclusion

We hypothesised that every image has cover content described by multiple noise sources (camera, space and time) and possibly some stego signal and that by making use of this we can perform better regression when calibrating with overlapping images. The results of our experiments confirm this hypothesis, which has interesting ramifications for both steganography and steganalysis at the same time.

Performing regression on calibrated overlapping images allows us to more reliably detect the difference in payload. While this may not appear useful when handling two images with close payload sizes, when it actually observes a difference, it is almost certain that at least one of them is a stego object. This means we have just confirmed the existence of secret communication and we can begin investing resources with the purpose of recovering the message. Previous work has assumed that it is necessary to have some images to be covers and possibly know which these are, our work shows that we do not need to have this knowledge, although with an insignificant drawback. Our method is not reliable when comparing two cover images or two stego with similar payload size, allowing the steganographer to either not embed in overlapping images or to embed in all of them. The first option means that cover selection needs to be done properly, having the steganographer invest more time and resources in choosing his covers, since his options are now much more limited (no overlapping images, nor any for which overlapping ones can be easily taken), making it slightly impractical. The latter option will certainly cause problems later on, since it is especially vulnerable against pooled steganalysis, as shown in [8] (regression for the average payload size).

We have also discussed the differences between 4 noises (camera, scene/space, time and stego), which have not been separated until now, as this does requite a large number of captures in different conditions (and no one else has done this). The results we have obtained do show that they are actually different and, based on that, we can make trade-offs via calibration, by enhancing some of them at the cost of others (difference calibration is the most illustrative, as camera and space noise are suppressed, while time and stego are amplified). Better designs in calibration functions may result as a consequence of this fact, since having some trade-offs might make the measuring of certain statistics easier and more reliable, especially when given images with significant overlapping content.

There are certainly some ways this project could have been improved, which may count as future work in this field. Firstly, having more captures for the Single Scene Experiments, might have made some of the results clearer (regarding the drop in performance). Another would be repeating the experiments for up-to-date embedding algorithms (like the UNIWARD family), which could not be attempted, mostly because of time constraints (while effective, the algorithms are still too slow). There is also the Ridge Regression, for which we have used a simple heuristic. Investigating the effects of different parameters on the efficiency of models would likely lead to better results. Lastly, there is the concern of how to better separate the noise sources, which, for example, may involve repeating the experiment with overlapping images from different cameras.

There are also lessons to be learned from the project. Building a large dataset takes time, running the experiments take even more time, as 1.5 years of CPU time clearly show. Working with this much data can also cause unexpected problems (PDF readers will crash when rendering vector-based plots of several hundred thousands of points) and even small mistakes can be time-consuming, requiring all the experiments to be repeated.

Taking all in consideration, we did obtain an important result: calibrating overlapping will cancel some noise components at the expense of double others. In consequence, using calibrated features to perform regression for the difference in payload (or some other metric) will lead to better models and predictions.

References

[1] Laura Bengescu and Andrew D. Ker. Steganalysis in overlapping JPEG images. 2015.

[2] DDE Lab Binghamton University. Steganographic algorithms. `http://dde.binghamton.edu/download/stego_algorithms/`.

[3] J. Fridrich and J. Kodovsky. Rich models for steganalysis of digital images. *IEEE Transactions on Information Forensics and Security*, 7(3):868–882, June 2012.

[4] Jessica Fridrich, Jan Kodovský, Vojtéch Holub, and Miroslav Goljan. Breaking hugo: The process discovery. In *Proceedings of the 13th International Conference on Information Hiding*, IH'11, pages 85–101, Berlin, Heidelberg, 2011. Springer-Verlag.

[5] Fumio. Hayashi. *Econometrics.* Princeton University Press, Princeton, N.J. ; Oxford, 2000. Includes bibliographical references, and an index.

[6] Vojtch Holub, Jessica Fridrich, and Tom Denemark. Random projections of residuals as an alternative to co-occurrences in steganalysis. *Proc. SPIE*, 8665:86650L–86650L–11, 2013.

[7] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[8] Andrew D. Ker. Batch steganography and pooled steganalysis. In *Proceedings of the 8th International Conference on Information Hiding*, IH'06, pages 265–281, Berlin, Heidelberg, 2007. Springer-Verlag.

[9] Andrew D. Ker. Implementing the projected spatial rich features on a gpu. *Proc. SPIE*, 9028:90280K–90280K–10, 2014.

[10] Andrew D. Ker. Information hiding lecture notes, 2016.

[11] Kevin P. Murphy. *Machine learning : a probabilistic perspective.* Adaptive computation and machine learning series. MIT Press, Cambridge, Mass. ; London, c2012. Includes bibliography (p. [1015]-1045) and index.

[12] Tomáš Pevný, Tomáš Filler, and Patrick Bas. Using high-dimensional image models to perform highly undetectable steganography. In *Proceedings of the 12th International Conference on Information Hiding*, IH'10, pages 161–177, Berlin, Heidelberg, 2010. Springer-Verlag.

[13] John A. Rice. *Mathematical statistics and data analysis.* Duxbury Press, Belmont, Calif, 2nd ed. edition, 1995. Includes bibliographical references and indexes.

[14] Toby Sharp. An implementation of key-based digital signal steganography. In *Proceedings of the 4th International Workshop on Information Hiding*, IHW '01, pages 13–26, London, UK, UK, 2001. Springer-Verlag.

[15] Gustavus J. Simmons. *Advances in Cryptology: Proceedings of Crypto 83.* Springer US, Boston, MA, 1984.

[16] Dennis D. Wackerly, William. Mendenhall, and Richard L. Scheaffer. *Mathematical statistics with applications.* Thomson, Brooks/Cole, Belmont, Calif., 7th ed. edition, 2008. "International student edition"–Cover.

[17] James M. Whitaker and Andrew D. Ker. Steganalysis of overlapping images. *Proc. SPIE*, 9409:94090X–94090X–15, 2015.