Conformal anomaly detection

Ryan Menzies

Submitted for the Degree of Master of Science in

Machine Learning MSc

Department of Computer Science
Royal Holloway University of London
Egham, Surrey TW20 0EX, UK

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 10,467

Student Name: Ryan Menzies

Date of Submission: 01-09-21

Signature:

Acknowledgement

I would like to thank my supervisor Alexander Gammerman. Their expert subject knowledge helped provide me with excellent source material and guidance throughout the project. Their advice helped shape the project to what it is today. I would also like to thank Cerys Evans for not only proof reading but for all the support they provided in what was a very turbulent and challenging time.

Abstract

This dissertation is devoted to conformal predictors and their applications to anomaly detection. The project explores visual aides in showing validity. It also looks into different nonconformity measures, comparing their predictive performances in both conformal predictors and conformal anomaly detectors. Predictions that produce a credibility less than 0.1 were consistently shown to produce incorrect predictions and/or shown to be produced from objects that can considered “strange”. The conformal predictors were applied to both the USPS and MNIST datasets.

Contents

1 Introduction

Given a predictive model, how confident can we be that the next prediction is accurate? This is a problem that conformal predictors (CPs) solve. CPs are built upon popular machine learning (ML) algorithms. They use previously seen predictions to calculate an accurate measure of confidence for the next prediction. These CPs can then be used to detect anomalies. These are called conformal anomaly detectors (CADs). Detecting anomalies can bring new insights whether it be detecting brand new classes, dirty data objects within a dataset, or showing where a ML model performs poorly.

The objective of this project is to show the varying levels of performance for CPs and CADs. A range of different CPs, such as: transductive, online, and inductive CPs have been used with each being shown to be valid. Since inductive conformal predictors (ICPs) are computationally efficient various ML algorithms such as nearest neighbour, support vector machine (SVM), decision tree, random forest, and neural networks have been adapted into ICPs. Visual guides have been provided to show the differences in performances when it comes to predictive efficiency and anomaly detection.

Experiments have been carried out to investigate how the credibility and/or confidence of predictions affect not only the predictive efficiency of a CP but whether those predictions can provide some insight into "strange" objects. Various images from the USPS and MNIST datasets, that provide varying levels of credibility have been shown for the reader to see how credibility effects the quality of the samples.

2 Background

2.1 Non-conformity measures

A *conformity measure* is a function that equivariantly maps a finite sequence of labelled samples $z_1, \ldots, z_m$, where $z_i = (x_i, y_i)$, to their corresponding conformity scores $\alpha_1, \ldots, \alpha_m$ as shown by Vovk in their lecture notes [10]. A conformity measure should show how much a sample conforms to the rest of the dataset. This is the Carnegie Mellon convention; where a measure of how much a sample conforms to a dataset is considered. In this project, the Royal Holloway convention will be used; where a measure of how much a sample does not conform to a dataset is considered. A small nonconformity score suggests a sample conforms to the rest of the dataset. Conversely, a large nonconformity score suggests the sample does not conform or is considered "strange". Hence, nonconformity can be used as a prediction measure.

In regression problems a popular nonconformity measure is $\alpha_i = |y_i - \hat{y}_i|$ for the predicted label $\hat{y}_i$. Using nonconformity scores in regression is mathematically simple and computationally efficient.

In classification problems the nonconformity measures normally depend on the machine learning model itself.

2.1.1 1-nearest neighbour

$d(u, v)$ is a distance measure between vectors u and v of same size. If $u, v \in \{0, 1\}^n$, then the Hamming distance can be used. The Hamming distance is defined as:

$$d(u, v) := |\{i = 1, \ldots, n : u_i \neq v_i\}|.$$

If $u, v \in \mathbb{R}^n$ then there are 3 popular distance measures.

- Euclidean distance:

$$d(u, v) := \sqrt{(u_1 - v_1)^2 + \ldots + (u_n - v_n)^2}.$$

- Manhattan distance:

$$d(u, v) := \Sigma_{i=1}^{n} |u_i - v_i|.$$

- Minkowski distance:

$$d(u, v) := (\Sigma_{i=1}^{n} |u_i - v_i|^p)^{\frac{1}{p}} \text{ where } p \in \mathbb{Z}^+.$$

For a new sample x, postulated label y and training set $\mathbf{D}_{\text{train}}$ of size m. Then for 1-nearest neighbour the following can be used as nonconformity measures:

$$A_1(x) := \min_{i=1,\ldots,m} \{d(x, x_i) : y = y_i\}. \tag{1}$$

The closer a new sample is to any sample in the training set with the same label the lower the nonconformity score. This nonconformity measure can potentially exclude information about differently labelled samples in the training set.

$$A_2(x) := \frac{1}{\min_{i=1,\ldots,m} \{d(x, x_i) : y \neq y_i\}}. \tag{2}$$

If a new sample is close to a sample in the training set which has a different label then the nonconformity score will be high. This can possibly suffer from a similar problem as the nonconformity measure in 1 by excluding some information from samples with the same label.

$$A_3(x) := \frac{\min\limits_{i=1,\ldots,m}\{d(x,x_i) : y = y_i\}}{\min\limits_{i=1,\ldots,m}\{d(x,x_i) : y \neq y_i\}}. \tag{3}$$

This combines both 1 and 2 so that it includes information about the nearest neighbour of the sample with both the same and different label.

All three of these conformity measures can be modified simply for KNN.

2.1.2 SKLearn

The scikit-learn (SKLearn) library [7] contains many built in ML algorithms including but not limited to: support vector machine (SVM), K-nearest neighbour, naive Bayes, decision trees and neural networks.

The SKLearn library contains a few variations of functions that can be applied to a CP. They are dependent on the underlying ML algorithm chosen.

For SVM multi-class classification the *SVC()* function from the *sklearn.svm* package is used. There are two functions that can be used as conformity scores that can easily be modified into nonconformity scores. The first is the function *predict_proba*. The SKLearn documentation for *SVC()* shows that *predict_proba* produces class membership probability estimates. Then since it outputs a probability estimate then the following nonconformity measure can be used for a sample x:

$$\alpha = -1 \cdot \mathit{SVC.predict\_proba}(x). \tag{4}$$

predict_log_proba produces the log probability estimate for class memberships. Similarly to 4, the following can be used as a nonconformity measure:

$$\alpha = -1 \cdot \mathit{SVC.predict\_log\_proba}(x). \tag{5}$$

Secondly, the function *decision_function* can be used as a conformity score. It gives a per-class score for each sample, meaning the higher the score for a class the more likely that sample belongs to that class. Then the following can be used as a nonconformity measure:

$$\alpha = -1 \cdot SVC.decision\_function(x). \tag{6}$$

The *predict_proba* and *predict_log_proba* functions can also be applied to the decision trees, neural network, naive Bayes and many other ML packages from scikit-learn.

The general non-conformity measure for a SKLearn model is:

$$\alpha = -1 \cdot model.predict\_proba(x). \tag{7}$$

2.2 Conformal Prediction

With the evolution of Machine Learning techniques, hardware, and access to larger datasets, ML models have been trained to make more accurate predictions. The issue with these predictions is that there is not always an accurate measure of confidence in the predictions; this is a problem solved by CPs.

CPs use past predictions to calculate an accurate measure of confidence in the next prediction. CPs are used in the on-line setting. CPs can be used for both classification and regression problems and can be applied to any machine learning model. They are built upon the assumption of randomness: the data objects are all independent and identically distributed (IID).

For any determined *significance level* ϵ (or *confidence* $1-\epsilon$) a *prediction region* Γ^ϵ can be calculated. For regression problems, the prediction region will be a closed interval, normally centred around the *point prediction* $\hat{y}$, $\Gamma^\epsilon = [\hat{y} - \beta, \hat{y} + \beta]$ where $\hat{y}, \beta \in \mathbb{R}$. For classification problems with label space $\mathbf{Y}$, the prediction region will be a set of all possible labels with p-value higher than ϵ. So, $\Gamma^\epsilon = \{y_i \in \mathbf{Y} | Pr(y_i) > \epsilon\}$. The point prediction in classification problems is $\arg\max_{y_i \in \mathbf{Y}}(Pr(y_i))$.

The p-value for a postulated label y_i is calculated as:

$$p^{y_j} = \frac{|\{i = 1, \ldots, m : \alpha_i \geq \ \alpha^{y_j}\}|}{m+1}. \tag{8}$$

If there are a lot of tie breaks then a lot of samples end up with the same p-values. To solve this a better p-value that can be used is the smoothed p-value:

$$p^{y_j} = \frac{|\{i = 1, \ldots, m : \alpha_i \geq \ \alpha^{y_j}\}| + \eta|\{i = 1, \ldots, m : \alpha_i = \alpha^{y_j}\}|}{m+1}, \tag{9}$$

where $\eta \sim uniform(0,1)$.

For CPs to be valid then two criteria must hold. The first, that the IID assumption holds. Secondly, that they show for the true label that $Pr(y \notin \Gamma^{\epsilon}) \leq \epsilon$. It is expected for the predictions to be wrong with frequency at most ϵ. This can be shown with the empirical calibration curve as shown by Vovk [11].

Let $D_{\text{train}} = \wr z_1, \ldots, z_m \int$ be a training set. The following is pseudocode for applying a full conformal predictor.

Algorithm 1 Conformal predictor (CP)

```
Require:
  • Training set $D_{\text{train}} = \wr z_1, \ldots, z_m \int$
  • Test set $D_{\text{test}} = \wr x_{m+1}, \ldots, x_{m+l} \int$
  • Non-conformity measure $A$.
  for all $i \in \{1, \ldots, m\}$ do
    Let $D^* = D_{\text{train}} \setminus z_i$
    Calculate $\alpha_i = A(D^*, z_i)$.
  end for
  for all $k \in \{1, \ldots, l\}$ do
    for all postulated labels $y_j \in \mathbf{Y}$ do
      Calculate $\alpha_{m+k}^{y_j} = A(D^*, (x_{m+k}, y_j))$.
      Calculate the p-value $p_{m+k}^{y_j} = \frac{|\{i=1,\ldots,m : \alpha_i \geq \alpha_{m+k}^{y_j}\}|}{m+1}$
    end for
  end for
```

2.3 Transductive CP

In most machine learning there are two main stages in making predictions. The first is the induction stage; the process of training a model or general rule. The second is the deduction stage in which the trained model is used to make predictions on new data. Vapnik [8] [9] suggests an alternate approach: making predictions for new data directly from the old data; this is called transduction.

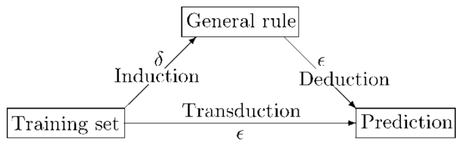


Figure 1: Figure by Gammerman [2]

The most obvious example of transductive learning is the k-nearest neighbour (KNN) algorithm. The predicted label of a new sample is the majority label of the nearest k samples in the training set. KNN does not require a "training" step, which is different to other ML approaches to making predictions.

2.4 Inductive CP

Inductive conformal predictors (ICP) work by using a training set to train a model to learn a prediction rule such that $f(x) = \hat{y}$. Then the p-values of the postulated labels are calculated using the nonconformity score of the postulated label against the non-conformity scores from a validation set. They are still provably valid.

Full conformal predictors can be quite computationally expensive. Hence, for extremely large and high dimensional datasets, they can be infeasible. ICP may suffer from less predictive efficiency than full CP, since they are trained on a smaller training set, but they are vastly more computationally efficient. This allows ICPs to be applied to large datasets like MNIST.

Algorithm 2 Inductive conformal predictor (ICP)

```
Require:
  • Training set $D_{\text{train}} = \wr z_1, \ldots, z_m \int$
  • Validation set $D_{\text{valid}} = \wr z_{m+1}, \ldots, z_{m+l} \int$
  • Test set $D_{\text{test}} = \wr x_{m+l+1}, \ldots, x_{m+l+k} \int$
  • Non-conformity measure $A$.

  Train model on training set
  for all $i \in \{1, \ldots, l\}$ do
     Calculate $\alpha_i = A(D_{\text{train}}, z_{m+i})$.
  end for
  for all $h \in \{1, \ldots, k\}$ do
     for all postulated labels $y_j \in \mathbf{Y}$ do
        Calculate $\alpha^{y_j}_{m+l+h} = A(D_{\text{train}}, (x_{m+l+h}, y_j))$.
        Calculate the p-value $p^{y_j}_{m+l+h} = \frac{|\{i=1,\ldots,l : \alpha_i \geq \alpha^{y_j}_{m+l+h}\}|}{l-m+1}$
     end for
  end for
```

2.5 Conformal anomaly detection

Hawkins [3]:"An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanis"

In statistics, an outlier or anomaly is a data sample x_i that does not conform to the rest of the data samples $\wr z_1, \ldots, z_m \int$. For conformal anomaly detectors (CADs), an anomaly is any sample x_i, where the prediction set $\Gamma^{\epsilon}_i = \emptyset$ where ϵ is the anomaly threshold. In other words, an anomaly is a sample that, for a defined anomaly threshold ϵ, has no predictions with p-values greater than or equal to ϵ. Since credibility is the largest p-value then anomalies can be seen when the credibility is less than the anomaly threshold. A problem with this definition is it can exclude some interesting cases such as samples which, according to the Hawkins definition, would be considered outliers. Laxhammer [5] states that there can be cases when an object's true label is excluded from the prediction set. They give the example that a poorly written "7" can look like "1". These examples could be considered as anomalies. Hence for anomalies, both empty prediction sets and prediction sets that do not include the true label can be considered as anomalies.

Laxhammer [5] stated the proposition:

Proposition 2.1. *Assume that $z_1, \dots, z_{l+1}$ are independent and identically distributed (IID). For any choice of non-conformity measure (NCM) A, the specified anomaly threshold ϵ corresponds to an upper bound of the probability of the event that z_{l+1} is classified as a conformal anomaly by conformal anomaly detector (CAD):*

$$Pr(Anom_{l+1}^{\epsilon} = 1) \leq \epsilon. \tag{10}$$

Proposition 2.1 states that as ϵ increases, so does the frequency of anomalies detected.

The following are pseudocode for a CAD and an inductive conformal anomaly detector (ICAD).

Algorithm 3 Conformal anomaly detector (CAD)

```
Require:
  • Training set $D_{\text{train}} = \wr z_1, \dots, z_m \int$.
  • Test set $D_{\text{test}} = \wr x_{m+1}, \dots, x_{m+l} \int$.
  • Non-conformity measure $A$.
  • Anomaly threshold $\epsilon \in (0, 1)$.

  Train model on training set
  for all $i \in \{1, \dots, m\}$ do
    Calculate $\alpha_i = A(D_{\text{train}}, z_{m+i})$.
  end for
  for all $h \in \{1, \dots, l\}$ do
    for all postulated labels $y_j \in \mathbf{Y}$ do
      Calculate $\alpha_{m+h}^{y_j} = A(D_{\text{train}}, (x_{m+h}, y_j))$.
      Calculate the p-value $p_{m+h}^{y_j} = \frac{|\{i=1,\dots,m : \alpha_i \geq \alpha_{m+h}^{y_j}\}|}{l+1}$
    end for
    if $p_{m+h}^{y} < \epsilon, \forall y \in \mathbf{Y}$ then
      $\text{Anom}_{m+h}^{\epsilon} = 1$
    else
      $\text{Anom}_{m+h}^{\epsilon} = 0$
    end if
  end for
```

Algorithm 4 Inductive conformal anomaly detector (ICAD)

```
Require:
 • Training set $D_{\text{train}} = \lbag z_1, \dots, z_m \rbag$
 • Validation set $D_{\text{valid}} = \lbag z_{m+1}, \dots, z_{m+l} \rbag$
 • Test set $D_{\text{test}} = \lbag x_{m+l+1}, \dots, x_{m+l+k} \rbag$
 • Non-conformity measure $A$.
 • Anomaly threshold $\epsilon \in (0, 1)$.

 Train model on training set
 for all $i \in \{1, \dots, l\}$ do
   Calculate $\alpha_i = A(D_{\text{train}}, z_{m+i})$.
 end for
 for all $h \in \{1, \dots, k\}$ do
   for all postulated labels $y_j \in \mathbf{Y}$ do
     Calculate $\alpha^{y_j}_{m+l+h} = A(D_{\text{train}}, (x_{m+l+h}, y_j))$.
     Calculate the p-value $p^{y_j}_{m+l+h} = \frac{|\{i=m+1,\dots,m+l: \alpha_i \geq \alpha^{y_j}_{m+l+h}\}|+1}{l-m+1}$
   end for
   if $p^{y}_{m+l+h} < \epsilon, \forall y \in \mathbf{Y}$ then
     $\text{Anom}^{\epsilon}_{m+l+h} = 1$
   else
     $\text{Anom}^{\epsilon}_{m+l+h} = 0$
   end if
 end for
```

3 Datasets

This project makes use of three different datasets. Since conformal predictors are built upon the assumption of randomness [2], then for this project it is assumed that each dataset contains independent and identically distributed samples. To further strengthen this assumption, each dataset has been randomly shuffled and split into training, validation and test sets.

USPS handwritten digits The USPS dataset [4] contains 9298 16x16 grayscale pixel images of individual numbers. The images were collected from scanned envelopes by the U.S. Postal Service. In figure 2 are 9 random samples from this dataset with their true label.

The dataset has been randomly split into the following subsets:

- Training set: 5,229 samples.

- Validation set: 1,744 samples.
- Test set: 2325 samples.

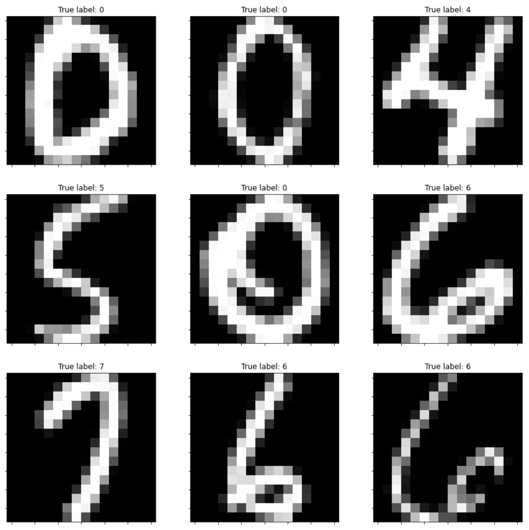


Figure 2: Random samples of the USPS dataset.

Synthetic data A test set compromising of 10,000 samples was generated for anomaly detection to be used alongside the USPS dataset. Each sample is a vector of size 256 where each entry x_i is drawn from a uniform distribution such that $x_i \sim U(-1, 1)$. A small sample can be seen in figure 3.

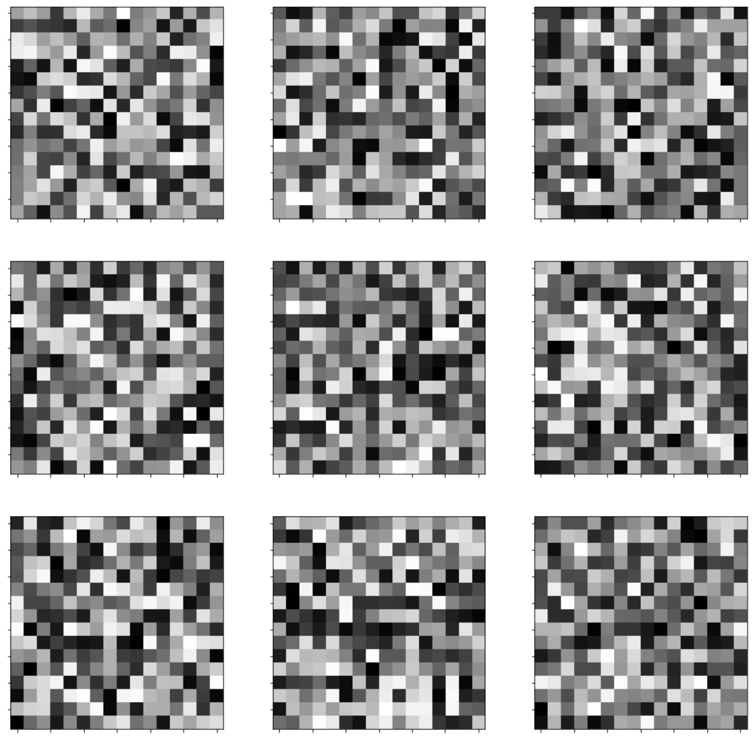


Figure 3: Random sample of generated test set.

MNIST Similar to the USPS dataset, the MNIST dataset [6] (a subset to a larger dataset called NIST) contains images of handwritten numbers ranging from 0-9. The images are of a higher resolution; each image is 28x28 grayscale pixel image. The dataset originally consists of 60,000 training images and 10,000 test images.

The dataset has been randomly split into the following subsets:

- Training set: 52,500 samples.
- Validation set: 13,125 samples.

- Test set: 17,500 samples.

In figure 4 are random samples from the MNIST dataset.

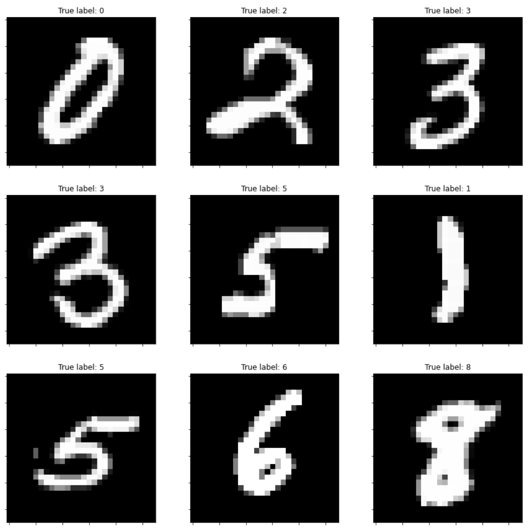


Figure 4: Random samples of the MNIST dataset.

EMNIST Letters The EMNIST letters dataset [1] contains 145,600 images of handwritten character digits. Each image is a 28x28 grayscale pixel image. It contains both upper and lowercase letters. For simplicity, the label of the image does not depend on the case, i,e, an image of an 'a' and 'A' will have the same label.

This dataset will be used for anomaly detection in the second part of the results section as a test set. For this reason, only a small random subset is required. A test set of size 10,000 random samples has been chosen since it should give an accurate measure of performance.

Some random samples can be seen in figure 5 with their corresponding label.

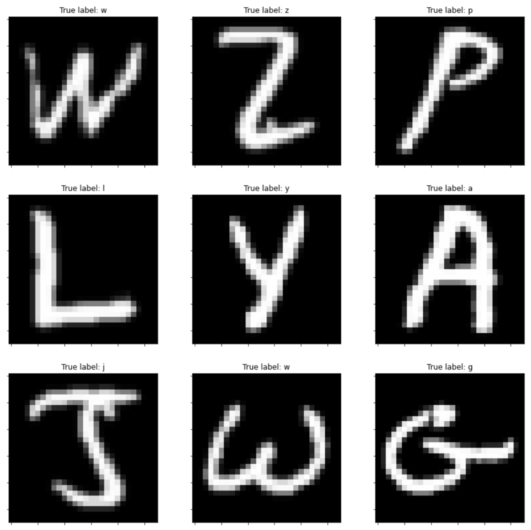


Figure 5: Random samples of the EMNIST letters dataset.

4 Results and discussion

All CPs need to be shown to be valid before they are applied to such tasks such as anomaly detection. For this reason, this section has been divided into two parts. The first part is dedicated to conformal predictors and their proof of validity. The second part is dedicated to anomaly detection using conformal predictors.

4.1 Conformal predictors

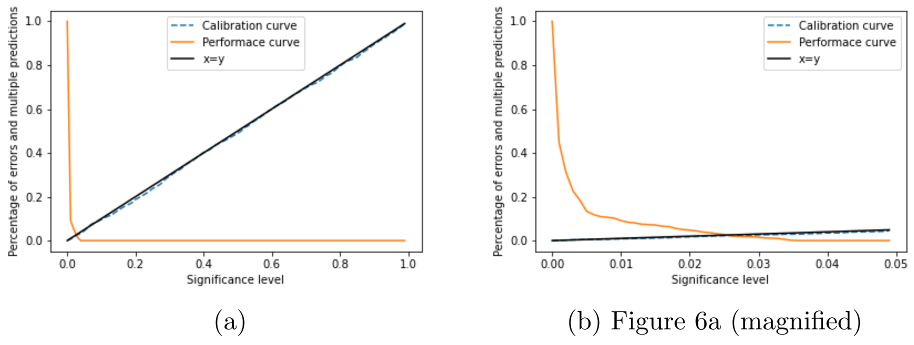
In this section conformal predictors have been applied to the USPS dataset and then the more computationally efficient CPs have been applied to the much larger MNIST dataset. Each CP has been shown to be valid using the empirical calibration curve.

4.1.1 Conformal predictors with USPS

Since the USPS dataset is considerably smaller than the MNIST dataset it gives the ability to experiment with more computationally expensive CPs such as a full CP or OCP. The OCP used in this project was found to be too inefficient to be applied to larger datasets. It was still included in the project since it provides very interesting results in showing performance and proof of validity. Later, inductive conformal predictors have been applied since they are so much faster to run. This gave the ability to experiment with a variety of different underlying ML algorithms and nonconformity measures. The predictive performance of these ICPs have then been compared against one-another.

Transductive conformal predictor with nearest neighbour The first implementation of a transductive conformal predictor in this project uses the nonconformity measure from equation 3 and uses Euclidean distance as a distance measure.

It is important to check that validity holds, because if validity does not hold then the conformal predictor fails. First, the IID assumption must hold. To strengthen this assumption, the initial training set and test set were merged together, then randomly split into new training and test sets using the *train_test_split* function in the *sklearn.preprocessing* package. This ensures that the samples in the new training and test set are independent from one another. Next, for the CP to be valid, the CP is expected to be wrong with frequency less than or approximately ϵ. This is shown by the calibration curve in figure 6a.



(a) (b) Figure 6a (magnified)

Figure 6: Calibration curve for nearest neighbour TCP on USPS dataset.

Figure 6 also includes a performance curve showing for each significance level what percentage of test samples have multiple predictions. A small percentage of multiple predictions for a small significance level suggests better predictive efficiency. Figure 6 shows that the CP performs very well with the nearest neighbour nonconformity measure 3. At significance level $\epsilon = 0.035$ the CP makes almost 0 multiple predictions.

Table 1 details the p-values of 5 random samples produced by the 1-nearest neighbour CP. The p-value is largest for the true label in each of the random samples. It can also be seen that the p-values for the incorrect labels are all very small showing that each of these predictions would have high confidence. For this CP the p-values have not been smoothed, since there are multiples of the same value. This happens when there are ties in nonconformity scores. Later in this project when dealing with the ICPs, the p-values have been smoothed.

0	1	2	3	4	5	6	7	8	9	Label
0.001004	0.006309	0.129624	0.001147	0.002868	0.002294	0.002868	0.002581	0.004732	0.002868	2
0.908231	0.000287	0.000430	0.000430	0.000287	0.000430	0.000430	0.000287	0.000430	0.000287	0
0.000143	0.181818	0.000143	0.000143	0.004302	0.000143	0.000287	0.000287	0.000287	0.000287	1
0.000574	0.000430	0.277459	0.000574	0.000574	0.000430	0.000430	0.002581	0.000574	0.000860	2
0.000430	0.000430	0.000430	0.000430	0.000430	0.000430	0.000430	0.715515	0.000430	0.000430	7

Table 1: Random sample with their corresponding p-values for nearest neighbour TCP on USPS dataset.

Since confidence is 1 minus the second highest p-value, a high confidence implies that a sample has a high p-value for only one of the postulated labels, suggesting the object conforms to one class. Hence, it is expected that predictions with high confidence will make correct predictions. A low credibility implies that the each p-value for each label is small: it is expected that predictions with low credibility will make incorrect predictions since these objects do not conform to any of the classes. Conversely, samples that produce predictions with high credibility are expected to be correct. This can be seen in figure 7. As both the credibility and confidence increase, the frequency of correct predictions increases. Conversely, as both confidence and credibility decreases, so does the frequency of correct predictions. The CP starts to make a substantial amount of incorrect predictions for $\epsilon \leq 0.1$ or for confidence less than 0.99.

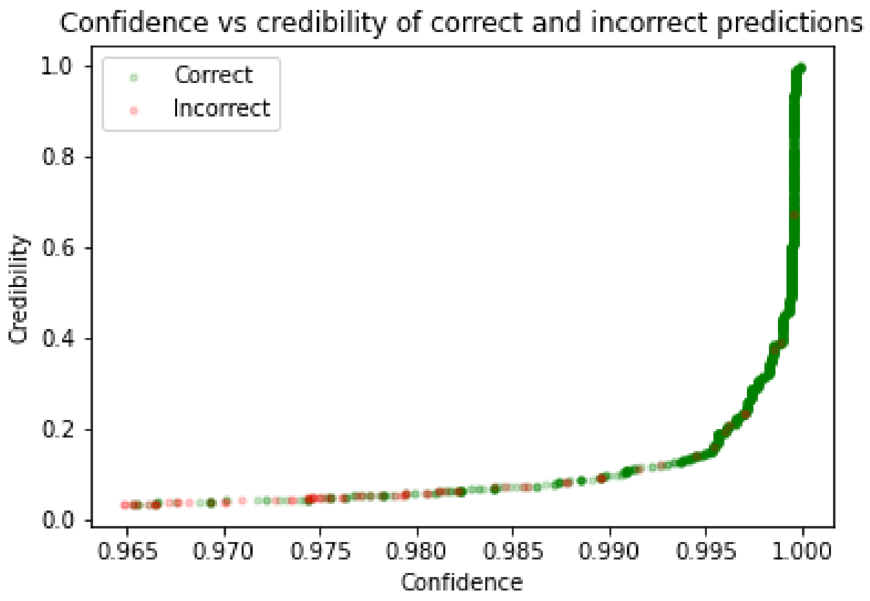


Figure 7: Confidence vs credibility for predictions made by a transductive CP using a 1-nearest neighbour non-conformity measure on USPS dataset.

In figures 8 - 13, are the predictions from the nearest neighbour CP with varying levels of credibility. The general trend is as credibility decreases so does the quality of the images. It can be seen in figure 8 that the predictions with the highest levels of credibility are mostly ones and some zeroes. As the credibility decreases it is interesting to see how the numbers become less clear. When the credibility becomes extremely low (for $\epsilon \leq 0.1$) then the samples become “abnormal”. For example, in figure 12, the top left image looks more like the letter u than the number 0. Surprisingly the CP still correctly predicts the label of the sample and does so with high confidence. There is a similar example in figure 13 where the 2nd example in the 3rd row looks like the letter h. The CP incorrectly predicts this sample as a 2. There are a few more examples in 12 and 13 which could be considered as abnormal. This again suggests that predictions with $\epsilon \leq 0.1$ could be of use in anomaly detection.

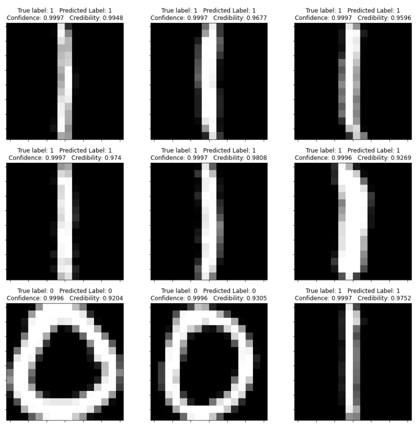

True label: 1 Predicted Label: 1
Confidence: 0.9997 Credibility: 0.9948
True label: 1 Predicted Label: 1
Confidence: 0.9997 Credibility: 0.9677
True label: 1 Predicted Label: 1
Confidence: 0.9997 Credibility: 0.9596
True label: 1 Predicted Label: 1
Confidence: 0.9997 Credibility: 0.974
True label: 1 Predicted Label: 1
Confidence: 0.9997 Credibility: 0.9808
True label: 1 Predicted Label: 1
Confidence: 0.9996 Credibility: 0.9269
True label: 0 Predicted Label: 0
Confidence: 0.9996 Credibility: 0.9204
True label: 0 Predicted Label: 0
Confidence: 0.9996 Credibility: 0.9305
True label: 1 Predicted Label: 1
Confidence: 0.9997 Credibility: 0.9752

Figure 8: Samples with credibility between 0.9 and 1

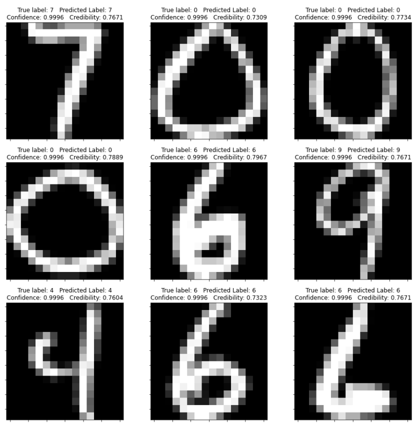

True label: 7 Predicted Label: 7
Confidence: 0.9996 Credibility: 0.7671
True label: 0 Predicted Label: 0
Confidence: 0.9996 Credibility: 0.7309
True label: 0 Predicted Label: 0
Confidence: 0.9996 Credibility: 0.7734
True label: 0 Predicted Label: 0
Confidence: 0.9996 Credibility: 0.7889
True label: 6 Predicted Label: 6
Confidence: 0.9996 Credibility: 0.7967
True label: 9 Predicted Label: 9
Confidence: 0.9996 Credibility: 0.7671
True label: 4 Predicted Label: 4
Confidence: 0.9996 Credibility: 0.7604
True label: 6 Predicted Label: 6
Confidence: 0.9996 Credibility: 0.7323
True label: 6 Predicted Label: 6
Confidence: 0.9996 Credibility: 0.7671

Figure 9: Samples with credibility between 0.7 and 0.8

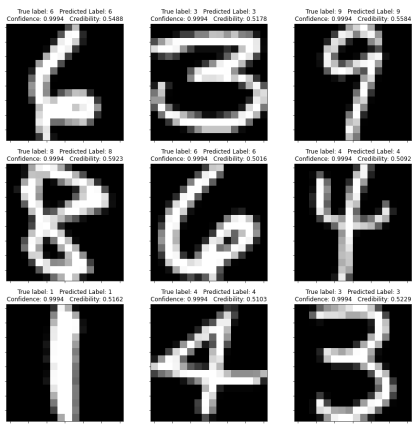

True label: 6 Predicted Label: 6
Confidence: 0.9994 Credibility: 0.5488
True label: 3 Predicted Label: 3
Confidence: 0.9994 Credibility: 0.5178
True label: 9 Predicted Label: 9
Confidence: 0.9994 Credibility: 0.5584
True label: 8 Predicted Label: 8
Confidence: 0.9994 Credibility: 0.5923
True label: 6 Predicted Label: 6
Confidence: 0.9994 Credibility: 0.5016
True label: 4 Predicted Label: 4
Confidence: 0.9994 Credibility: 0.5092
True label: 1 Predicted Label: 1
Confidence: 0.9994 Credibility: 0.5162
True label: 4 Predicted Label: 4
Confidence: 0.9994 Credibility: 0.5103
True label: 3 Predicted Label: 3
Confidence: 0.9994 Credibility: 0.5229

Figure 10: Samples with credibility between 0.5 and 0.6

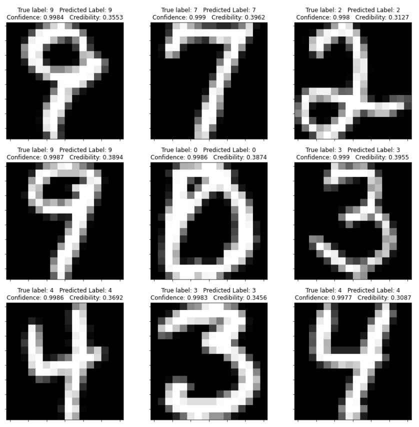

True label: 9 Predicted Label: 9
Confidence: 0.9984 Credibility: 0.3553
True label: 7 Predicted Label: 7
Confidence: 0.999 Credibility: 0.3962
True label: 2 Predicted Label: 2
Confidence: 0.998 Credibility: 0.3127
True label: 9 Predicted Label: 9
Confidence: 0.9987 Credibility: 0.3894
True label: 0 Predicted Label: 0
Confidence: 0.9986 Credibility: 0.3874
True label: 3 Predicted Label: 3
Confidence: 0.999 Credibility: 0.3955
True label: 4 Predicted Label: 4
Confidence: 0.9986 Credibility: 0.3692
True label: 3 Predicted Label: 3
Confidence: 0.9983 Credibility: 0.3456
True label: 4 Predicted Label: 4
Confidence: 0.9977 Credibility: 0.3087

Figure 11: Samples with credibility between 0.3 and 0.4

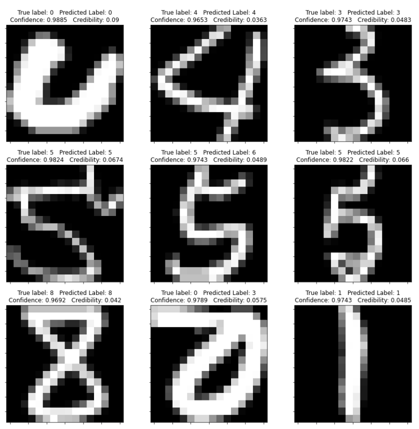

True label: 0 Predicted Label: 0
Confidence: 0.9885 Credibility: 0.09
True label: 4 Predicted Label: 4
Confidence: 0.9653 Credibility: 0.0363
True label: 3 Predicted Label: 3
Confidence: 0.9743 Credibility: 0.0483
True label: 5 Predicted Label: 5
Confidence: 0.9824 Credibility: 0.0674
True label: 5 Predicted Label: 6
Confidence: 0.9743 Credibility: 0.0489
True label: 5 Predicted Label: 5
Confidence: 0.9822 Credibility: 0.066
True label: 8 Predicted Label: 8
Confidence: 0.9692 Credibility: 0.042
True label: 0 Predicted Label: 3
Confidence: 0.9789 Credibility: 0.0575
True label: 1 Predicted Label: 1
Confidence: 0.9743 Credibility: 0.0485

Figure 12: Samples with credibility between 0 and 0.1

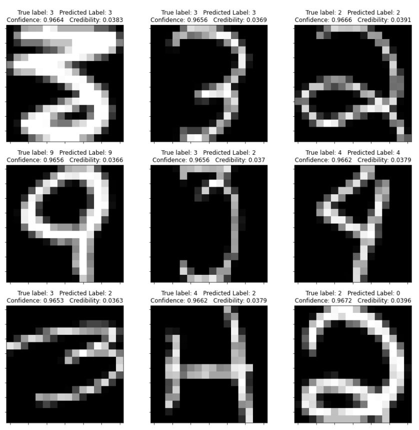

True label: 3 Predicted Label: 3
Confidence: 0.9664 Credibility: 0.0383
True label: 3 Predicted Label: 3
Confidence: 0.9656 Credibility: 0.0369
True label: 2 Predicted Label: 2
Confidence: 0.9666 Credibility: 0.0391
True label: 9 Predicted Label: 9
Confidence: 0.9656 Credibility: 0.0366
True label: 3 Predicted Label: 2
Confidence: 0.9656 Credibility: 0.037
True label: 4 Predicted Label: 4
Confidence: 0.9662 Credibility: 0.0379
True label: 3 Predicted Label: 2
Confidence: 0.9653 Credibility: 0.0363
True label: 4 Predicted Label: 2
Confidence: 0.9662 Credibility: 0.0379
True label: 2 Predicted Label: 0
Confidence: 0.9672 Credibility: 0.0396

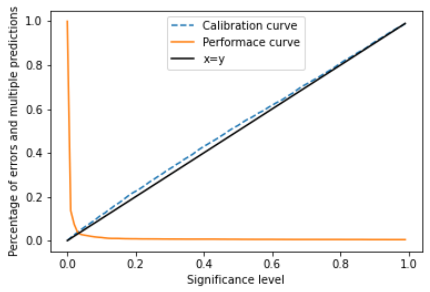
Figure 13: Samples with credibility between 0 and 0.1

On-line conformal predictor with nearest neighbour Conformal predictors can be adapted in the on-line setting. *On-line conformal pre-*

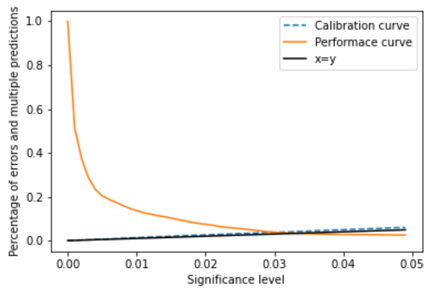
dictors (OCPz) provide a prediction for each new sample and add the new sample into the training dataset. Gammerman [2] states, in the pure on-line setting, CP may not be completely practical. However, as there is access to such large labelled datasets, then it is interesting to investigate how they perform.

The OCP used the nonconformity measure from equation 3 and Euclidean distance as a distance measure, just like the TCP used in the previous part. There were some difficulties running the OCP since it was computationally inefficient, but it still produced good results that have been shown below. The OCP used the full USPS data, being updated after each new sample was added to the training set.

In figure 14a the calibration curve shows that the OCP is almost a perfect diagonal but bows slightly. It bows into the region that breaks $Pr(y \notin \Gamma^{\epsilon}) \leq \epsilon$. This can suggest that the OCP used might be poorly calibrated or that the OCP is not valid. Since the calibration curve is still very tight towards the diagonal, the predictions produced should still be fairly reliable but it is important to remember that validity might not be held. Also shown is the performance curve, similarly to the previous section, the performance of the OCP is very high. From 14b, it is noticeable that the OCP has lost a bit of performance compared to the TCP in the previous section; the new CP now makes zero multiple predictions for significance level $\epsilon \geq 0.5$.

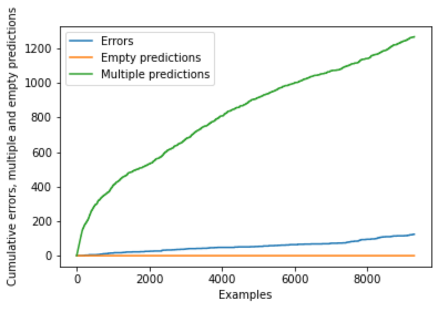


(a)

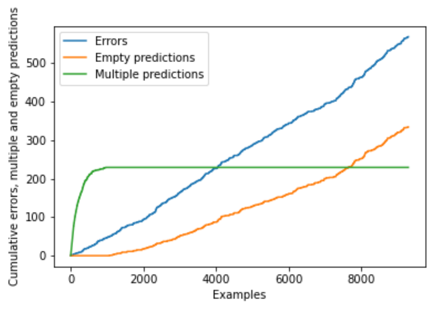


(b) Figure 14a magnified on the x-axis

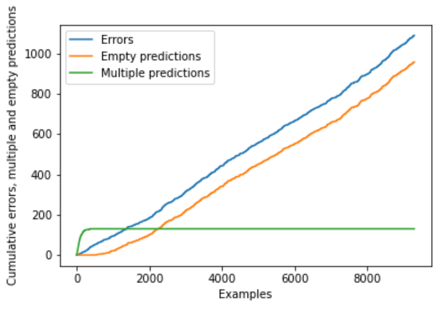
Figure 14: Performance and calibration curve for an OCP using the USPS dataset.



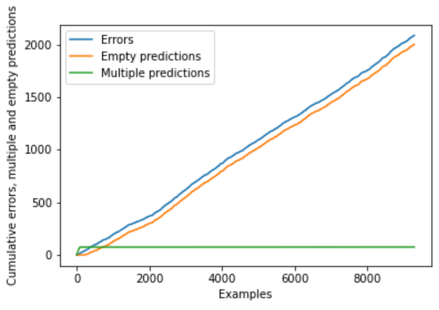
(a) Significance level $\epsilon = 0.01$



(b) Significance level $\epsilon = 0.05$



(c) Significance level $\epsilon = 0.1$



(d) Significance level $\epsilon = 0.2$

Figure 15: 1-nearest neighbour OCP cumulative errors, multiple and empty predictions for significance level ϵ on USPS dataset.

Figure 15 clearly shows as ϵ grows, so does the frequency of errors made. This is expected as this the basis of the validity of the CP; it can be observed directly from these graphs that validity holds. It would be expected that

$$\frac{\mathrm{Err}_n^\epsilon}{n} \lessapprox \epsilon$$

for number of examples n.

Since $\Gamma^{\epsilon_1} \subseteq \Gamma^{\epsilon_2}$ for $\epsilon_1 \geq \epsilon_2$. Then

$$|\Gamma^{\epsilon_1}| \leq |\Gamma^{\epsilon_2}|. \tag{11}$$

This implies that as the significance level decreases, the size of the prediction set decreases. This can be seen in figure 15 and 17. It can also be seen that the curve for multiple predictions converges more quickly for larger ϵ. Since the curve converges this implies that the OCP's predictive efficiency improves as it is trained on more samples.

For a label y_i to be in a prediction set Γ^ϵ then its p-value $p^{y_i} \geq \epsilon$. Hence for larger ϵ, the total number of empty prediction sets will increase. This can be seen clearly in figure 15.

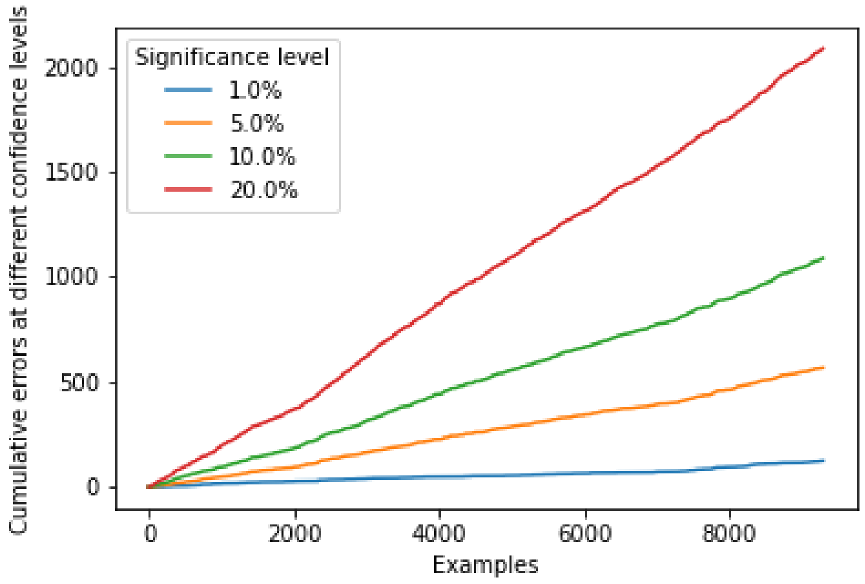


Figure 16: 1-nearest neighbour OCP cumulative errors for different significance levels.

Average prediction set size can be used as a measure of performance. The smaller the average prediction set size for a chosen ϵ, the better the performance of the CP. As expected for CP in the on-line setting, the performance increases after each new sample as shown in 17.

Another good measure of performance of a CP is to use the average false p-value as suggested by Vovk [10]. The smaller the average false p-value, the better performance the CP has. In an on-line setting, the false p-value can be calculated after each prediction. From figure 18, it can be seen that for the first 1000 examples there is downward trend showing improvement in predictive efficiency of the CP. After the first 1000 examples, the trend plateaus.

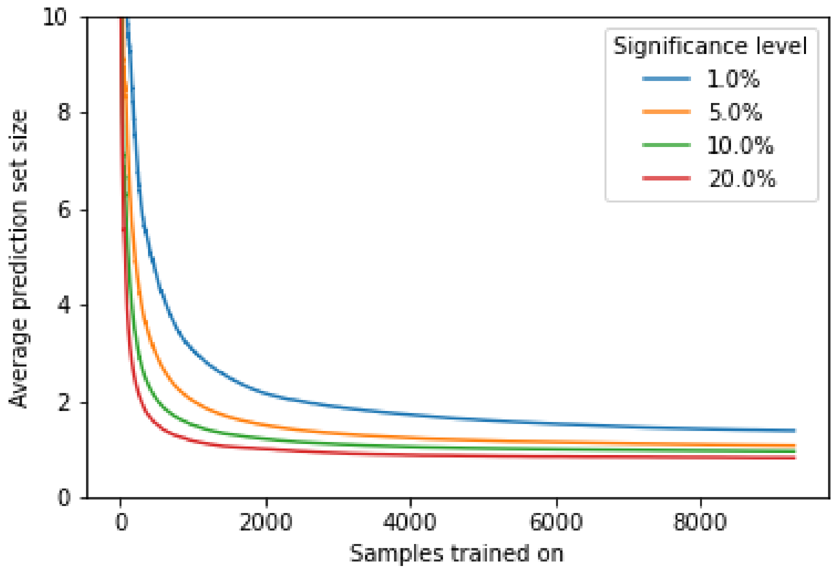


Figure 17: Average prediction set size after each epoch for 1-nearest neighbour OCP on USPS dataset.

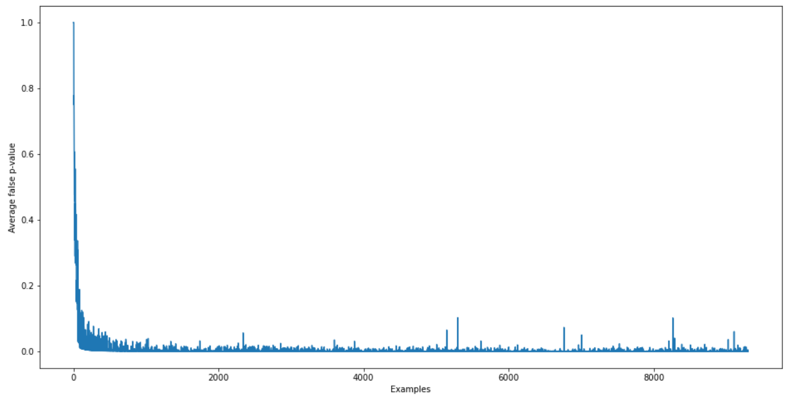


Figure 18: Average false p-values after each epoch for 1-nearest neighbour OCP on USPS dataset.

Inductive conformal predictors ICPs trade some small predictive efficiency, since they use a smaller training set, for a large amount of computational efficiency. They run much faster than TCPs and OCPs. Four different ICPs have been used: 1-nearest neighbour, SVM, neural network, and random forest. For 1-nearest neighbour, the nonconformity from equation 3 has been used. The SVM, decision tree, neural network and random forest TCP have been built using the SKLearn packages. The nonconformity measure from equation 7 has been used. When calculating the p-values, the smooth p-value has been used.

For following is the code for each of the SKLearn models:

```
SVM_model = SVC(probability=True)
net_model = MLPClassifier(solver='adam',
                          hidden_layer_sizes =
                          (16*16,16*16,16*16,10),
                          max_iter=100,
                          activation = 'relu',
                          batch_size=32
                         ),
tree_model_usps = DecisionTreeClassifier(max_depth=20,
                                         max_leaf_nodes=100),
rf_model = RandomForestClassifier(max_depth=10)
```

From figures 19 and 20 it can be seen that the 1-nearest neighbour, SVM, decision tree, neural network, and the random forest ICP are valid. The neural network has a much tighter fit to the diagonal than the other ICPs. The calibration curves for each ICP show that each ICP is valid. Each of the ICP's performance curve are almost identical apart from the decision tree suggesting they have similar predictive efficiencies. The calibration curve for the decision tree TCP, although valid, is poorly calibrated suggesting the decision tree does not fit the data well. The performance curve for the decision tree ICP makes multiple predictions for the the highest significance level. This is important to consider since different ML algorithms take a different amount of computational power to run. For example the neural network ICP was the slowest to run, slightly beaten by the SVM ICP. The 1-nearest neighbour ICP was very fast to run but even that could not even compete with the extremely fast random forest ICP that completed its training in moments even on a less than ideal PC. Given the similarity in performance, this comparison of time and power taken to achieve this is important to consider; by this consideration, the random forest ICP outperformed the others.

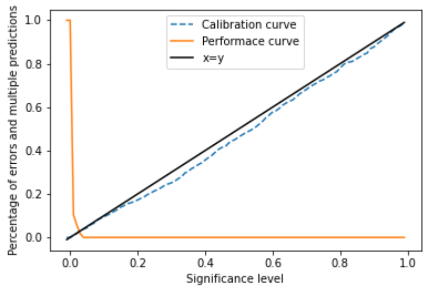
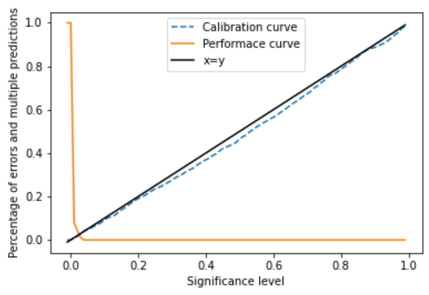
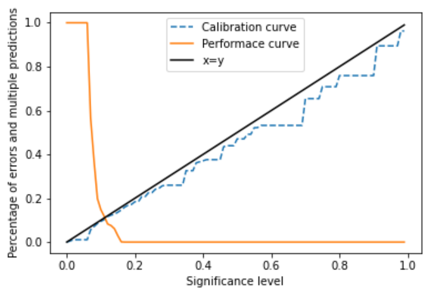


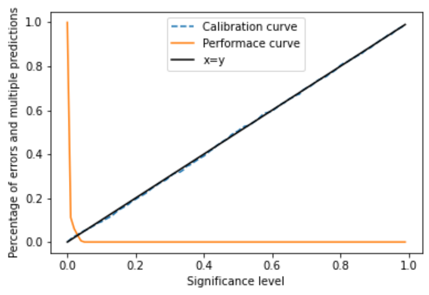
Figure 19: Calibration and performance curves for 1-Nearest neighbour TCP.



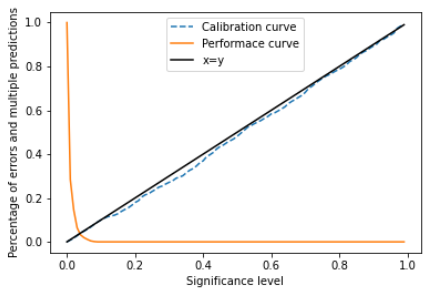
(a) Support vector machine



(b) Decision tree



(c) Neural network



(d) Random forest

Figure 20: Calibration and performance curves for different ICP.

The next experiment was to show how confidence and/or credibility values affected the ICPs in their predictive accuracy. It would be expected that predictions with low confidence and/or credibility would be incorrect.

Figures 21 and 22 shows how the confidence and credibility affect different ICPs. They also show exactly . The general trend is similar in all figures but each have interesting differences. The decision tree ICP in figure 22b produces predictions with the lowest confidence with some predictions having around 93% confidence. This could suggest that the decision tree ICP may perform the most poorly. The 1-nearest neighbour and Neural network ICPs in figures 21 and 22c both have quite distinguishable points in the trend where they make correct and incorrect predictions. Similarly for both, for confidence below 0.98 and/or credibility below 0.1 they both make a lot of incorrect predictions. Conversely for the SVM and random forest ICP in figures 22a and 22d, these points are a lot less distinguishable.

Most importantly, these plots show that it can be expected that any prediction with credibility less than 0.1 is incorrect. For confidence levels, the threshold in which incorrect predictions are made can vary between different ICPs. These results demonstrate a good rule of thumb that is consistent between ICPs in general: samples with credibility less than 0.1 should be investigated further. In the conformal anomaly detection section, examples with low credibility will be examined.

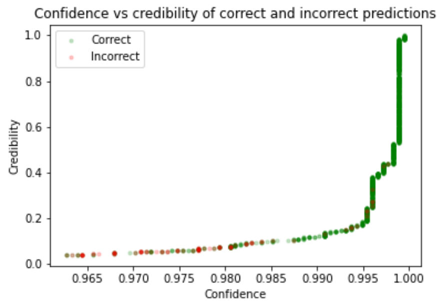
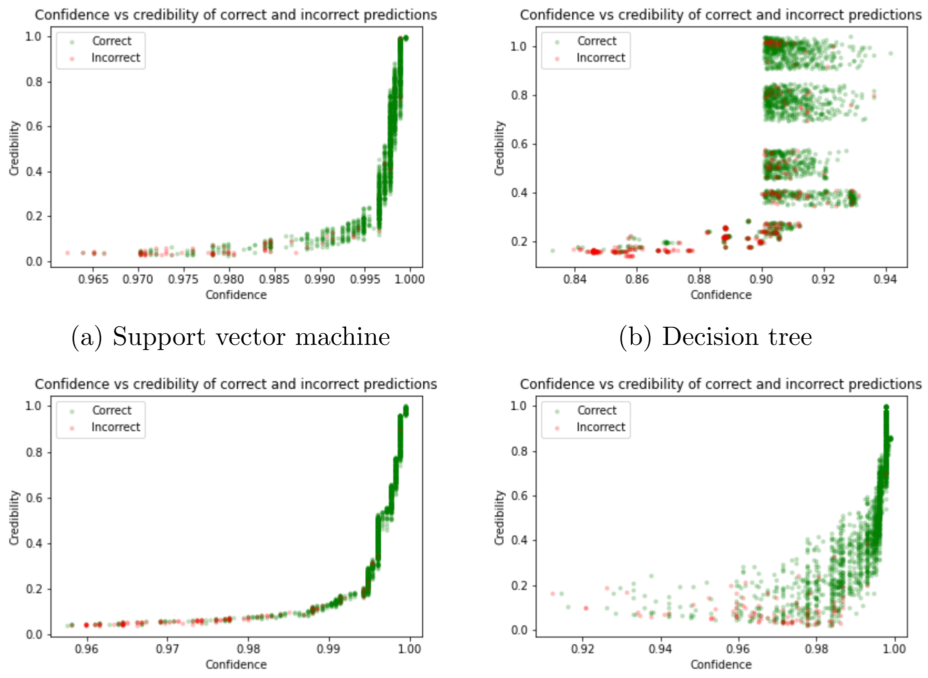


Figure 21: 1-Nearest neighbour



(a) Support vector machine

(b) Decision tree

(c) Neural network

(d) Random forest

Figure 22: Confidence vs credibility for different ICP on USPS dataset.

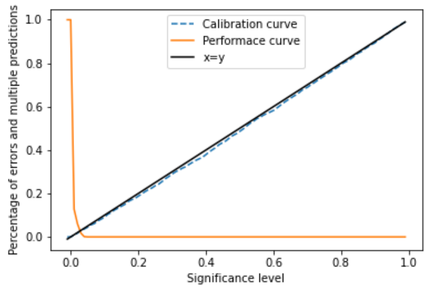
4.1.2 Conformal predictors with MNIST

Inductive conformal predictors Since the MNIST dataset is so large, only ICPs have been used in this section since the datasets are too large for the TCP and OCP used in the previous section. There were also some issues with the computing power required for some models; running the 1-nearest neighbour model would issue an error saying that there was not enough RAM to run this. To fix this a smaller subset of the MNIST dataset was used. The new dataset consists of 30,000 samples, with a training, validation and test set of sizes 20,000, 5,000 and 5,000 respectively. The neural network ICP was removed since there were some coding issues and the ICP was not valid. The 1-nearest neighbour model uses the nonconformity used in equation 3 and Euclidean distance as a distance measure. The SKLearn models use the nonconformity measure from equation 7. The following are the models used using the SKLearn package:

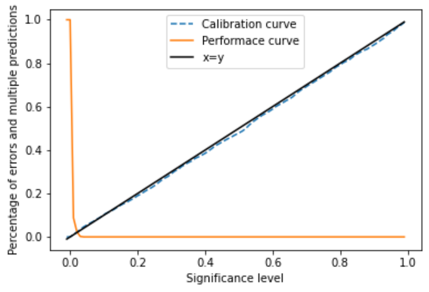
```
SVM_model = SVC(probability=True)
```

```
tree_model = tree.DecisionTreeClassifier(max_depth=10,
                                         max_leaf_nodes=100)
rf_model = RandomForestClassifier(max_depth=10)
```

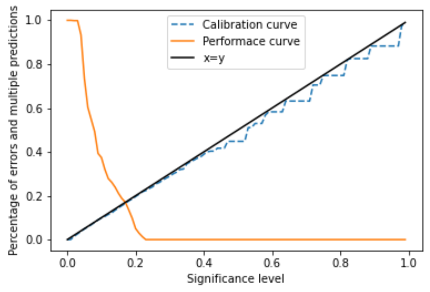
Similar to the TCP and ICPs applied to the USPS dataset, an experiment was carried out to investigate what confidence and/or credibility values start producing incorrect predictions.



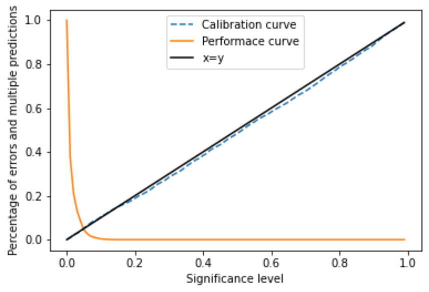
(a) 1-Nearest neighbour



(b) Support vector machine



(c) Decision tree

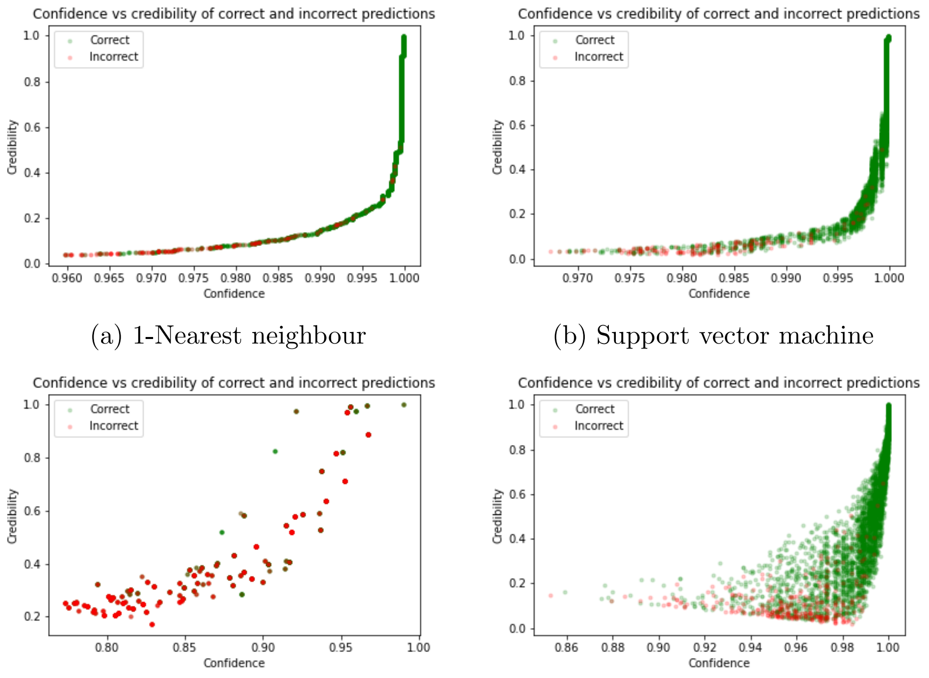


(d) Random forest

Figure 23: Calibration and performance curves for different ICP on the MNIST dataset.

It can be seen in figure 23 that each of the model is valid since the calibration curve sits just under the diagonal. The calibration curve for the decision tree model in 23c is not as well calibrated for larger significance levels but still it shows the bound $Pr(y \notin \Gamma^{\epsilon}) \leq \epsilon$ holds. The performance curve for the decision tree model shows that it performs much more poorly compared to the other models. For significance level $\epsilon = 0.2$ it still makes multiple predictions. This poor performance is further supported in figure 24c since the plot shows that the decision tree based ICP makes really poor predictions even at high levels of confidence and credibility. It can also been seen that both 1-nearest neighbour and SVM perform very well. The minimum confidence produced by both models exceeds 0.96.

These results further reinforce the suggestion that prediction that produces a credibility less than 0.1 should be investigated further.



(a) 1-Nearest neighbour

(b) Support vector machine

(c) Decision tree

(d) Random forest

Figure 24: Confidence vs credibility for different ICP on MNIST dataset.

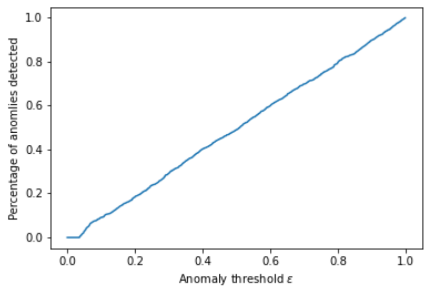
4.2 Conformal anomaly detection

Once a CP has been shown to be valid, applying it to anomaly detection is quite simple. - From this point forwards ϵ is now used to refer to the anomaly detection threshold. Once the p-values for each of the postulated labels has been calculated for a sample, if all the p-values are less than ϵ then that sample is considered an anomaly. Since credibility is the largest p-value, this is equivalent to stating that if the credibility is less than ϵ then that sample is an anomaly. As the anomaly threshold decreases then it is expected that the samples that are considered anomalies should get stranger. The aim of this section is to: first, show the bound in proposition 2.1; secondly, to explore samples that produce predictions with low credibility, and; finally, to investigate what a good anomaly threshold is.

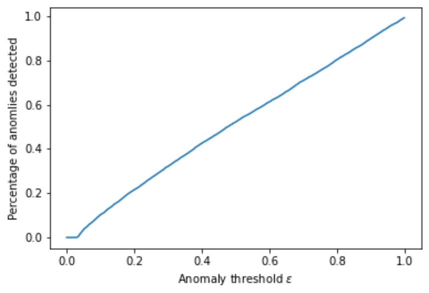
Since ICADs are computationally efficient they have been used for anomaly detection. ICADs are simply an extension to ICPs. When an ICAD is referred to it simply means that a specific ICP has been applied to anomaly detection. For example, the nearest neighbour ICAD uses the nearest neighbour ICP but specifically for anomaly detection.

4.2.1 Conformal anomaly detection with USPS

The CPs used in section 4.1.1 have all been used for anomaly detection on the USPS dataset. Proposition 2.1 implies that for a CAD, the frequency of anomalies detected should be less than or approximately equal to the anomaly threshold ϵ. This is analogous to the calibration curves used earlier in the project. The bound for proposition 2.1 can clearly be seen to hold in figures 25 and 26. Each curve is almost a perfect diagonal apart from the decision tree. An almost perfect diagonal suggest a CAD is well calibrated, suggesting that the decision tree CAD will perform poorly.



(a) 1-nearest neighbour TCP.



(b) 1-nearest neighbour OCP.

Figure 25: Percentage of anomalies detected for different significance levels ϵ.

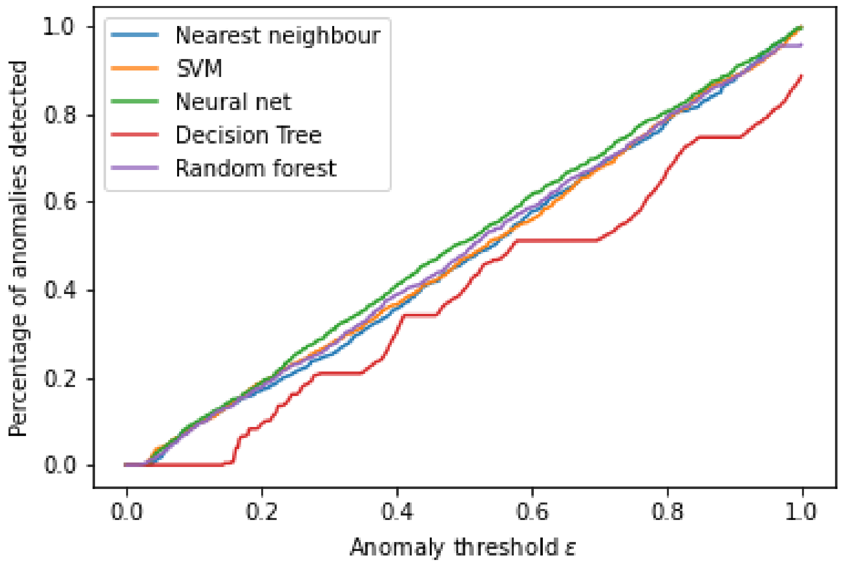
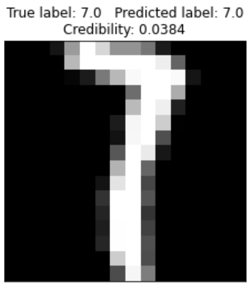
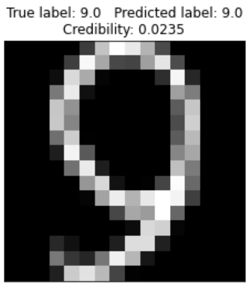


Figure 26: Number of anomalies detected for significance levels ϵ for different ICPs.

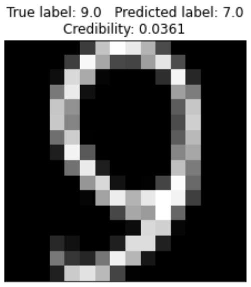
Investigating samples with low credibility A small credibility implies that all p-values are each postulated label are small. Investigating exactly what objects produce predictions with small credibility can show two things. The first is that the objects could potentially be a new class since it does not conform to the training set. If the object is not a new class then it can show what types of objects a CP performs poorly on. A sample that produces the lowest credibility could be considered as the most abnormal sample. It is interesting to see what different CAPs consider the most abnormal sample. The samples with the lowest credibility from different CAPs are shown in 27. It is interesting to see that for 27a and 27b the correct predictions were made even with extremely low credibility. Conversely, the neural network, decision tree and random forest ICADs make incorrect predictions which is to be expected with the low credibility. To the human eye each of the samples are still readable which for extremely low credibility would be unlikely. The samples should look “strange”. The “strangest” would probably be in figure 27d. It looks like a mirrored ‘6’ but the decision tree predicted ‘0’.



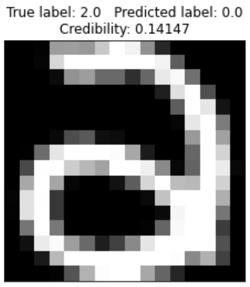
(a) 1-Nearest neighbour



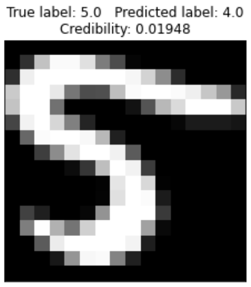
(b) Support vector machine



(c) Neural network

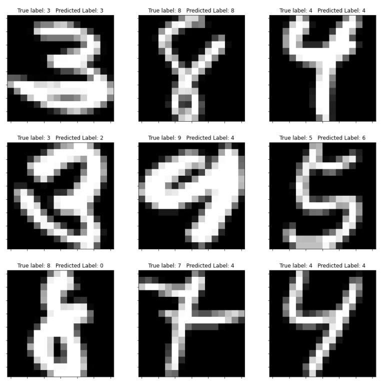


(d) Decision tree

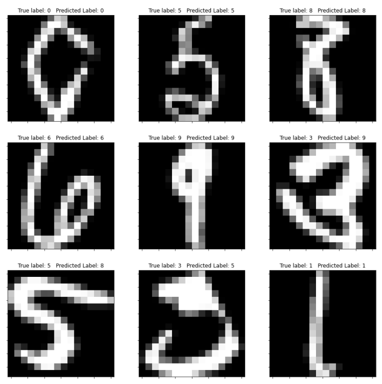


(e) Random forest

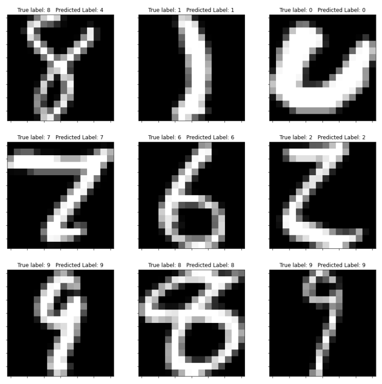
Figure 27: The lowest credibility sample for different CADs on the USPS dataset.


True label: 3 Predicted Label: 3
True label: 8 Predicted Label: 8
True label: 4 Predicted Label: 4
True label: 3 Predicted Label: 2
True label: 9 Predicted Label: 4
True label: 5 Predicted Label: 6
True label: 8 Predicted Label: 0
True label: 7 Predicted Label: 4
True label: 4 Predicted Label: 4

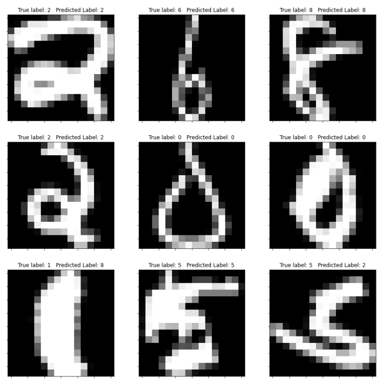
(a) 1-nearest neighbour


True label: 0 Predicted Label: 0
True label: 5 Predicted Label: 5
True label: 8 Predicted Label: 8
True label: 6 Predicted Label: 6
True label: 9 Predicted Label: 9
True label: 3 Predicted Label: 9
True label: 5 Predicted Label: 8
True label: 3 Predicted Label: 5
True label: 1 Predicted Label: 1

(b) SVM


True label: 8 Predicted Label: 4
True label: 1 Predicted Label: 1
True label: 0 Predicted Label: 0
True label: 7 Predicted Label: 7
True label: 6 Predicted Label: 6
True label: 2 Predicted Label: 2
True label: 9 Predicted Label: 9
True label: 8 Predicted Label: 8
True label: 9 Predicted Label: 9

(c) Neural network


True label: 2 Predicted Label: 2
True label: 6 Predicted Label: 6
True label: 8 Predicted Label: 8
True label: 2 Predicted Label: 2
True label: 0 Predicted Label: 0
True label: 0 Predicted Label: 0
True label: 1 Predicted Label: 8
True label: 5 Predicted Label: 5
True label: 5 Predicted Label: 2

(d) Random forest

Figure 28: Random samples that produce predictions with credibility less than 0.1 for different ICADs.

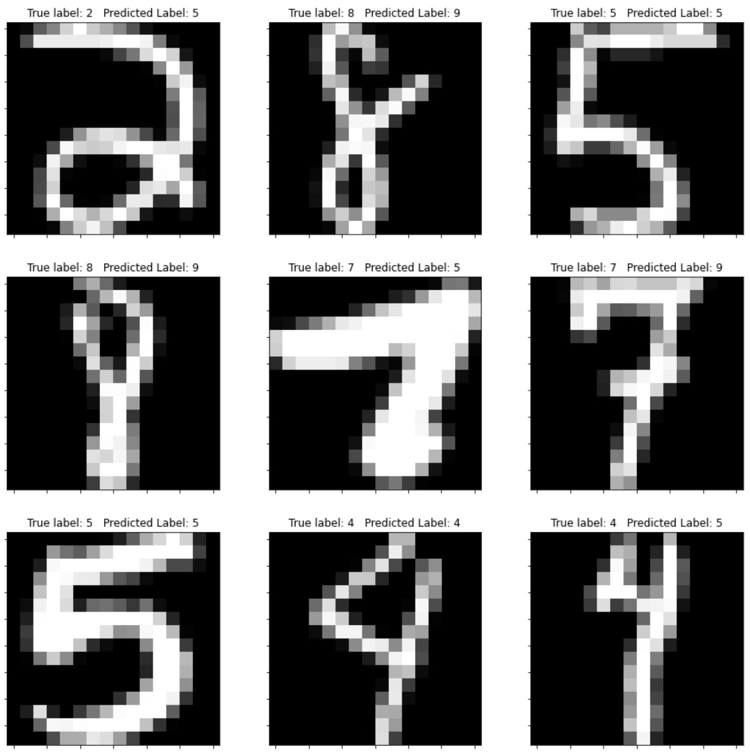
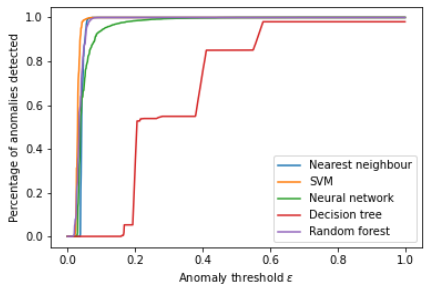

True label: 2 Predicted Label: 5
True label: 8 Predicted Label: 9
True label: 5 Predicted Label: 5
True label: 8 Predicted Label: 9
True label: 7 Predicted Label: 5
True label: 7 Predicted Label: 9
True label: 5 Predicted Label: 5
True label: 4 Predicted Label: 4
True label: 4 Predicted Label: 5

Figure 29: Credibility between 0 and 0.2 for decision tree

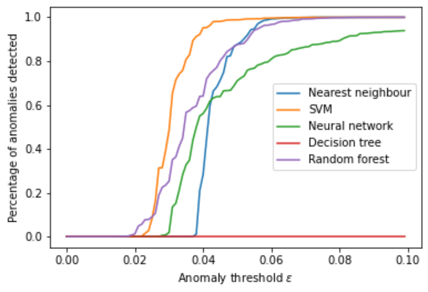
Since looking at the lowest credibility prediction did not provide too much insight, a random sample of 9 predictions from each ICAD with small credibilities have been shown in figures 28 and 29. Since the decision tree did not produce any predictions with credibilities less than 0.1 then a random sample of predictions with credibilities less than 0.2 have been shown in figure 29. For 1-nearest neighbour, there are a few odd looking samples. Numbers start to get shown where they look partially incomplete. There are two examples of the number '8' where the tops are open from where a pen ends short. In the center there is a number that looks like a '9' or a poorly written '5'. The SVM identifies stranger samples with the top left looking more like a rhombus or the letter ‘C’ than the number ‘0’. The neural network also identifies some strange objects. The top right object us probably the strangest object out of all the objects. It looks a lot like the letter ‘U’. There are also a couple of samples that look more like a ‘Z’ than a ‘2’. Again, there are similar results for the random forest ICAD. It can be seen that the images shown here are a lot more strange compared to the images in figure 9 which had examples with really high credibility.

Experiment with anomalous dataset An experiment was carried out using the synthetic dataset. Each of the inductive conformal anomaly detectors (ICADs) was trained on the USPS training set. The validation set used was also the USPS validation set. The only difference was the test set used the synthetic dataset. This synthetic dataset consists of noise. The idea was that this test set contained objects that are not from the same distribution as the USPS numbers. Since they are not from the same distribution, they are considered anomalies. Then the rate of detection of these anomalies can be used to compare the performances between different ICADs.

Figure 30 shows the results to this experiment. For each anomaly threshold the frequency of anomalies detected is plotted. The decision tree ICAD has awful performance in detecting the anomalies. This is to be expected since it performed poorly as an ICP too. What is more interesting is how well all the other ICADs perform. From an anomaly threshold of around 0.02, each of the ICADs excluding decision tree start detecting anomalies and at 0.04 the majority of the anomalies have been detected. Though figure 31 shows the same result, a histogram provides another good visual aid in seeing the credibility values given to each anomaly. Apart from the decision tree ICAD, the vast majority of the anomalies detected had a credibility less than 0.1. This provides the strongest evidence suggesting that the best anomaly threshold is 0.1.



(a)



(b) Figure 30a magnified on x-axis

Figure 30: Percentage of anomalies classified by different ICADs for different anomaly thresholds ϵ.

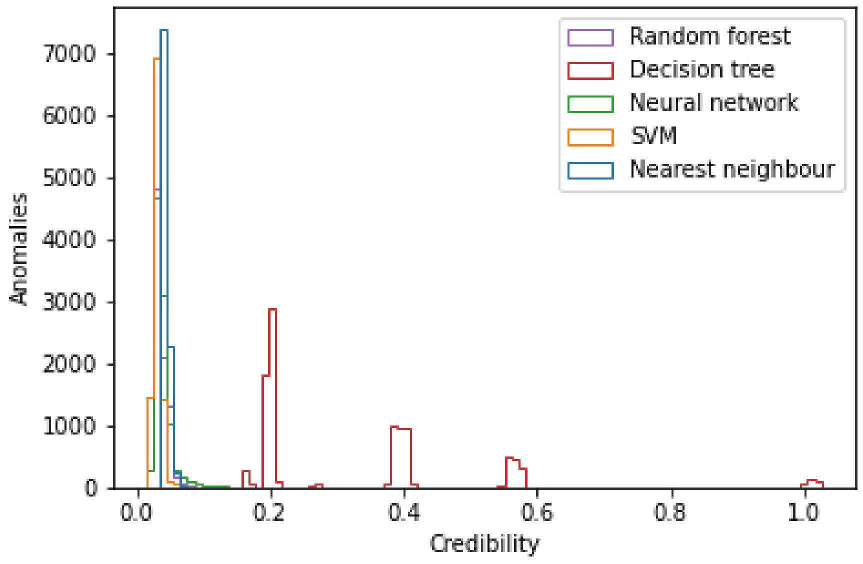


Figure 31: Histogram showing the credibilities of anomalies for different ICAPs.

4.2.2 Conformal anomaly detection with MNIST and EMNIST letters

Just as was done with the ICPs on the MNIST dataset, the ICADs used in this section used the smaller dataset consisting of 30,000 samples, with a training, validation and test set of sizes 20,000, 5,000 and 5,000 respectively.

In figure 32, the nearest neighbour, SVM and random forrest ICADs are well calibrated. Similar to when using the USPS dataset, the decision tree ICAD is poorly calibrated yet the bound in proposition 2.1 still holds.

Figures 34a to 34d show examples with extremely low credibility for each of the ICPs. For each of the ICPs apart from the decision tree ICP, a few strange samples with low credibility can be seen. Some of the samples have correct predictions from the ICP even with a very low credibility value. There are many examples shown that would even be difficult for a human to label and yet some have still had the correct label predicted.

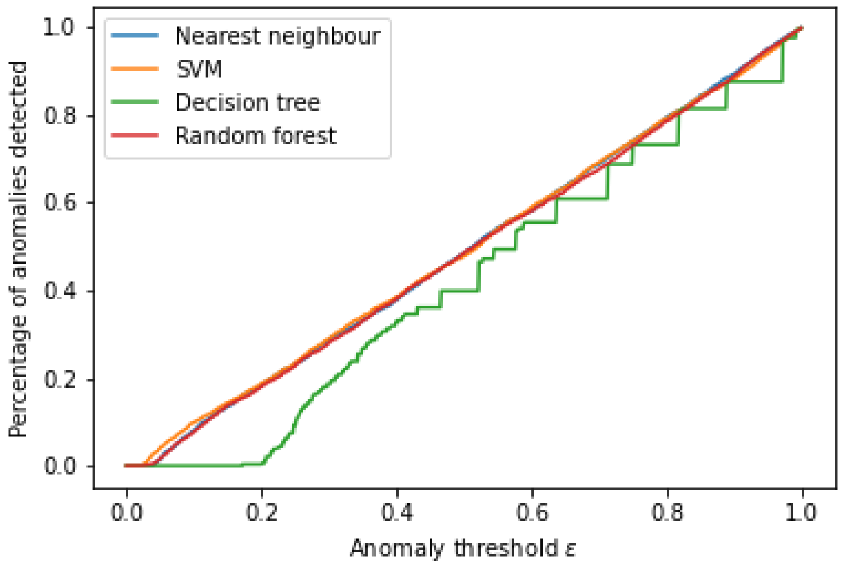
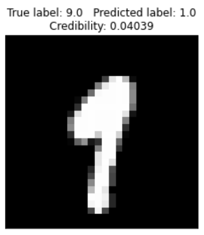
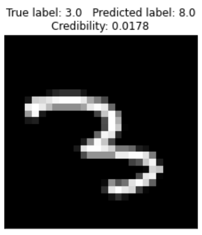


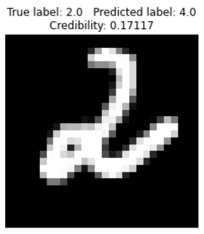
Figure 32: Frequency of anomalies detected for anomaly thresholds ϵ for different ICADs.



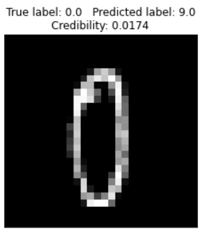
(a) 1-Nearest neighbour



(b) Support vector machine



(c) Decision tree



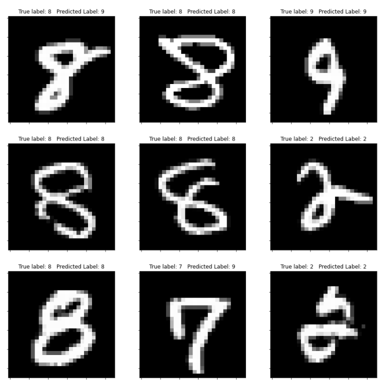
(d) Random forest

Figure 33: The lowest credibility sample for different CAPs on the USPS dataset.

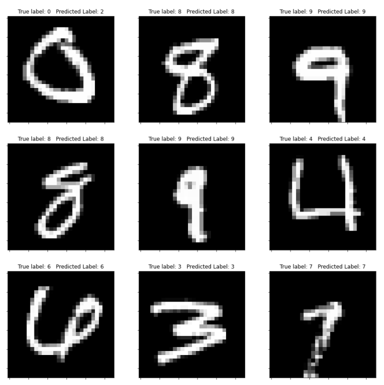
It can be seen in 33 the objects that produce the predictions with the smallest credibility for each ICAD. Each of the predictions are incorrect. The most interesting sample is the one produced by the 1-nearest neighbour ICAD in figure 33a. It looks a lot like the number '1' but the true label is in fact '9'.

Just as was done for USPS dataset, 9 random objects that produce predictions with low credibility for each ICAD can be seen in figure 34.

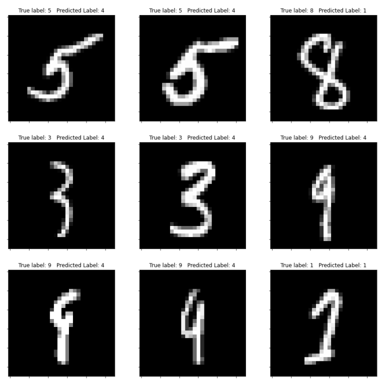
In figures 34a and 34b, a selection of the lowest credibility predictions made by the 1-nearest neighbour and SVM ICADs can be observed. It can be seen that all but one of the predictions for each is correct. Even though most of predictions are correct a lot of the samples can definitely be considered abnormal. On the other hand, in figure **??** a random selection of the low credibility predictions made by the decision tree ICAD can be observed. Of these 9 examples, the ICAD only correctly predicted the true label for one of the samples. Finally, in Figure 34d, a selection of the lowest credibility predictions made by the random forest ICAD can be observed. The ICAD has correctly predicted the true label of 3 (one third) of these samples. What is interesting is that for all the ICADs, there seem to be quite a few correct predictions even with samples a person might struggle identifying.


True label: 8 Predicted Label: 9
True label: 8 Predicted Label: 8
True label: 9 Predicted Label: 9
True label: 8 Predicted Label: 8
True label: 8 Predicted Label: 8
True label: 2 Predicted Label: 2
True label: 8 Predicted Label: 8
True label: 7 Predicted Label: 9
True label: 2 Predicted Label: 2

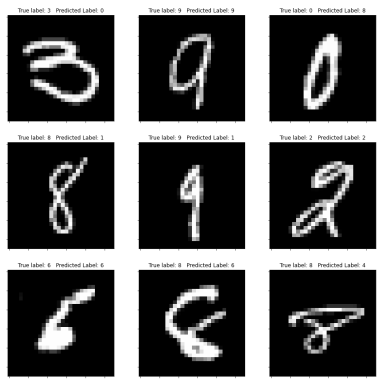
(a) 1-nearest neighbour


True label: 0 Predicted Label: 2
True label: 8 Predicted Label: 8
True label: 9 Predicted Label: 9
True label: 8 Predicted Label: 8
True label: 9 Predicted Label: 9
True label: 4 Predicted Label: 4
True label: 6 Predicted Label: 6
True label: 3 Predicted Label: 3
True label: 7 Predicted Label: 7

(b) SVM


True label: 5 Predicted Label: 4
True label: 5 Predicted Label: 4
True label: 8 Predicted Label: 1
True label: 3 Predicted Label: 4
True label: 3 Predicted Label: 4
True label: 9 Predicted Label: 4
True label: 9 Predicted Label: 4
True label: 9 Predicted Label: 4
True label: 1 Predicted Label: 1

(c) Decision tree


True label: 3 Predicted Label: 0
True label: 9 Predicted Label: 9
True label: 0 Predicted Label: 8
True label: 8 Predicted Label: 1
True label: 9 Predicted Label: 1
True label: 2 Predicted Label: 2
True label: 6 Predicted Label: 6
True label: 8 Predicted Label: 6
True label: 8 Predicted Label: 4

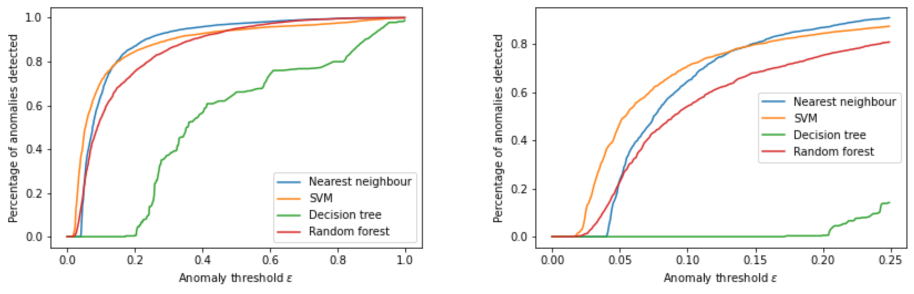
(d) Random forest

Figure 34: Random samples that produce predictions with credibility less than 0.1 (0.2 for decision tree) for different ICADs.

Experiment with EMNIST letters dataset Similar to the anomaly detection with the USPS and synthetic dataset, an experiment to compare the performances of the ICADs was carried out. Each ICAD used the training and validation set from the shortened MNIST dataset. The test set used contained 10,000 random samples from the EMNIST letters dataset.

In the previous section noise was used to test anomaly detection. Since the EMNIST dataset is an extension of the MNIST dataset, the objects were collected in a similar fashion and some can look similar. For example, the number "1" and letter "l" can look almost identical at times. Anomalies are not always so distinct in the real world as they were in the previous section using noise. The aim of this experiment was to test anomaly prediction using a more "real-world" example, thus producing results with a higher ecological validity than the previous experiment.

Figure 35 shows the frequency of anomalies detected for different anomaly thresholds ϵ. It is quite clear how poorly the decision tree ICAD performs. The SVM ICAD detects anomalies at the smallest anomaly threshold, followed closely by the random forest ICAD. The SVM ICAD performs the best until an anomaly threshold of around 0.14 where the nearest neighbour ICAD surpasses. Figure 36 shows how close the performances of nearest neighbour, SVM, and random forest are. SVM looks to perform best but only by a small measure of performance. These results are similar to what was found when using the USPS dataset.



(a) (b) Figure 35a magnified on x-axis

Figure 35: Percentage of EMNIST anomalies classified by different ICADs for different anomaly thresholds ϵ.

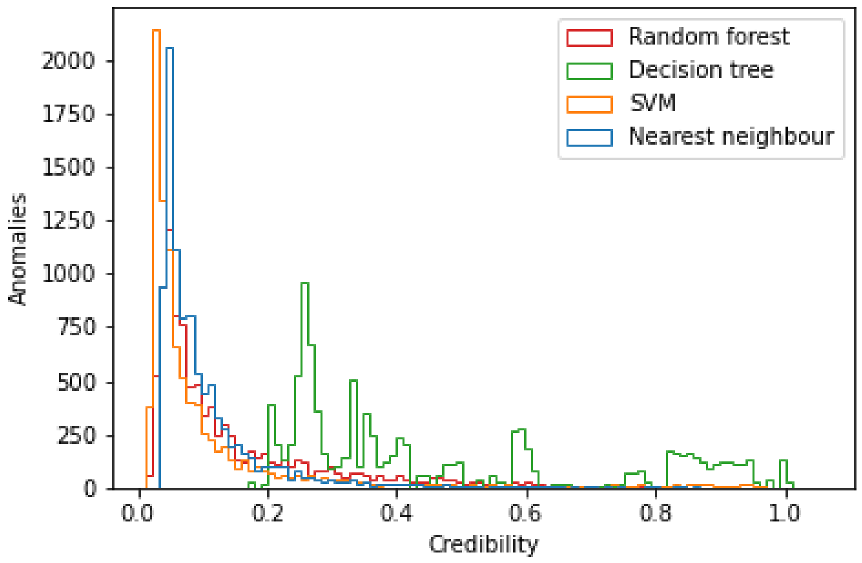


Figure 36: Histogram showing the credibilities of anomalies for different ICAPs.

5 Conclusions and further work

Conclusion It was found, for both the USPS and MNIST datasets, that ICPs and ICADs built upon 1-nearest neighbour, SVM, and random forest all had very similar performances in predictive efficiency and anomaly detection. There was quite a difference in computational efficiency with random forest coming out on top. This is an interesting result as, in recent times, there is an enhanced focus on more complex, or "sophisticated", ML models such as neural networks. However, as shown when using the USPS dataset such models did not significantly outperform than simple models. In fact the neural network had worse predictive performance than most of the other CPs used. It was shown that when a CP has poor predictive efficiency, it was also poor when applied to anomaly detection. This was shown with the decision tree ICP. If a CP was shown to perform quite well on a dataset then it was consistently shown that predictions with credibility less than 0.1 tended to be incorrect or produced from "strange" objects or anomalies. This supports findings of previous work where samples that produce predictions with credibility less than 0.1 tend to be non representative of the training set.

Future work Projects such as these are always tight when it comes to time. There are always new ideas that arise throughout but could not be trialled since time is a precious commodity. If I were to add to this project here are some ideas I personal would look into:

- More testing of the performance of a CAD, i.e. to test the false alarm rate which was not tested in this project.
- It would be interesting to see if the conclusions reached in this project were consistent across different datasets since the USPS and MNIST datasets are very similar. This project also only touches upon classification problems, but CPs and CADs can be applied to regression problems too.
- This project made the assumption of randomness, which was not tested in this project. The assumption of randomness can be tested using martingales and future work should consider testing this to provide greater confidence in the reliability and validity of results.
- Future work could also look to build upon this project by comparing the conformal anomaly performance of other popular anomaly detection methods and the CPs looked at in this project.

6 Running the code

Supplied with this project is a file called *Conformal Predictors - Final.ipynb*. It supplies all the information to run all the code used including the links to the datasets, and the code for CPs and the plots shown. Simply open the Jupyter notebook file and click "Kernel", then click "Restart & Run All". Since it runs all the CPs it can take a little while to complete running.

7 Professional issues

CPs can be considered as a new ML algorithm, but just like any other ML algorithm, careful considerations need to be made when applying them to real-world problems. ML algorithms offer quick solutions to real-world problems, and as such there may be temptation for people to rely heavily on these before properly testing the validity of their application to a given problem or situation. This project has shown how CPs can be used to detect anomalous images. It has demonstrated this in principle, with very basic images. Should this technology be used for wider applications, further testing should be conducted to assess the accuracy and validity of any such applications. In this project there was always an assumption of randomness. But what happens if this assumption is ill founded? There are methods to test the IID assumption but it was not applied in this project. Therefore it is important to remember this project was built upon an assumption. For predicting numbers this doesn't really impose a moral dilemma if the assumption fails. If the techniques used in this project were used and applied to problems where the predictions can have much stronger implications, such as being used as a diagnosis tool, then there would be quite severe consequences if these assumptions fail. Hence, the assumptions should be rigorously tested.

Another issue to consider is the potential impact of such technology should it be developed further down the line. This project looks at detecting anomalies in images but anomaly detection has also been applied to detecting crime through CCTV. They work by monitoring an area and the idea is when something against the norm happens it flags up. The issue with applying ML to such tasks is that any predictive model is only as good as the data its fed with. Problems arise as soon as the training data is biased. Now consider that a CCTV camera is monitoring a shop that is in a predominantly white neighbourhood. Could the system learn to consider that white people in a shop is normal but if a group of people of a different ethnicity entered the shop, they could get flagged up on the system? Could an anomaly discriminate against people in wheelchairs? Or conversely, suppose there was a crime that happened so regularly that the anomaly detector stopped reporting it. These are all realistic problems. Since it would be really difficult (almost impossible) detecting when a anomaly detector has learnt these prejudices, then it would be imperative that no final decision of enforcement should be made by the model. Such considerations should always be made when progressing and developing any applications of ML algorithms, as considerations of dangerous applications should be taken prior to developing something.

A ground-breaking application for CPs or anomaly detection is in medical diagnosis using imagery. There are many advances through diagnosing certain cancers such as breast cancer. These can detect a cancer from an image in moments whereas it can take a fully trained doctor tens of minutes. This can free up a considerable amount of time for a doctor to be applying their highly valuable skills elsewhere. As powerful as these tools are there comes a considerable risk. No matter how accurate the predictive model is, there is always a risk of an incorrect diagnosis. Even for a CP using a significance level of $\epsilon = 0.00001$ it would be expected to be wrong for 1 in 100,000 predictions. So imagine a tool giving a diagnosis with this extremely low probability. That one incorrect diagnosis still has a person at the end of it and can impact them greatly. Anomaly detectors are susceptible to false positive (false alarms) and false positives. Using the breast cancer diagnosis as an example, a false positive in this case would be misdiagnosing a patient with breast cancer. Hopefully this patient would be further examined by a doctor to conform or reject this diagnosis. A more difficult problem is with false negative since anomaly detectors only flag up objects that they consider anomalies. In this cancer scenario, the anomaly detector has failed to detect that a patient does in fact have breast cancer. Since an anomaly detector only brings attention to those that it deems anomalies then false negatives are missed. These are the issues that must be considered when applying such tools.

These scenarios bring attention to key points. The first is that ML models must be rigorously tested so that the assumptions they are built upon hold true. The final point is that there needs to be some human intervention into final decisions. There are moral implications in wilfully trusting a ML model.

8 Self-assessment

This project was a huge learning curve from start to finish. Before this project, I had never attempted a project of this size since my BSc did not contain a dissertation. The project provided a great opportunity to learn lots of new skills and refine existing skills.

Strengths This project looks at CPs for making predictions and detecting anomalies. This is a relatively novel area in Machine Learning, and as such this project adds to the body of literature in this field. I made efforts to read as much of the literature as possible and to summarise the key findings as best I can. I think that the background section in this project summarises the key points of the literature as it applies to this project quite well. I applied ICPs to detect anomalies within datasets and took a new approach to exploring this application by using samples from two datasets to detect letters (anomalies) amongst numbers (the learning set). This was a good idea to test the ICPs in a more "real world" scenario, and I think this a strength of the experiment because it has good ecological validity due to the realistic nature of the anomaly detection tested. I wrote code to train and test each of the CPs used, and this ran well. There was some issues with making things run properly and I wrote code to address issues that arose and also solved problems with things not running efficiently. The code ran well, making predictions about the labels for the sample datasets and then also calculating the accuracy of these predictions. I also wrote code to produce plots displaying all of the results, and I am impressed with the qualities of these.

Weaknesses Though I had a good understanding of CPs, I believe you need excellent foundations on CPs to apply them to anomaly detection. For this reason, a good part of my time was taken really refining my knowledge of CPs. I spent a lot more time on this than I had anticipated would be required. This may have been helped if I had started going through my reading material a lot sooner, but I did not have opportunity to commence this learning earlier than I did. I feel like I could have potentially looked to include more references in this project, as there was material I read which did not feel relevant to include but this has resulted in a short reference list. Another point was that it was brand new for me to do a project with little structure. There is a general structure provided but you never quite know where your project will end up. Because of this, I struggled at first adapting to the unknown. There were also times where I spent too much time trying to solve a specific problem. For example, I wasted a good two weeks trying to get my OCP to not only work, but to be efficient enough that it could be run on the larger MNIST dataset. I say wasted because when it comes to projects like this, time comes at a premium. This was time that could have been experimenting with things that I touched upon in further work. There were also a lot of unfortunate circumstances within my family and personal life that happened during the period of this project. These issues greatly affected my work output for large parts of this period. I did my best trying to adapt and continue working through with my project, and I am happy and proud with what I have managed to achieve, but I believe had these issues not arisen my project would have been stronger than it is.

Skills When reflecting on the strengths and weaknesses of this project, I reflected on the work completed and became aware of my personal development and new skills that I have gained. Doing this project has helped me to learn and develop as an academic. There are a lot of positives to take from this. For starters, although I am doing a computer science MSc, my programming knowledge is still relatively new having come to this from a Mathematics background. I am a much stronger programmer in Python now than I was before I started this project. I always aim to make concise, efficient and readable code. This project challenged me to improve the two latter points especially. Apart from the problems I had with the OCP, my conformal predictors ran quite quickly which I am pleased with. This then gave me the ability to experiment with the larger MNIST dataset. I also decided to type up the project in LaTeX. My proficiency in LaTeX has greatly improved as a consequence. The project also gave me the confidence to approach another project of this magnitude in the future. I've learned a lot from this experience, and believe I will be better equipped to take on something of this nature in the future. My biggest personal point I can take from this though is the knowledge I now possess about CPs. CPs are a relatively new tool, emerging from academics at Royal Holloway. I now possess a lot of knowledge about an incredibly useful and powerful tool that I can share with others within the ML field and make use of in my future career.

References

[1] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: an extension of mnist to handwritten letters, 2017.

[2] Alexander Gammerman and Vladimir Vovk. Hedging predictions in machine learning. *The Computer Journal*, 50(2):151–163, 2007.

[3] Douglas M Hawkins. *Identification of outliers*, volume 11. Springer, 1980.

[4] J. J. Hull. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554, 1994.

[5] Rikard Laxhammar. Anomaly detection for sea surveillance. In *2008 11th international conference on information fusion*, pages 1–8. IEEE, 2008.

[6] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[8] Vladimir Vapnik. *The nature of statistical learning theory.* Springer science & business media, 2013.

[9] Vladimir N Vapnik. An overview of statistical learning theory. *IEEE transactions on neural networks*, 10(5):988–999, 1999.

[10] Vladimir Vovk. Lecture notes on conformal prediction, September 2021.

[11] Vladimir Vovk, Alex Gammerman, and Glenn Shafer. *Algorithmic learning in a random world.* Springer Science & Business Media, 2005.

9 Appendix

```
####################################
predictors.py
####################################

import numpy as np
from sklearn.metrics import pairwise_distances
from tqdm.notebook import tqdm

def rank(array,num):
        """Find rank of number against an array"""
        # much more efficient way
        return np.sum(array>=num)+1

def p_values(post_conform_scores, valid_conform_scores):
    '''Calculates the smooth p-value of a non-conformity score against an array
        of nonconformity scores'''
    # preload to store values
    table = np.ones(post_conform_scores.shape)
    for i, row in enumerate(post_conform_scores):
        for label, conform_score in enumerate(row):
            # get rank
            smooth = np.random.uniform()*np.sum(valid_conform_scores==conform_score)
            table[i,label] = rank(valid_conform_scores, conform_score) + smooth
     # no need for +1 since count starts at 1
    # divide table by size of validation set + 1 (to include the postulated sample)
    table = table/(len(valid_conform_scores)+1)
    return table

### Nearest neighbour conformal predictors

def Conform_diff_same(distances, label, y_labels):
    """
    Applies the conformal measure diffdist/samedist on distances
    Input:  distances - is an nx1 array containing distances from training set
    Output: diffdist/samedist
            where diffdist is the distance to the nearest sample of a
            different class
            and samedist is the distance to the nearest sample of the same class
    """
    tempsame = distances[y_labels == label]
    # create array containing all distances against samples with same class
    tempdiff = distances[y_labels != label]
    # create array containing all distances against samples with diff class
    samedist = np.min(tempsame)
    # grab min dist from training samples with same label
    diffdist = np.min(tempdiff)
    # grab min dist from training samples with diff label
    if samedist == 0 and diffdist == 0:
        return 0
    elif samedist == 0 and diffdist !=0:
        return np.inf
    else:
        return samedist/diffdist

def NNpvalues_full(X_train, y_train, X_test, y_test):
    '''Transductive conformal predictor using 1-nearest neighbour'''
    labels = np.unique(y_train) # get labels of training data
    rows = len(y_train) # number of training samples
    train_conf_scores = np.zeros(rows)
    # preload array
    p_values = np.zeros((X_test.shape[0], len(labels)))
    # preload matrix of p-values
    indices = np.arange(rows)
    # to easily change diagonal of distances
    train_distances = np.array(pairwise_distances(X_train, n_jobs=-1))
    # get training distances
    train_distances[indices,indices] = np.inf
    # make the diagonal infinity because they will all be 0
    #and we don't want to consider them
    test_distances = np.array(pairwise_distances(X_test, X_train, n_jobs=-1))
    # get distances against training samples for each test sample
    # get conformity scores of training samples
    # don't need to cycle through each label for training
    for i, dist in tqdm(enumerate(train_distances)):
        train_conf_scores[i] = Conform_diff_same(dist, y_train[i], y_train)
        # get conformity score for sample given it's correct label
    # get conformity scores of test samples
    for i, dist in tqdm(enumerate(test_distances)):
        # cycle through each postulated label
        for j, label in enumerate(labels):
            conf_score = Conform_diff_same(dist, j, y_train)
            # calculate conformity score
            smooth = np.random.uniform()*np.sum(train_conf_scores==conf_score)
            p_values[i,j] = rank(train_conf_scores, conf_score)+smooth
            # calculate the rank here
    p_values = p_values/(len(y_train)+1)
    # need to divide by (# of training sample +1) to update to p-values
    return p_values

def NNpvalues_online(X_data, y_data, labels):
    '''On-line conformal predictor using 1-nearest neighbour'''
    rows = len(y_data) # get total number of samples
    train_conf_scores = np.zeros(rows) # preload array
    p_values = np.zeros((X_data.shape[0], len(labels)))
    # preload matrix
    print('Step 1: Getting distances')
    all_distances = np.array(pairwise_distances(X_data, n_jobs=-1))
    # get all distances now so we don't have to repeat
    all_distances[np.arange(rows),np.arange(rows)] = np.inf
    # change diagonals to infinity since they will all be 0
    print('Completed')
    seen_labels = []
```

```
# as we see each label we will append to this since we
# can't calculate non-conformity scores if the label is unseen
print('Step 2: Getting non-conformity scores')
for k in tqdm(range(1,rows)):
# start at 1 since we want at least one training sample to begin with
    train_rows = k # number of training samples
    if y_data[k] not in seen_labels:
        seen_labels.append(y_data[k])
        # add new label to our list of labels
    # create new variables to make the code easier to read
    train_distances = all_distances[:k,:k]
    # our training samples include all distances up to kth
    # row and column
    train_labels = y_data[:k]
    # our training labels
    train_conf_scores = np.zeros(train_rows)
    # preload array to save non-conformity scores
    test_distances = all_distances[:k+1,k]
    # our test distances have one extra row
    for j, dist in enumerate(train_distances):
    # cycle through each training sample
        if y_data[j] in seen_labels and len(seen_labels)>1:
        # to make sure we have at least one label the same
        # and one different
            train_conf_scores[j] = Conform_diff_same(dist,
                                                     train_labels[j],
                                                     train_labels)
            # get non-conformity score for correct label
    for label in seen_labels:
    # only need to cycle through seen labels
        if len(seen_labels)>1:
        # need at least one different label
            conf_score = Conform_diff_same(test_distances,
                                           label,
                                           y_data[:k+1])
            # get non-conformity score
            smooth = np.random.uniform()*
                            np.sum(train_conf_scores==conf_score)
            p_values[k,label] = (rank(train_conf_scores, conf_score)+smooth)
                /(len(train_labels)+1)
                # calculate p-values here since number of training
                # samples is continuously changing
    print('Completed')
    return p_values

def NNpvalues_induct(X_train, y_train, X_valid, y_valid, X_test, y_test):
    '''Inductive conformal predictor using 1-nearest neighbour'''
    labels = np.unique(y_train) # get labels of training data
    rows = len(y_train) # number of training samples
    valid_conf_scores = np.zeros(rows) # preload array
    p_values = np.zeros((X_test.shape[0], len(labels)))
    # preload matrix of p-values
    indices = np.arange(rows) # to easily change diagonal of distances
    valid_distances = np.array(pairwise_distances(X_valid, X_train, n_jobs=-1))
    # get training distances
    test_distances = np.array(pairwise_distances(X_test, X_train, n_jobs=-1))
    # get distances against training samples for each test sample
    # get conformity scores of validation samples
    # don't need to cycle through each label for training
    for i, dist in tqdm(enumerate(valid_distances)):
        valid_conf_scores[i] = Conform_diff_same(dist, y_valid[i], y_train)
        # get conformity score for sample given it's correct label
    # get conformity scores of test samples
    for i, dist in tqdm(enumerate(test_distances)):
        # cycle through each postulated label
        for j, label in enumerate(labels):
            conf_score = Conform_diff_same(dist, j, y_train)
            # calculate conformity score
            smooth = np.random.uniform()*np.sum(valid_conf_scores==conf_score)
            p_values[i,j] = rank(valid_conf_scores, conf_score)+smooth
            # calculate the rank here

    p_values = p_values/(len(y_valid)+1)
    # need to divide by (# of training sample +1) to update to p-values
    return p_values

### SKLearn inductive conformal predictors

def inductive_conform_predictor(X_train, y_train,
                                X_valid, y_valid,
                              X_test, y_test, model):
    '''Inductive CP for SKLearn models'''
    model.fit(X_train, y_train)
    valid_conform_scores = np.array(model.predict_proba(X_valid))
    valid_conform_scores = -1 * np.array([valid_conform_scores[i,label]
                               for i, label in enumerate(y_valid)])
    probs = model.predict_proba(X_test)
    post_conform_scores = -1 * probs
    p_vals = p_values(post_conform_scores,valid_conform_scores)
    return p_vals

### Evaluation of CPs

def conf_cred(p_values, y_test):
    '''
    Outputs nx3 table
    1st column contains true lable
    2nd column contains the point prediction
    3rd column contains the confidence of the prediction
    4th column contains the credibility of the prediction
    '''
    # sort the p_values with 2 largest at the end (efficient sort)
    idx = np.argpartition(p_values,-2, axis=1)
    # preload to store
    table = np.zeros((p_values.shape[0],3))
    for i, row in enumerate(p_values):
        point_pred = idx[i,-1] # get point prediction
        cred = p_values[i,idx[i,-1]] # get credibility value for prediction
        conf = 1-p_values[i,idx[i,-2]] # get confidence value for prediction
        table[i] = [point_pred, conf, cred]
    return np.column_stack((y_test,table))

### Anomaly detection

def conform_predictor_model(X_valid, y_valid, X_test, model):
    '''Calculates the p-values of test samples against a
       validation set on a pre trained model'''
    valid_conform_scores = np.array(model.predict_proba(X_valid))
    valid_conform_scores = -1 * np.array([valid_conform_scores[i,label]
                                      for i, label in enumerate(y_valid)])
    post_conform_scores = -1 * model.predict_proba(X_test)
    p_vals = p_values(post_conform_scores,valid_conform_scores)
    return p_vals


####################################
plots.py
####################################

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
import pandas as pd

def to_image(vector):
    '''Outputs vector of size n^2 into a nxn matrix form'''
    n = np.sqrt(len(vector))
    n = np.int(n)
    final = []
    for i in range(n):
        loc = i*n
        final.append(vector[loc:loc+n])
    return np.array(final)

def rand_samples(X_data, y_data, letters = False):
    '''Plots 3x3 grid with random images from X_data with their labels
    If X_data contains samples of letters then use letters=True
    '''
    num_samp = X_data.shape[0]
    samp = np.random.randint(1,num_samp, 9)
    fig, ax = plt.subplots(3,3,figsize = (15, 15))

    for i in range(3):
        for j in range(3):
            k = i*3+j
            fig = ax[i,j].figure
            image = to_image(X_data[samp[k]])
            if letters == False:
                ax[i,j].set_title('True label: '+str(int(y_data[samp[k]])))
            elif letters == True:
                image = np.rot90(np.fliplr(image),1)
                ax[i,j].set_title('True label: '+chr(ord('`')
                                      +int(y_data[samp[k]])))
            ax[i,j].imshow(image, cmap='gray')
            ax[i,j].set(xticklabels=[],yticklabels=[])

    return fig

def predict_set(p_values, sig_level):
    '''Returns each prediction set for each sample for a
       defined significance level'''
    final = []
    for sample in p_values:
        pred_set = np.where(sample>=sig_level)
        # find where each sample has a p-value greater than
        # epsilon for each label
        final.append(list(pred_set[0]))
    return final

def calibration_curve(y_test, p_values, sig_levels = np.arange(0,1,0.001),):
    ''' Plots a calibation and performance curve'''
    error_rates = np.zeros(len(sig_levels)) # to store error rates
    mult_rates = np.zeros(len(sig_levels))
    for i, sig in enumerate(sig_levels):
        temp_set = predict_set(p_values,sig)
        for loc, predictions in enumerate(temp_set):
            if y_test[loc] not in predictions:
                error_rates[i] += 1
            if len(predictions)>1:
                mult_rates[i] += 1
    error_rates = error_rates/len(y_test)
    mult_rates = mult_rates/len(y_test)
    fig = plt.figure()
    plt.plot(sig_levels,error_rates,
             linestyle='dashed', label = 'Calibration curve')
    plt.plot(sig_levels,mult_rates, label = 'Performace curve')
    plt.plot(sig_levels,sig_levels, color = 'k', label = 'x=y')
    plt.legend()
    plt.xlabel('Significance level')
    plt.ylabel('Percentage of errors and multiple predictions')
    plt.show()
    return fig

def corr_incorr_plot(p_values, y_test):
    '''Plots confidence vs credibility with correct
       and incorrect predictions'''
    predictions = np.argmax(p_values,axis=1)
    correct = p_values[predictions==y_test]
    # get p-values of correct predictions
    correct_conf_cred = np.partition(correct,-2, axis=1)
    wrong = p_values[predictions!=y_test]
    # get p-values of incorrect predictions
    wrong_conf_cred = np.partition(wrong,-2, axis=1)
    fig = plt.figure()
    plt.scatter(1 - correct_conf_cred[:,-2], correct_conf_cred[:,-1],
                color='g', alpha = 0.2, s=10)
    plt.scatter(1 - wrong_conf_cred[:,-2], wrong_conf_cred[:,-1],
                color='r', alpha=0.2, s=10)
    plt.legend(['Correct','Incorrect'])
    plt.title('Confidence vs credibility of correct and incorrect predictions')
    plt.xlabel('Confidence')
    plt.ylabel('Credibility')
    plt.show()
    return fig

### For OCP

def cumm_plot(p_values, y_data, sig_level):
    '''Plots the cummulative errors, empty and multple predictions for an OCP'''
    rows = len(y_data)
    err = 0
    mult = 0
    emp = 0
    err_store = np.zeros(rows)
    mult_store = np.zeros(rows)
    emp_store = np.zeros(rows)
    for i, sample in enumerate(p_values):
        predict_set = np.where(sample>=sig_level)[0]
        if y_data[i] not in predict_set:
            err += 1
```

```
                err_store[i] = err
            elif y_data[i] in predict_set:
                err_store[i] = err
            if predict_set.size==0:
                emp += 1
                emp_store[i] = emp
            if predict_set.size!=0:
                emp_store[i] = emp
            if predict_set.size>1:
                mult += 1
                mult_store[i] = mult
            if predict_set.size<=1:
                mult_store[i] = mult
        x = np.arange(rows)
        fig = plt.figure()
        plt.plot(x, err_store, label='Errors')
        plt.plot(x, emp_store, label='Empty predictions')
        plt.plot(x, mult_store, label='Multiple predictions')
        plt.xlabel('Examples')
        plt.ylabel('Cumulative errors, multiple and empty predictions')
        plt.legend(loc=2)
        plt.show()
        return fig

    def err_plot(p_values,y_data,
                sig_levels = np.array([0.01, 0.05, 0.1, 0.2])):
        ''' Plots the cummulative errors for different significance levels
            for an OCP'''
        rows = len(y_data)
        x = np.arange(rows)
        fig = plt.figure()
        for sig_level in sig_levels:
            err = 0
            mult = 0
            emp = 0
            err_store = np.zeros(rows)
            mult_store = np.zeros(rows)
            emp_store = np.zeros(rows)
            for i, sample in enumerate(p_values):
                predict_set = np.where(sample>=sig_level)[0]
                if y_data[i] not in predict_set:
                    err += 1
                    err_store[i] = err
                elif y_data[i] in predict_set:
                    err_store[i] = err
            plt.plot(x, err_store, label=str(sig_level*100)+'%')
        plt.xlabel('Examples')
        plt.ylabel('Cumulative errors at different confidence levels')
        plt.legend(title = 'Significance level')
        plt.show()
        return fig

    def ave_false_p(p_values, y_data):
        '''Plots the average false p-value for each epoch for an OCP'''
        rows = len(y_data)
        x = np.arange(rows)
        indices = np.arange(len(np.unique(y_data)))
        values = np.zeros(rows)

        for i, p in enumerate(p_values):

            predict_set = p[indices!=y_data[i]]
            ave_false = np.mean(predict_set)
            values[i] = ave_false


        fig = plt.figure(figsize=(16,8))
        plt.plot(x, values)

        plt.xlabel('Examples')
        plt.ylabel('Average false p-value')
        plt.show()

        return fig

    def ave_set_plot(p_values, epsilons = np.array([0.01, 0.05, 0.1, 0.2])):
        '''Plots the average prediction set size after each epoch for an OCP'''
        rows = p_values.shape[0]
        averages = np.zeros(rows) # to store average set size for each epsilons
        fig = plt.figure()
        for i, eps in enumerate(epsilons):
            for j in range(rows):
                temp = np.sum(p_values[:j+1]>eps,axis=1) # first average of set size
                averages[j] = np.average(temp) # then overall average
            plt.plot(np.arange(rows),averages, label = str(eps*100)+'%')
        plt.xlabel('Samples trained on')
        plt.ylabel('Average prediction set size')
        plt.ylim([0,10])
        plt.legend(title='Significance level')

        return fig

    def cred_samples(conf_table, X_test, min_cred=0.99, max_cred=1, heading=True):
        '''Plots 3x3 images of predictions with a defined range of credibility'''
        cred_table = conf_table[:,-1]
        cred_great = np.where(cred_table>min_cred)
        # locations of samples that have very high credibility
        cred_low =  np.where(cred_table<=max_cred)
        cred_loc = np.intersect1d(cred_great,cred_low)
        cred_values = conf_table[cred_loc]
        cred = X_test[cred_loc]
        cred_df = pd.DataFrame(cred_values, columns=['Label',
                                                     'Prediction',
                                                     'Confidence',
                                                     'Credibility'])
        samp_values = cred_df.sample(9)
        samp_idx = samp_values.index
        samp = X_test[cred_loc[samp_idx]]
        fig, ax = plt.subplots(3,3,figsize = (15, 15))
        loc = cred_loc[samp_idx]
        table = conf_table[loc]
        for i in range(3):
            for j in range(3):
                k = i*3+j
                fig = ax[i,j].figure
                if heading==True:
                    ax[i,j].set_title('True label: '+str(int(table[k,0]))+
                                       '   Predicted Label: '+
                                       str(int(table[k,1]))+
                                       '\nConfidence: '+
                                       str(round(table[k,2],4))+
                                       '   Credibility: '+str(
                                                 round(table[k,3],4)))
                else:
                    ax[i,j].set_title('True label: '+str(int(table[k,0]))+
                                       '   Predicted Label: '+
                                       str(int(table[k,1])))
                ax[i,j].imshow(to_image(samp[k]), cmap='gray')
                ax[i,j].set(xticklabels=[],yticklabels=[])
        return fig

    ### For anomaly detection

    def anomaly_eps(conf_table):
        '''Plots the number of the anomalies detected for different
           anomaly thresholds'''
        sig_levels = np.arange(0,1,0.001)
        num_anom = np.zeros(len(sig_levels))
        fig = plt.figure()
        for i, sig in enumerate(sig_levels):
            num_anom[i] += np.sum(conf_table[:,-1]<sig)
        plt.plot(sig_levels,num_anom)
        plt.xlabel('Anomaly threshold $\epsilon$')
        plt.ylabel('Anomalies detected')
        return fig

    def anomaly_eps_errs(p_values):
        '''Plots the frequency of the anomalies detected for
           different anomaly thresholds'''
        rows = p_values.shape[0]
        sig_levels = np.arange(0,1,0.001)
        num_anom = np.zeros(len(sig_levels))
        fig = plt.figure()
        for i, sig in enumerate(sig_levels):
            temp = p_values<sig
            num_anom[i] = np.sum(np.all(temp, axis=1))
        num_anom = num_anom/rows
        plt.plot(sig_levels,num_anom)
        plt.xlabel('Anomaly threshold $\epsilon$')
        plt.ylabel('Percentage of anomlies detected')
        return fig

    def anomaly_online(p_values, sig_levels = np.array([0.01, 0.05, 0.1, 0.2])):
        '''Returns plot of number of anomalies detected for
           different anomaly threshold for each epoch'''
        rows = p_values.shape[0]
        fig = plt.figure()
        for sig in sig_levels:
            anom = 0
            num_anom = np.zeros(rows)
            for i, sample in enumerate(p_values):
                temp = np.any(sample>sig)
                if temp==False:
                    anom += 1
                num_anom[i] = anom
            plt.plot(np.arange(rows), num_anom, label=str(sig*100)+'%')
        plt.xlabel('Examples')
        plt.ylabel('Anomalies detected')
        plt.legend(title='Anomaly threshold')
        return fig

    def anomaly_eps_models(p_values_set, models):

        fig = plt.figure()
        sig_levels = np.arange(0,1,0.001)
        for j, p_values in enumerate(p_values_set):
            rows = p_values.shape[0]
            num_anom = np.zeros(len(sig_levels))
            for i, sig in enumerate(sig_levels):
                temp = p_values<sig
                num_anom[i] = np.sum(np.all(temp, axis=1))
            num_anom = num_anom/rows
            plt.plot(sig_levels,num_anom, label = models[j])
        plt.xlabel('Anomaly threshold $\epsilon$')
        plt.ylabel('Percentage of anomalies detected')
        plt.legend()
        return fig

    def anom_detect_curve(p_values_set, models, sig_levels = np.arange(0,1,0.001)):
        '''Plots the frequency of anomalies detected for different
           anomaly thresholds'''
        fig = plt.figure()
        for j, model in enumerate(p_values_set):
            p_values = p_values_set[model]
            rows = p_values.shape[0]
            num_anom = np.zeros(len(sig_levels))
            for i, sig in enumerate(sig_levels):
                temp = p_values<sig
                num_anom[i] = np.sum(np.all(temp, axis=1))
            num_anom = num_anom/rows
            plt.plot(sig_levels,num_anom, label = models[j])
        plt.xlabel('Anomaly threshold $\epsilon$')
        plt.ylabel('Percentage of anomalies detected')
        plt.legend()
        return fig

    def anom_detect_hist(p_values_set, models, hist=20):
        '''Plots a histogram of the credibilities of samples for different models'''
        temp_cred = []
        for model in model_pvalues:
            p_values = model_pvalues[model]
            cred_vals = np.max(p_values, axis=1)
            temp_cred.append(cred_vals)
        temp_cred = np.array(temp_cred).T
        fig = plt.hist(temp_cred, histtype='step', label = model_names, bins = hist)
        plt.xlabel('Credibility')
        plt.ylabel('Anomalies')
        plt.legend()
        return fig

    def rand_anom_samples(X_data):
        '''Plots 9 random samples'''
        num_samp = X_data.shape[0]
        samp = np.random.randint(1,num_samp, 9)
        fig, ax = plt.subplots(3,3,figsize = (15, 15))
        for i in range(3):
            for j in range(3):
                k = i*3+j
                fig = ax[i,j].figure
                image = to_image(X_data[samp[k]])
                ax[i,j].imshow(image, cmap='gray')
                ax[i,j
                  ].set(xticklabels=[],yticklabels=[])
        return fig
```