Machine Learning Engineer Nanodegree

Capstone Project

Sourish Banerjee
May 1st, 2018

I. Definition

Project Overview

Steganography is the art of hiding messages in images by altering the least significant bits of the pixels in images with that of the message bits. The result is an image with a message hidden in it. However, the change is imperceptible to the human eye. This is because on changing the least significant bits in the pixels of an image, the pixel values are only altered by a small amount, resulting in a visually similar altered or steg image obtained from the original or cover image. Steganography has been widely used because of its potential capability to hide the existence of sensitive data. In situations where this kind of data hiding is illegal, potentially dangerous or inherently unethical, it becomes necessary to detect the presence of steganography in images.

Steganalysis is the art of detection of steganography in an image. There are two major types of steganography, and the preferred steganalysis methods for them are also different [1]. The naïve method is called LSB embedding. In this method, the LSB bit remains unchanged if the message bit is the same as the LSB bit, otherwise, the bit is altered. Hence, the odd pixels are reduced by 1 in intensity, whereas the even pixel values are incremented by 1. However, this causes an imbalance in the image histogram, which can be easily exploited by statistical methods for steganalysis. The second method of LSB steganography, LSB matching solves this issue by randomly incrementing or decrementing the pixel values by 1 in case of an LSB bit mismatch. This avoids the issue of histogram imbalance and makes it difficult to perform steganalysis by statistical methods alone.

In this project, I have implemented a detector for presence of LSB matching in greyscale images using classification techniques. Over the years, several feature sets have been proposed for the purpose of training, including HCFHOM [2], A. HCFCOM, C.A. HCFCOM [3] and HOMMS [4]. In my project, I have used the CF feature set proposed in [1] on the BOSSbase dataset. The implementation involves feature extraction from 20000 images (10000 cover and 10000 steg images with a payload of 0.4), training of classifiers on the resultant feature dataset, persisting of the models having the best performance, and building of a detector using a voting ensemble of these classifiers. The code for the feature extraction is available in the

'Image Preprocessing.ipynb' notebook. The original BOSSbase dataset has 10,000 greyscale images, each of size 512 x 512. All the images in the dataset are treated as the cover images. From this dataset, I have generated 10,000 corresponding images corrupted with LSB matching steganography with a payload of 0.40 (payload refers to the fraction of pixels in the original image that has been corrupted due to the steganography process). For this purpose, I have used the tool available in [2]. The tool to generate the steg images is as follows:

```
$ python aletheia.py lsbm-sim bossbase 0.40 bossbase_lsb
```

Problem Statement

My main goal is to classify greyscale images based on whether they are corrupted with LSB matching or not. I have used a labelled dataset of greyscale images, both with and without LSB matching and have developed a supervised learning workflow to solve this problem. The final result of the project is a tool that takes as input any grayscale image and gives a prediction as to whether the image is corrupted with LSB matching or not. The project workflow is detailed in brief under the project overview.

Metrics

I have used the F-Score as an evaluation metric for my benchmark and solution model. The F-Score is defined as the harmonic mean over precision and recall for a given test. Precision is the number of correct positive results (True Positives) divided by the number of all positive results returned by the classifier (True Positives + False Positives), and recall is the number of correct positive results True Positives) divided by the number of all samples that should have been identified as positive (True Positives + False Negatives). The above metrics are formalized as follows:

Precision = TP / (TP + FP)

Recall = TP / (TP + FN)

F-Score = (2 * Precision * Recall) / (Precision + Recall)

The F-Score is a good metric for the problem as applications of steganalysis require both good precision and recall. For example, in a naive use case where all detected cases of steganography are penalized, we would like to detect as many true cases as possible (high recall) while at the same time trying to prevent false positives and hence unjust penalties (high precision).

II. Analysis

Data Exploration

I Unlike that in LSB embedding, the imbalances caused by LSB matching in the spatial domain are not obvious enough to be exploited by statistical techniques alone. The CF feature set [1] comprises of 41 features and is based on the premise that given enough spatial information to train on, learners will eventually be able to pick up these subtler imbalances. Because LSB matching primarily alters the spatial data in the Least Significant Bit Plane (LSBP) and the Second Least Significant Bit Plane (LSBP2), the feature set mostly focuses the spatial information in these bit planes. Quantifying the correlation between bit planes and auto correlation within a bit plane are good ways of capturing such spatial information. This is the primary technique used in the CF feature set. It also captures the correlation between various slices of the image histogram in its density form, as well as the autocorrelations between the noise in the LSBP, which is obtained by subtracting the original image from various denoised images. These denoised image are in turn obtained via the removal of low complexity features from the Haar transform of the image using several different thresholding values.

Following are the formal expressions for each of the 41 features in the CF feature set:

M1 (1: m, 1: n) denotes the binary bits of LSBP and M2 (1: m, 1: n) denotes the binary bits of LSBP2.

C 1 = cor (M1, M2)

The autocorrelation of LSBP, C (k, l), is defined as:

C (k, l) = cor (Xk, Xl)

where,

Xk = M1(1: m – k, 1: n – l) and Xl = M1 (k + 1: m, l + 1: n)

Different values are set to k and l, and C2 to C15 is defined as:

C2 = C (1, 0); C3 = C (2, 0); C4 = C (3, 0);

C5 = C (4, 0); C6 = C (0, 1); C7 = C (0, 2);

C8 = C (0, 3); C9 = C (0, 4); C10 = C (1, 1);

C11 = C (2, 2); C12 = C (3, 3); C13 = C (4, 4);

C14 = C (1, 2); C15 = C (2, 1).

The histogram probability density, H, is denoted as (ρ0, ρ1, ρ2 …ρN-1). The histogram probability densities, He, Ho, Hl1, and Hl2 are denoted as follows:

He = (ρ0, ρ2, ρ4 …ρN-2), Ho = (ρ1, ρ3, ρ5 …ρN-1);

Hl1 = (ρ0, ρ1, ρ2 …ρN-1-l), Hl2 = (ρl, ρl+1, ρl+2 …ρN-1).

The autocorrelation coefficients C16 and CH(l) are defined as follows:

C16 = cor (He, Ho)

CH (l) = cor (Hl1, Hl2)

The features from C17 to C20 are defined as follows:

C17 = CH (1), C18 = CH (2),

C19 = CH (3), C20=CH (4).

Besides the features mentioned above, we consider the difference between test image and the denoised version. Firstly, the test image is decomposed by Haar wavelet. Zero is set to the coefficients in HL, LH and HH subbands, whose absolute value are smaller than some threshold value, t. The image is reconstructed according to the inverse wavelet transform. The reconstructed image is treated as denoised image. The difference between test image and reconstructed version is Et (t is the threshold value).

CE (E; k, l) = cor (Et, k, Et, l)

where,

Et, k = Et (1: m – k, 1: n – l) and Et, l = Et (k + 1: m, l + 1: n)

Different values are set to t, k and l, and features from C21 to C41 are defined as follows:

C21 = CE (1.5; 0, 1); C22 = CE (1.5; 1, 0);

C23 = CE (1.5; 1, 1); C24 = CE (1.5; 0, 2);

C25 = CE (1.5; 2, 0); C26 = CE (1.5; 1, 2);

C27 = CE (1.5; 2, 1); C28 = CE (2; 0, 1);

C29 = CE (2; 1, 0); C30 = CE (2; 1, 1);

C31 = CE (2; 0, 2); C32 = CE (2; 2, 0);

C33 = CE (2; 1, 2); C34 = CE (2; 2, 1);

C35 = CE (2.5; 0, 1); C36 = CE (2.5; 1, 0);

C37 = CE (2.5; 1, 1); C38 = CE (2.5; 0, 2);

C39 = CE (2.5; 2, 0); C40 = CE (2.5; 1, 2);

C41 = CE (2.5; 2, 1).

I have extracted the above features for each image in both the cover as well as the steg image datasets. The extraction code is implemented in the 'Image

Preprocessing.ipynb' notebook. The result is two csv files, 'steg_features.csv' containing 10000 rows representing the features extracted from the cover images and 'steg_lsb_features.csv' containing 10000 rows representing the features extracted from the steg images. The attributes in each row are the values of the correlation features for the corresponding image as defined above. Exploration of the data reveals that all the data values lie between -1 and 1. This is expected as all the attributes are Pearson's correlation coefficients, and its range is between -1 and 1. Some values in the csv files are NaN, which is the case because the denominator of the Pearson's correlation coefficient tends to 0 in certain situations. I have also explored the mean, median and fivefold summary of all the attributes in both the csv files to obtain a basic understanding of the data distributions for the attributes.

Exploratory Visualization

Since a lot of the features in the CF feature set include auto correlated values in different intervals, there is a high probability of observing strong correlations between the different attributes. To try and observe these correlations in the data, I have used a scatter matrix for the features, as well as a correlation heatmap. From these visualizations, it can clearly be observed that there are strong positive as well as negative correlations between several pairs of attributes. This leads me to believe that dimensionality reduction techniques like Principal Component Analysis can be performed on the data without significant loss in information. I have also plotted the kernel density estimation graphs for each attribute. Most of the attributes seem to follow a close to normal distribution. Features 15 to 18 seem to be highly right skewed. However, given the context of the data, I feel that non-linear corrections are not necessary. Note that these visualizations have been provided after certain data pre-processing steps have been performed. Procedures like removal of NaN values are necessary before these visualizations can be made. However, this section has been discussed in prior to conform to the report template.

Algorithms and Techniques

Several algorithms and techniques will be used in this project. They are described in detail in the following section.

Principal Component Analysis (PCA)

It is a way of identifying patterns in data, and expressing the data in such a way as to highlight their similarities and differences. Since patterns in data can be hard to find in data of high dimension, where the luxury of graphical representation is not available, PCA is a powerful tool for analysing data. The other main advantage of PCA is that once you have found these patterns in the data, and you compress the data, i.e. by reducing the number of dimensions, without much loss of information.

PCA works stepwise as described below:

- Standardize the data.

- Obtain the Eigenvectors and Eigenvalues from the covariance matrix or correlation matrix, or perform Singular Vector Decomposition.
- Sort eigenvalues in descending order and choose the k eigenvectors that correspond to the k largest eigenvalues where k is the number of dimensions of the new feature subspace.
- Construct the projection matrix W from the selected k eigenvectors.
- Transform the original dataset X via W to obtain a k-dimensional feature subspace Y.

PCA is used for dimensionality reduction of the data. This follows from the observation that several features in the dataset are highly correlated, and hence PCA is likely to achieve significant data reduction without a great loss in variance.

Gaussian Naive Bayes Classifier

The Naive Bayes classifier is based on Bayes Theorem. The naive Bayes classifier assumes all the features are independent to each other. Even if the features depend on each other or upon the existence of the other features. Naive Bayes classifier considers all of these properties to independently contribute to the probability. A Gaussian Naive Bayes algorithm is a special type of NB algorithm. It’s specifically used when the features have continuous values. It’s also assumed that all the features are following a Gaussian distribution i.e., normal distribution.

The Naive Bayes is a fast, simple and efficient algorithm. Because of these features, I have used it as a benchmark for my classification problem.

Random Forest Classifier

Random forest algorithm is a supervised classification algorithm. In general, the more trees in the forest, the more robust the forest looks like. In the same way in the random forest classifier, the higher the number of trees in the forest gives the high accuracy results. Given the training dataset with targets and features, a decision tree algorithm will come up with some set of rules. The same set rules can be used to perform the prediction on the test dataset. Random forest classifier will handle the missing values. When we have more trees in the forest, random forest classifier won’t overfit the model.

Random Forest:

- Randomly select “k” features from total “m” features, where k << m.
- Among the “k” features, calculate the node “d” using the best split point.
- Split the node into daughter nodes using the best split.
- Repeat 1 to 3 steps until “l” number of nodes has been reached.
- Build forest by repeating steps 1 to 4 for “n” number times to create “n” number of trees.

The Random Forest classifier combines the simplicity and lightweight-ness of decision trees with high performance and resistance to overfitting that is characteristic of ensemble learners. This makes it ideal for our classification problem. One disadvantage is that it is slow to train.

Support Vector Classifier (SVC)

Support Vector Machines (SVMs) are a set of supervised learning methods used for classification, regression and outlier detection. It uses support vector points to find the optimal decision boundary in the case where multiple candidate boundaries are possible. By taking advantage of different kernel functions, it can model non-linear decision boundaries as well.

The advantages of support vector machines are:

Effective in high dimensional spaces: Still effective in cases where number of dimensions is greater than the number of samples. Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient. Versatile: Different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation. This makes SVMs slow to train.

Multi-Layer Perceptron (MLP) Classifier

The multilayer perceptron (MLP) is a feedforward artificial neural network model that maps sets of input data onto a set of appropriate outputs. An MLP consists of multiple layers and each layer is fully connected to the following one. The nodes of the layers are neurons using nonlinear activation functions, except for the nodes of the input layer. There can be one or more non-linear hidden layers between the input and the output layer.

MLPs are good at modelling decision boundaries that are inherently complex in nature, even in very large dimensional spaces. Due to their high performance, MLPs are ideal for our learning problem.

Adaptive Boosting Classifier (AdaBoost)

AdaBoost is a type of "Ensemble Learning" where multiple learners are employed to build a stronger learning algorithm. AdaBoost works by choosing a base algorithm (e.g. decision trees) and iteratively improving it by accounting for the incorrectly classified examples in the training set. An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

We assign equal weights to all the training examples and choose a base algorithm. At each step of iteration, we apply the base algorithm to the training set and increase the weights of the incorrectly classified examples. We iterate n times, each time applying base learner on the training set with updated weights. The final model is the weighted sum of the n learners.

The advantages of AdaBoost are similar to that of the Random Forest Classifier. It is high performing and robust to outliers. This makes it a good choice for our learning problem.

Benchmark

I compare the performance of my final classification model against a benchmark model trained by a Gaussian Naïve Bayes learner. This serves as a check for solvability of the problem undertaken and provides a basis for comparison and interpretation of the performance scores of our final model. The Gaussian Naïve Bayes learner on learning the original dataset without cleaning returns a model with an accuracy score of 61% and an f-score of 68%.

III. Methodology

Data Pre-processing

The first step in data pre-processing was data cleaning, which involved removal of all rows having nan values. These were caused by overly uniform LSBP, which in turn caused some correlation and autocorrelation features to tend to infinity (Pearson’s correlation coefficient has standard deviation values in its denominator which tend to zero in these cases). Next was aggregation of the two csv file into a single data frame, followed by addition of the target column. I then plotted the scatter matrix and heatmap of the features in the dataset to spot correlations between the same.

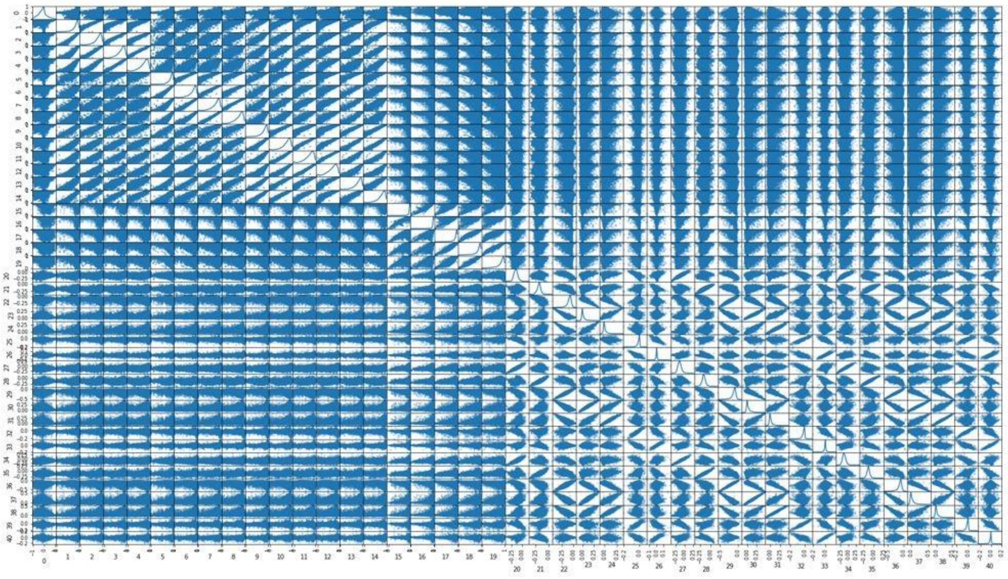


Fig 1: Scatter Matrix for the features in the dataset

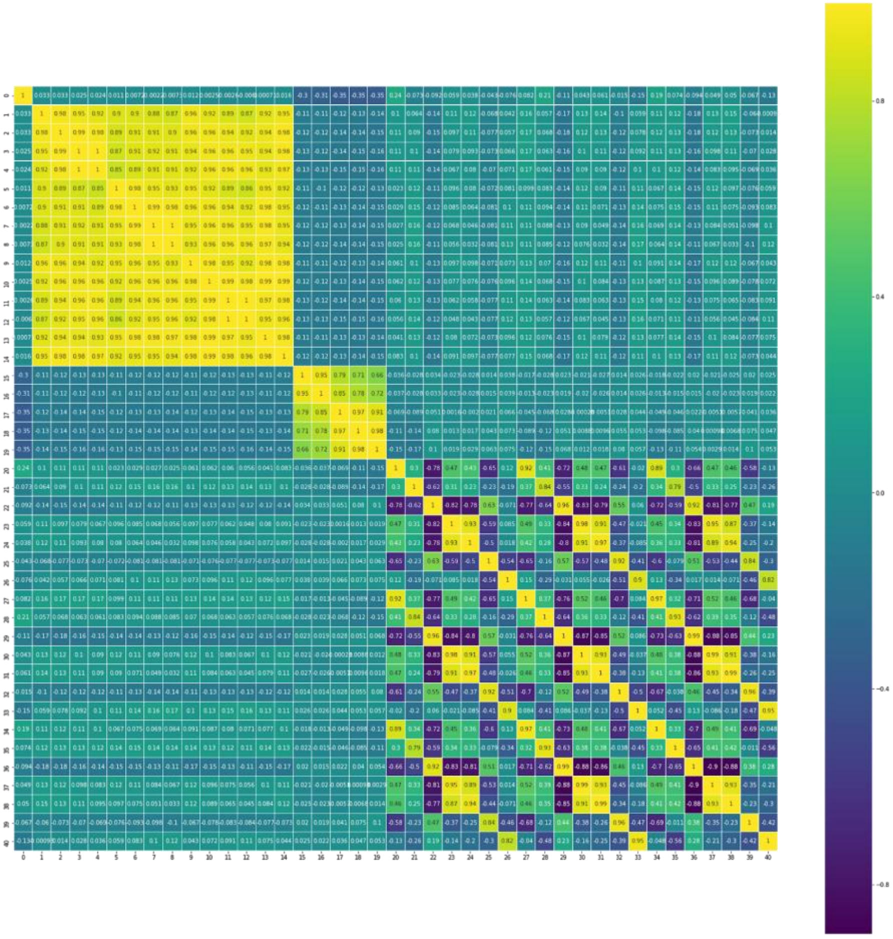


Fig 2: Heatmap for the features in the dataset

I noticed the presence of several strong correlations, both positive and negative from the above figures. This indicated the possibility of dimensionality reduction without a significant loss in information. I then went on to plot the kernel density estimation plots for each of the features in the dataset. My intention was to observe and transform in a non-linear fashion, any highly skewed feature. However, since none of the features were especially highly skewed, this step was not necessary.

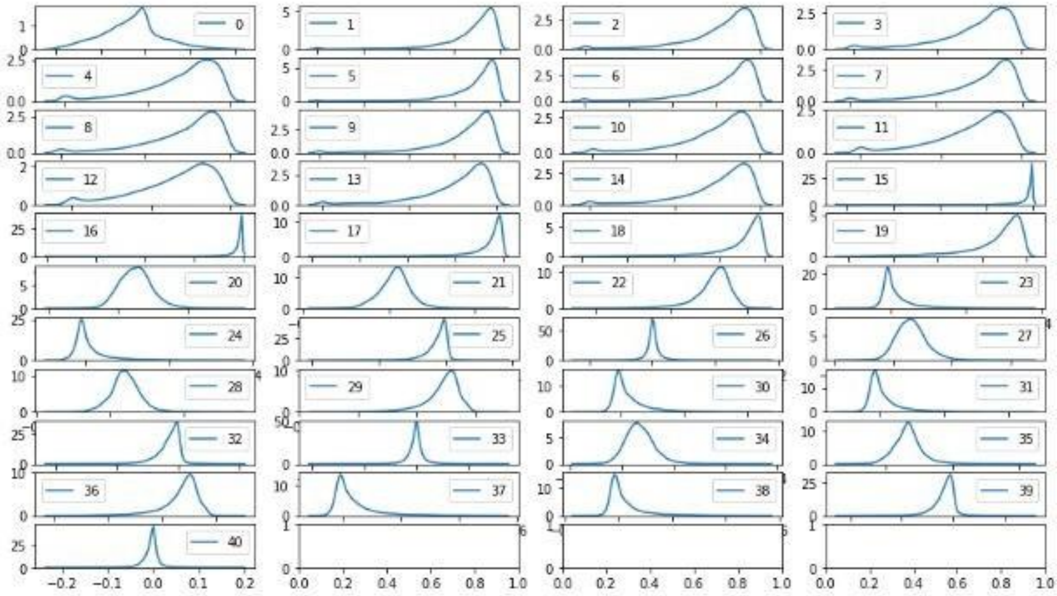


Fig 3: Kernel Density Estimation Plots for the features in the dataset

Outlier detection and removal is essential for high performance on several supervised as well as unsupervised techniques. I used the IQR rule for detection of outliers with respect to each feature. By this rule, an entry is considered an outlier with respect to a given feature if it lies outside the [Q1 – (1.5 * IQR): Q3 + (1.5 * IQR)] bracket, where Q1, Q3, and IQR are the first quartile, third quartile and inter quartile range respectively for the given feature. I then went on to remove all entries which were outliers with respect to more than 5 features.

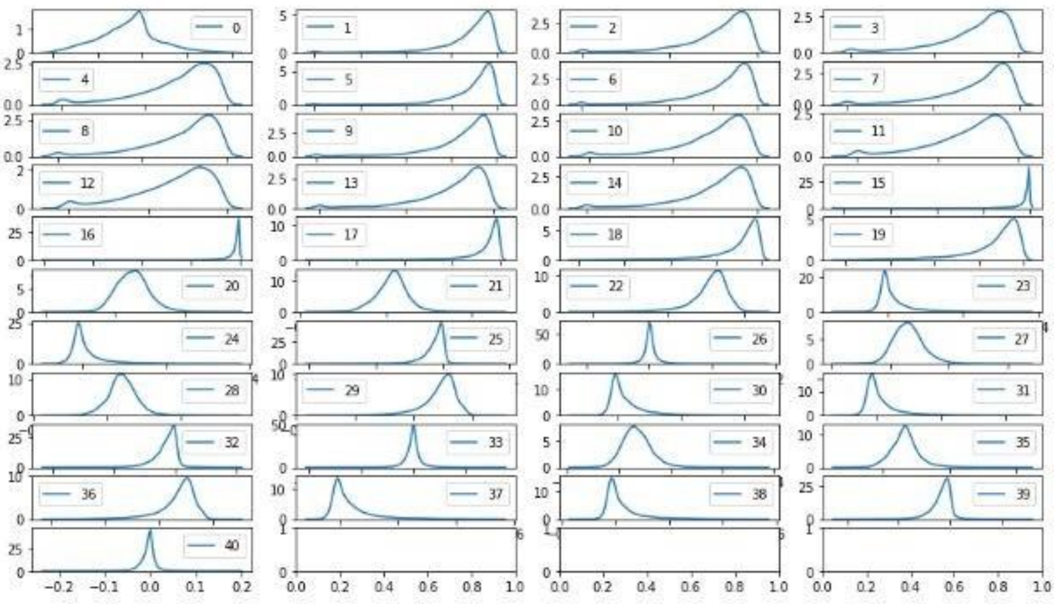
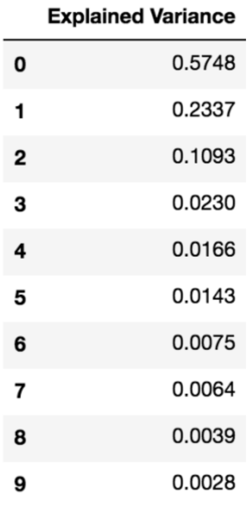


Fig 4: Kernel Density Estimation Plots for the features in the dataset after outlier removal

Finally, I performed Principal Component Analysis on the data. Removal of outliers was essential before this step as PCA is highly sensitive to outliers. PCA performs a linear transformation of the feature space that results in a new feature space in which each axis is in the direction of maximum variance in the data. Hence, the first k principal components (features after the transformation) are guaranteed to capture the maximum variance in the data, as opposed to any other combination of k features in the new feature space. After PCA, the first 10 principal components captured 99.23% of the variance in the data. Hence, the original 41 dimensional was reduced to only 10 dimensions with a mere 0.77% loss in variance.



	Explained Variance
0	0.5748
1	0.2337
2	0.1093
3	0.0230
4	0.0166
5	0.0143
6	0.0075
7	0.0064
8	0.0039
9	0.0028

Fig 5: Variance explained by the first 10 principal components

Implementation

I trained several classifiers on each of the original, cleaned and reduced datasets respectively. These classifiers include Gaussian Naïve Bayes, Random Forests, Support Vector Classifier, AdaBoost and a Multi-Layer Perceptron Neural Network. For each of the classifiers, I noted the performance based on the train time, prediction time, train accuracy, test accuracy, train f-score and test f-score for different training sizes. I also compared the accuracy and f-score with that of the Benchmark Gaussian Naïve Bayes Predictor on the original dataset. I found that the MLP Classifier and AdaBoost consistently outperformed all the other classifiers in terms of test accuracy and test f-score.

Refinement

Since the MLP Classifier and AdaBoost consistently outperformed all the other classifiers, I performed hyper parameter tuning on these classifiers using grid search. The AdaBoost classifier was tuned with respect to number of estimators and learning rate whereas the MLP Classifier was tuned with respect to its optimization tolerance and hidden layer structure. The tuned models for both the classifiers boasted improved performance. In the end, I exported the tuned models learned by the MLP Classifier and AdaBoost on the original and cleaned data. Thus, I had four high performing models which I used as voters in my ensemble voting based steganalysis prediction system. It is also worth mentioning that while the models trained on the reduced data were not as high performing, the comparatively smaller training time makes the reduced dataset ideal for training on larger datasets.

IV. Results

(approx. 2-3 pages)

Model Evaluation and Validation

In this section, the final model and any supporting qualities should be evaluated in detail. It should be clear how the final model was derived and why this model was chosen. In addition, some type of analysis should be used to validate the robustness of this model and its solution, such as manipulating the input data or environment to see how the model's solution is affected (this is called sensitivity analysis). Questions to ask yourself when writing this section:

- *Is the final model reasonable and aligning with solution expectations? Are the final parameters of the model appropriate?*
- *Has the final model been tested with various inputs to evaluate whether the model generalizes well to unseen data?*
- *Is the model robust enough for the problem? Do small perturbations (changes) in training data or the input space greatly affect the results?*
- *Can results found from the model be trusted?*

Justification

In this section, your model's final solution and its results should be compared to the benchmark you established earlier in the project using some type of statistical analysis. You should also justify whether these results and the solution are significant enough to have solved the problem posed in the project. Questions to ask yourself when writing this section:

- *Are the final results found stronger than the benchmark result reported earlier?*
- *Have you thoroughly analysed and discussed the final solution?*
- *Is the final solution significant enough to have solved the problem?*

V. Conclusion

(approx. 1-2 pages)

Free-Form Visualization

In this section, you will need to provide some form of visualization that emphasizes an important quality about the project. It is much more free-form, but should reasonably support a significant result or characteristic about the problem that you want to discuss. Questions to ask yourself when writing this section:

- *Have you visualized a relevant or important quality about the problem, dataset, input data, or results?*
- *Is the visualization thoroughly analysed and discussed?*
- *If a plot is provided, are the axes, title, and datum clearly defined?*

Reflection

In this section, you will summarize the entire end-to-end problem solution and discuss one or two particular aspects of the project you found interesting or difficult. You are expected to reflect on the project as a whole to show that you have a firm understanding of the entire process employed in your work. Questions to ask yourself when writing this section:

- *Have you thoroughly summarized the entire process you used for this project?*
- *Were there any interesting aspects of the project?*
- *Were there any difficult aspects of the project?*
- *Does the final model and solution fit your expectations for the problem, and should it be used in a general setting to solve these types of problems?*

Improvement

In this section, you will need to provide discussion as to how one aspect of the implementation you designed could be improved. As an example, consider ways your implementation can be made more general, and what would need to be modified. You do not need to make this improvement, but the potential solutions resulting from these changes are considered and compared/contrasted to your current solution. Questions to ask yourself when writing this section:

- *Are there further improvements that could be made on the algorithms or techniques you used in this project?*
- *Were there algorithms or techniques you researched that you did not know how to implement, but would consider using if you knew how?*
- *If you used your final solution as the new benchmark, do you think an even better solution exists?*