Detection of Least Significant Bit Replacement Using Machine Learning

CORBIN GRAHAM, Iowa State University, USA

Least Significant Bit Replacement is a method of Steganography. This method relies on embedding data into the least significant bit plane of a 2-Dimensional image. There are multiple methods of detecting a forged image (one that is using the LSBR method) which could be automated using Machine Learning. This paper will hopefully provide a consistent method of detection for the LSBR Steganographic method.

CCS Concepts: • **Machine Learning** → **Classification**; *Detection*; • **Steganography** → Least Significant Bit Replacement.

Additional Key Words and Phrases: machine learning, least significant bit replacment, LSB, LSBR, Steganography

ACM Reference Format:
Corbin Graham. 2023. Detection of Least Significant Bit Replacement Using Machine Learning. *ACM J. Exp. Algor.* NA, NA, Article MATH 535 Steganography (April 2023), 10 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Since it has been proven that there are several methods of steganalyis that can accurately predict if an imaged has been forged, it can be hypothesized that a Machine Learning model can be trained to classify this forgery.

Using various machine learning classification models and various features, it should be possible to accurately and consistently predict if an image has been forged.

1.1 Problem Being Solved

Steganography has two sides; hiding and detecting. This approach should provide a method for detecting the Least Significant Bit Replacement method of Steganography.

1.2 Background and Related Work

My work is mostly related to machine learning and different machine learning approaches to problem solving. This is my first course in Steganography but I am hoping to combine what I have learned from my machine learning courses and apply it to this course.

1.3 Approach

The goal of this research is to determine if it is possible to accurately and consistently detect Least Significant Bit Replacement using Machine Learning.

Author's address: Corbin Graham, Iowa State University, Ames, Iowa, USA, cgraham1@iastate.edu.

1084-6654/2023/4-ARTMATH 535 Steganography $15.00
https://doi.org/XXXXXXX.XXXXXXX

2 METHOD

To gather the data, we will gather a collection of steganographic images that are generated using a LSBR embedding algorithm[1] at two different rates, 10% and 40%[2]. Generate features through various methods[3] then train and test a machine learning model on the generated features.

Accuracy will be measured from the percentage of accurate predictions during the testing phase of the machine learning process.

2.1 Apparatus and Instrumentation

Python, sci-kit learn.

2.2 Materials

There are two primary materials that will be used in the detection of LSBR embedding. First, embedded images, and their cover images. Second, machine learning algorithms / models to be used for the detection of the embedding.

2.2.1 Images. Using a set of 100 cover images, we will generate 200 steganographic images. The first 100 generated images will have an embedding rate of 10%, or 10% of the LSB plane will be replaced with hidden data. The second 100 generated images will have an embedding rate of 40%, or 40% of the LSB plane will be replaced with hidden data. These images will be embedded using Least Significant Bit Replacement, or replacing the exact bits of the least significant bit plane. This is only one approach to LSB embedding, another would be the two-sum approach which we will be ignoring for this example. The images will then have features extracted and used for training and testing the machine learning model.

2.2.2 Libraries. The libraries I will be using to implement the machine learning algorithms are from the open source scikit-learn Python library.

2.3 Design

2.3.1 Least Significant Bit Embedding. Formatting cover images: to ensure consistency between images, all will be flattened from 3-dimensional color images to 2-dimensional black-and-white images.

Cover images will be embedded with payloads of 0.1 and 0.4 (rates dependent of image size) where 0.1 is 10% of image size and so on.

2.3.2 Feature Engineering. Features will be generated by feeding an image into an algorithm that will return some value that will be added to the DataFrame containing images and their feature values.

2.3.3 Labels. This is a classification problem where the label will be $True$ or $False$ and one-hot encoded as 1 for $True$ and 0 for $False$.

2.4 Procedure

The images will be first fed into the program as any shape or size and then converted to flat 2-dimensional black-and-white images. Then these 2-dimensional images will be embedded at rates of 0.1 and 0.4. These embedded images will be fed into a feature-generator that will couple them in a DataFrame with their features. These Machine Learning models will train on the generated features and tested for their accuracy. Accuracies above 50% will show successful rates of determining embedding.

3 DATA

3.1 Cover Images

Cover images are gathered using any camera. These are all taken on the same camera on the same settings as to not introduce external risk factors. These can be any size or resolution so long as they are unfiltered.

3.2 Steganographic Images

Steganographic images are created by first flattening the cover images from 3-dimensional color images to 2-dimensional black-and-white images. Then, the images are uniformly sized to 1024x1024. Once we have a set of uniform images, we will embed randomly generated message data.

3.2.1 Generating Message Data. The stegonographic image will be generated by embedding (as bits) a random payload of bits of K size and at rate: $rate \in \{0.1 \vee 0.4\}$. Where $capacity$ is the maximum image capacity in the LSB plane.

$$M, N \leftarrow \{Image \rightarrow shape : M = 1024, N = 1024\}$$

$$\texttt{capacity} \leftarrow \lceil log_2(M * N) \rceil$$

$$K \leftarrow \texttt{rate} \times (M \times N - \texttt{capacity})$$

$$payload \leftarrow \{(x, x) : x = \texttt{Random} \in K\}$$

3.2.2 LSB Embedding. The generated payload will be embedded in the LSB plane of the image by sub-sectioning the plane to three sections: The K-bit section which shall be the length of the generated messaage (used for delimiting); The message bits (the message data itself); And the cover bits which shall be all remaining bits that are unaffected. These will not be read since the message is only read to the assigned length provided by K-bit section.

$$\texttt{embed(Image, payload)}$$

$$\texttt{Image}(x_0..x_k) = |\texttt{payload}|$$

$$n = k + 1$$

$$\texttt{Image}(x_n) = \begin{cases} 1 & \text{if } \texttt{payload}(n) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

3.3 Features

Features are used for training and testing the machine learning models.

3.3.1 Image. The image is stored as an array of size (1024×1024). This is flattened where each value is used as a feature. This is ineffective because the actual result should not be noticeable since we do not have the cover comparatively.

3.3.2 LSB Plane. The LSB plane, or Least Significant Bit Plane, is the sequence of each least significant bit. This is also an inadequate feature because it cannot be determined from the plane alone what the embedding may be.

3.3.3 Histogram. A 2-bin histogram is created using the LBS plane to determine the distribution of bits as either 1 or 0. This could potentially yield some results if linearly modeled.

3.3.4 Chi-Squared Test. The expected value, E, for distribution of 1 and 0 is 50%. The observed value, O, is the actual result. This test should yield some results because it alone is often used for determining if an image has been embedded.

$$x^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

3.3.5 Neighborhood Test. It is expected that values within the same neighborhood of an image (measured by distance from center) should have similar values and any outliers should give indication that there has been some forgery on this image. We can measure the neighborhood as:

$$D_s(P, Q) = max(|x - s|, |y - t|)$$

and the mean of each of these values should give us the expected value for each and we can measure the greatest sum of outliers using the Chi-Squred test given previously.

3.4 Machine Learning Models

The problem is a classification problem so we will be using classification models for determining if the image is embedded. Features are generated using multiple methods and labels are True or False to represent an embedded message.

3.4.1 Splitting Data. `train_test_split` from `sklearn.model_selection` was used to split the data between a training and testing set.

Test size was set to 20% and 10% according to common split percentages to not over-fit data.

Random state was set to 1 to ensure the same random selection between all models.

Shuffling was enabled since the data is ordered.

3.4.2 Random Forest Classifier. Random Forest Classifier was set with 100 estimators.

3.4.3 K-Nearest Neighbors (KNN). For this algorithm, I am leaving all measurements to their default including the number of neighbors at 5.

3.4.4 Support Vector Machines (SVM). For this algorithm, I am using the Linear Support Vector Classification algorithm (SVC).

The max number of iterations is limited to 10 and the random state is set to 1 to ensure all cases have the same random probabilities.

4 ANALYSIS

See Table 1 for accuracy data.

4.1 Features

Different features yielded different accuracies for the machine learning model.

4.1.1 Image. A 2-dimensional (1024×1024) image flattened yielded 45% to 52% accuracy meaning this cannot accurately determine if an image is embedded.

4.1.2 LSB Plane. The LSB plane yielded 45% to 52% accuracy meaning this cannot accurately determine if an image is embedded.

4.1.3 Histogram. A histogram (1×2) showing the distribution of LSB planar values yielded 45% to 52% accuracy meaning this cannot accurately determine if an image is embedded.

4.1.4 Chi-Squared Test. A 2-dimensional (1024×1024) image flattened yielded 45% to 52% accuracy meaning this cannot accurately determine if an image is embedded.

4.2 Embedding Rate Inaccuracy

The 40% embedding rate had a lower false-positive rate than the 10% embedding rate.

5 RESULTS

The image, LSB plane, and histogram provided no accurate predictions of LSB embedding. The Chi-Squared test provided a 60% accuracy average during testing meaning it can accurately predict LSB embedding only 60% of the time. This alone is not accurate enough to be considered consistent for determining embedding.

6 DISCUSSION AND LIMITATIONS

LSBR detection is difficult when attempting to approach it naively. When detecting image embedding, knowing the algorithm used for embedding is what allowed for this detection to be possible. Without knowledge of the algorithm used, this machine learning model would be inadequate because it does not contain enough features to accurately determine if an image was embedded using any algorithms other than replacement.

6.1 Limitations

Research was limited by the selection of features and number of images used in testing.

6.1.1 Features. The number of features were limiting because of their accuracy on number of images and total number of features generated per image.

6.1.2 Image Choice. The number of images was limited to 100 under time constraint. More images may yield better testing accuracy.

7 CONCLUSION AND FUTURE

The image set consisted of 100 cover images and their 100 steganographic counterparts. The larger dataset may have yielded greater results for the same features.

The features were limited to two steganographic tests which limited the available determinants for detecting image embedding.

60% is a weak starting point for detecting LSBR embedding but it shows that it is possible to use machine learning classification to detect this method of steganography. Further improving the features and size of the dataset would allow a model to be better trained on detection.

For future testing, using a larger set of images and a wider range of features may allow for the greatest possible results but since it has shown itself to be possible, it is worth further study.

8 DECLARATION OF CONFLICTS OF INTERST

LSBR detection somewhat relies on the understanding of what method was used for steganographic embedding used when hiding data in an image.

Research was conducted with an elementary understanding of LSBR and classification problems. Outcomes may change dependant of understanding and expertise.

9 ACKNOWLEDGEMENTS

This research project was conducted as part of the final project for Math 535. Steganography and Digital Image Forensics at Iowa State University in Spring 2023 for Dr. Jennifer Newman.

10 REFERENCES

Sklearn.neighbors.kneighborsclassifier. scikit. (n.d.). Retrieved May 2, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn-neighbors-kneighborsclassifier

Support Vector Machines. scikit. (n.d.). Retrieved May 2, 2023, from https://scikit-learn.org/stable/modules/svm.html#svc

Sklearn.ensemble.randomforestclassifier. scikit. (n.d.). Retrieved May 2, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier

The Chi Squared Tests: The BMJ. The BMJ | The BMJ: leading general medical journal. Research. Education. (2021, April 12). Retrieved May 2, 2023, from https://www.bmj.com/about-bmj/resources-readers/publications/statistics-square-one/8-chi-squared-tests

11 APENDICES

11.1 Table 1: Model-Feature Accuracy

Model	Features	Accuracy
Random Forest Classifier	Image	0.55
	LSB Plane	0.50
	Histogram	0.50
	Image, LSB Plane, Histogram	0.45
	Chi-Squared Test	0.60
	Neighborhood Test	0.50
	Chi-Squared and Neighborhood Test	0.45
KNN	Image	0.45
	LSB Plane	0.60
	Histogram	0.70
	Image, LSB Plane, Histogram	0.35
	Chi-Squared Test	0.45
	Neighborhood Test	0.50
	Chi-Squared and Neighborhood Test	0.55
SVM	Image	0.55
	LSB Plane	0.40
	Histogram	0.45
	Image, LSB Plane, Histogram	0.45
	Chi-Squared Test	0.60
	Neighborhood Test	0.50
	Chi-Squared and Neighborhood Test	0.45

11.2 LSB Embedding

```
def get_size(img):
  print(img.shape)
  (M,N) = img.shape
  return (M,N)

def get_payload(img, rate):
  (M,N) = get_size(img)
  capacity = np.ceil(np.log2(M*N))
  K = int(rate * (M*N - capacity))
  payload = np.random.randint(2, size=K)
  return payload

def embed(img, payload):
  M,N = get_size(img)

  K = len(payload)
  t = int(np.ceil(np.log2(M*N)))
  img_dst = copy.deepcopy(img)

  kb = np.binary_repr(K, width=t)
  # print("Len (msg, img):", int(kb,2), M*N)

  tmp = img_dst.reshape(-1)
  h = 0

  # Embed K with t-bits
  for i in range(t):
    lsb = np.binary_repr(tmp[i], width=8)
    lsb = lsb[:-1] + kb[i]
    tmp[i] = int(lsb,2)
    h = i + 1

  # Embed Message Bits
  for i,p in enumerate(payload):
    i = i + h
    lsb = np.binary_repr(tmp[i], width=8)
    lsb = lsb[:-1] + str(p)
    # print(tmp[i])
    tmp[i] = int(lsb,2)

  img_dst = tmp.reshape(*img_dst.shape)
  return img_dst
```

11.3 Importing Images

```
# Function to retrieve a folder of images
def get_images(folder="/content/covers"):
  images = {} # name: image
  for file in os.listdir(folder):
    name, ext = os.path.splitext(file)
    path = os.path.join(folder, file)
    img = io.imread(path)
    if len(img.shape) >= 3:
      img = color.rgb2gray(img)
    img = util.img_as_ubyte(img) # Normalize
    img = transform.resize(img, (1024,1024), preserve_range=True).astype(np.uint8)
    images[name] = img
  return images

# Retrieve steganographic and cover images and save them to cover_images and stego_images
img_loc = 'Images/'
cover_loc = img_loc + 'cover/'
stego_loc = img_loc + 'stego/'
cover_images = get_images(cover_loc)
stego_images = get_images(stego_loc)
```

11.4 Creating a DataFrame

```
def image_frame(image_dict, stego=False):
    names = image_dict.keys()
    images = image_dict.values()
    data = {
        'name': list(names),
        'image': list(images),
    }
    df = pd.DataFrame(data)
    df['stego'] = stego
    return df

# Create a dataframe from the cover and stego images
df1 = image_frame(cover_images)

# Get an even split of embedded images of rates 0.1 and 0.4
# ASSUMPTION: Embedded images are exactly twice the size of the cover set of images where 50% are 0.1 and 50% are
df2 = image_frame(stego_images,stego=True)
df2 = df2.sort_values('name')[:int(len(df2)/2)]

# Merge dataframes
df = pd.concat([df1, df2], ignore_index=True)
```

11.5 Feature Engineering Formatting

```
# Drop potential duplicates
df.drop_duplicates('image')

# Encode the Labels (stego: True or False) as 1 and 0 for binary classification
df['stego'] = df['stego'].astype(int)

# Extracting LSB for each image and asserting it as a feature
df['lsb'] = df['image'].apply(lambda x: (x & 1).flatten())

# Extracting a histograph using the LSB
df['hist'] = df['lsb'].apply(lambda x: np.histogram(x,bins=2)[0])

# Extracting C Stat and P-val from chisq
df['c_stat'] = df['lsb'].apply(
    lambda x: chisq(x)[0]
)
df['p_val'] = df['lsb'].apply(
    lambda x: chisq(x)[1]
)
```

11.6 Chi-Squared Test

```
def chisq(lsb):
    obs = np.bincount(lsb, minlength=2)
    total = len(lsb)
    exp = np.array([total/2, total/2])

    cst, p_val, _, _ = chi2_contingency([obs, exp])
    return cst,p_val
```

11.7 Neighborhood Test

```
def neighborhood(img):
    x,y = img.shape
    x_c = int(x /2)
    y_c = int(y / 2)

    # Count of 1s
    c = 0

    while (K):
        for i in range(N) :
            for j in range(N):
                c = 0

                # Counting all neighbouring 1s
```

```
                if (i > 0 and arr[i - 1][j] == 1):
                    c += 1
                if (j > 0 and arr[i][j - 1] == 1):
                    c += 1
                if (i > 0 and j > 0 and
                    arr[i - 1][j - 1] == 1):
                    c += 1
                if (i < N - 1 and arr[i + 1][j] == 1):
                    c += 1
                if (j < N - 1 and arr[i][j + 1] == 1):
                    c += 1
                if (i < N - 1 and j < N - 1
                    and arr[i + 1][j + 1] == 1):
                    c += 1
                if (i < N - 1 and j > 0
                    and arr[i + 1][j - 1] == 1):
                    c += 1
                if (i > 0 and j < N - 1
                    and arr[i - 1][j + 1] == 1):
                    c += 1

                # Comparing the number of neighbouring
                # 1s with given ranges
                if (arr[i][j] == 1) :
                    if (c >= range1a and c <= range1b):
                        b[i][j] = 1
                    else:
                        b[i][j] = 0

                if (arr[i][j] == 0):
                    if (c >= range0a and c <= range0b):
                        b[i][j] = 1
                    else:
                        b[i][j] = 0
        K -= 1

        # Copying changes to the main matrix
        for k in range(N):
            for m in range( N):
                arr[k][m] = b[k][m]

    return arr
```

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009