Department of Information Engineering and Computer Science

Bachelor's Degree in
Computer Science

FINAL DISSERTATION

Explanation-based Interactive Debugging of Deep Neural Networks

A User Study

Supervisor
Prof. Andrea Passerini

Student
Massimo Stefan

Co-supervisors
Dott. Stefano Teso
Dott. Andrea Bontempelli

Academic year 2021/2022

Acknowledgments

To my family, my friends and my Biriam. Thank you all to for the support, the strength, and the laughs we shared all together. For me it has not been easy, but with you all it has been fun.

Contents

Abstract

Do you trust the pilot of your plane? And the algorithms that support the pilot? If the algorithm was an AI, that is, not programmed to follow a series of actions, but rather to learn to perform the task, would you trust it? Even if someone would reply "yes, I trust the science behind the algorithm", in reality very few would be comfortable to be the first passengers of a new self-driving plane. But what if its decisions about how to fly were public and human understandable? What if they were "explainable"? Now, maybe some more people would trust the machine.

Let us then assume that you are in charge of the training phase of the algorithm, where it learns how to fly and what to do in several situations, inside a very realistic simulation environment. Suppose that after a proper train, at every simulated crash, it displays one previous footage where the situation and the decision taken were similar. Even if the plane did not crash before, it was probably that particular instance that wrongly taught the algorithm how to behave. Having the awareness and consciousness that fixing a previous error would result in better performances and less probability of crashing the plane would make you feel more prone to test it in real life? Of course, maybe after some more trials.

CINCER (Contrastive and InflueNt CounterExample stRategy) [18] is a new explainable interactive label cleaning algorithm that aims to do exactly that. Given a data set with a certain percentage of label noise, firstly train the model with it. Then for every incoming suspicious example, CINCER decides if the example is compatible or not with the model. If it is compatible, it is directly added to the training set; otherwise a counter-example is found and an annotator is asked to relabel the suspicious example, the counter-example, or both, updating the data set accordingly. A counter-example is a training example that maximally supports the machine's suspicion. [18] presents a simulated evaluation of the algorithm compared to Influence Functions (IFs) and Nearest Neighbor (1-NN). It is shown that the counter-examples selected by CINCER improve the quality of the data, that influence-based selection strategies identifies the most mislabeled counter-examples and that both influence and curvature contribute to the effectiveness of CINCER. Overall, substituting the Hessian matrix with the Fisher information matrix (FIM) works great to identify influential examples in a robust manner. However,the quality of the counter-examples identified by CINCER has not been evaluated with real users.

My work bridges this gap, enabling a proper evaluation of the CINCER algorithm compared to IFs and NN with actual humans. The evaluation process is subdivided in different modules: the first (built on to of the CINCER algorithm) generate the example images for the survey, the second generate the actual surveys and handle the response's storage. Then, after the end of the survey, it processes the user's responses and generates the statistical plots displayed in Chapter 5. The experiment highlights and remarks the theoretical expectation, i.e., CINCER's counter-examples are chosen more times that IF's counter-examples, but the users tend to prefer NN anyway, as result of similarity at pixel's level. This result indicates a possible path forward to improve user acceptance of the counter-examples extracted by CINCER.

1 Introduction

1.1 Machine learning, explainability and label noise

Machine learning refers to algorithms for making computers learn and improve from experience. It is a broad subject subdivided into three branches, i.e., Supervised Learning, Unsupervised Learning, and Reinforcement Learning. The objective of supervised learning, which is the focus of this thesis, is to learn a predictive model that maps data features (e.g., car size, speed, brand) to an output (e.g., car price). Predictions from never before seen instances can be obtained from a fully trained machine learning model. The generated models are usually implemented as "black boxes" in the sense that they often are incredibly complex (especially in the case of deep models), making it near impossible to understand why some output has been generated. This lack of "explainability" or "interpretability" is one of the most significant problems for some types of machine learning models, such as neural networks (NNs). Throughout the thesis, these two terms will be used interchangeably.

Another issue is label noise, a widespread problem when handling machine learning models, and it is considered to be the observed labels that are classified incorrectly. Label noise may occur for various reasons [13], for instance:

1. **Insufficient information**, such as poor data quality or limited expressiveness. For example, if only part of the symptoms are given to an annotator, the algorithm may provide incorrect disease labels, since multiple illnesses may have the same symptoms.

2. **Experts make mistakes**: no one (not even doctors!) is always correct.

3. **Classification is subjective in some cases**, the illness of one patient could be diagnosed differently. For example a simple headache could be related to too many hours playing video games or a brain tumor.

Most of these problems are generally present in each experiment and are hard to solve, so label noise is almost omnipresent. Around 5% of encoding errors are estimated to be contained in Real-world databases [5].

1.2 CINCER

The work presented in this thesis has the main objective to evaluate CINCER, with real users. CINCER (Contrastive and InflueNt CounterExample stRategy) is a new explainable interactive label cleaning algorithm that lets an annotator observe and fix the reasons behind the model's suspicions. Once the model has been trained, at test time, every new example is defined compatible or suspicious. In the first case, the example is directly added to the data set. Otherwise a counter-example is found by CINCER that, according to the model, is maximally incompatible with the suspicious example. The counter-examples help the model understand previous mistakes from current examples. Subsequently, it asks an annotator to relabel either or both examples resolving the possible inconsistency, updating the data set accordingly. CINCER is not the only algorithm that can be used to find counter-examples that support the model's decision; IF (Influence Functions) and 1-NN (Nearest Neighbours) are possible alternatives [18].

In particular, the counter-examples selected by CINCER are calculated approximating IF replacing the default Hessian matrix with the Fisher information matrix (FIM). Moreover, CINCER enables the user to only focus on those examples that directly affect the model behavior, making the interactive algorithm more responsive. This feature correspond to a game-changing approach since usually the entire data set needs to be checked to find interesting counter-examples.

1.3 The missing gap

A focal point of building an algorithm, in this case an explainable interactive label cleaning one, is evaluating its performances and compare them with others obtained from algorithms with the same goal. In [18], multiple simulations have been run to evaluate the algorithm performances against others. The objectives achieved are various. Compared to previous approaches, CINCER identifies the reasons behind the model's skepticism and asks the supervisor to double-check them too. Then, their experiments shows that, by removing inconsistencies in the data, CINCER enables acquiring better data and models than less informed alternatives. Finally, substituting the Hessian matrix with the Fisher information matrix (FIM) works greatly to identify influential examples in a robust manner. However, the tests did not specifically evaluate the quality and believability of CINCER's explanations from the user's perspective. They have been only simulated, therefore there is no evidence that the interpretability proven in theory from the building blocks of CINCER and the mathematical formulas exploited, is consistent in the real world, in practice. There is a need to fill this gap, i.e. to test the algorithm with real users through a survey, and evaluate their answers.

To achieve this, I developed a whole process to prove the effectiveness of CINCER in a real environment, comparing the performances of CINCER, IFs, and NN, with real users. The process can be split into four different modules:

- the supporting example creation tool (built on top of the CINCER algorithm, Section 4.2) to generate the example images for the survey respondents
- the survey generator (Section 4.3) to create various surveys with the corresponding responses and links file
- the web app (Section 4.4) to access the survey generated from the previous algorithm randomly
- the statistic generator (Section 4.5) to generate visual statistics

1.4 Structure of the thesis

Chapter 2 is a broad introduction to Explainability, explaining the concept in general, when its usage is recommended (and when it is not), and the interpretable models classification. Then, Influence functions (IFs) and the relative equations are heavily described since they are frequently referenced in the following section. Finally CINCER, the protagonist of the Evaluation environment process, is in detail explained, from the pseudo code to the equations involved.

Chapter 4 describes the content of my internship, the Evaluation environment. The software and platforms utilized are described in Section 4.1, then an in-depth explanation is presented for each module.

Chapter 5 presents and discusses the results and reports the statistic we obtained, highlighting the goals reached and answering the desideratums. In conclusion, a sums up of the work is presented in Chapter 6, also describing possible future improvements.

2 Background

2.1 Explainability

Explainability in machine learning means it is possible to explain what happens in a model from input to output, making models transparent and solving the black box problem. Explainable AI (XAI) focuses on developing methods that help human experts understand solutions developed in the content of AI [2].

The need for interpretability stems from incompleteness in the problem formalization, creating a fundamental barrier to optimization and evaluation. It means that it is not enough for specific

problems or tasks to get the prediction (the **what**). The model must also explain how it came to the prediction (the **why**) because a correct one only partially solves the original problem [12].

In most real-world tasks, a single metric description, such as classification accuracy, is an incomplete representation of the solution (and therefore of the problem). So, knowing the 'why' can help learn more about the problem, the data, and how a model might fail.

Explainability is essential in many domains (such as health care, marketing, and bio-informatics). In other domains, such as finance, algorithms have been operating without human interaction and explanations for years. If a system predicts trending sufficiently well, there are no further concerns about it, and no explanation is required. The absence of intepretability requirements translates in a highest commitment on achieving more precise and trustworthy predictions, more relevant on certain fields.

2.1.1 When interpretability is useful and not

The motifs that have led to a particular focus on this matter have been various and not only practical.

For example, explanations are used to **manage social interactions**, and the explainer influences the users' actions, emotions, and beliefs by generating a proper output. For example, a vacuum cleaner could create a shared meaning of a "stop motion" explaining that it cannot exit from a bathroom instead of simply stopping to work without comment [16].

Next, Machine learning models can only be **debugged and audited** when they can be interpreted. An explained incorrect prediction helps to understand the cause of the error and deliver a direction towards a fix [16].

One goal of incorporating explainability into decision-making is to make an algorithm accountable for its actions. For a system to be **accountable**, it must be able to explain and justify its decisions [16].

The machine learning model's biases are picked up from the training data by default. For this reason, underrepresented groups could be discriminated against and hardly identified. Interpretability is even helpful for **detecting biases** in machine learning models [16].

Interpretability is not required when the model has no significant impact (binary classification problem), the problem is well studied (face recognition), and when might enable people or programs to manipulate the system. Problems with users who deceive a system result from a mismatch between the goals of the creator and the user of a model. Take into consideration a dating site, where the creator wants users to buy premium features to have greater chances of getting picked from other people, while users prefer to have more matches for free. This mismatch between the goals introduces incentives for applicants to game the system to increase the number of matches without paying. If a user knows that overtaking the daily maximum number of "swiping right" (liking a person) reduces the visibility from other users, he simply reduces the active search increasing the passive one, without spending money. While the overall chances of new matches increases, the app remains free for him. The system can only be gamed if the inputs are proxies for a causal feature, but do not actually cause the outcome [16].

2.1.2 Interpretable models classification

Various criteria can be used to classify methods for machine learning interpretability.

XAI models can first be grouped into **intrinsic** interpretability and **post-hoc** interpretability. Intrinsic interpretability refers to machine learning models with a simple structure, such as short decision trees or sparse linear models. Post-hoc interpretability instead refers to the application of interpretation methods after model training, widely used for non-shallow machine learning models. An example could be the permutation of feature importance.

Then, they can be classified either as **model-agnostic** or **model-specific**. Model-agnostic interpretation tools are applied after the model has been trained (post-hoc) and can be used on any machine learning model. These methods usually work by analyzing feature input and output pairs since, by definition cannot have access to model internals such as weights or structural information. Model-specific tools instead are limited to specific model classes. The interpretation of intrinsically interpretable models is always model-specific, but the opposite is not always true. For example, tools that only work for interpreting, e.g., neural networks, are model-specific.

Finally, XAI models can be categorized as **local** if they interpret each instance and feature in the data individually and how features affect the result, or **global** if they explain the entire model behavior showing the big picture view of the model [16].

2.1.3 Related work: explainable interactive learning

Several methods have been devised to leverage the model's explanations within an interactive learning loop, such as CAIPI (Interactive Medical Image Classification), Debugging Deep Text Classifiers (FIND), or Debugging of Natural Language Inference Models Using Influence Functions (HILDIF).

The first aims to enable domain experts such as clinicians to train and apply trustworthy ML models. Their domain of interest is the classification of medical images from diagnostics in everyday clinical practices, such as classifying computer tomography scans into their corresponding categories, e.g. abdomenf, chest, and brain. Although they do not provide an explicit architecture, their scope is to enable human evaluation as they aim to enable experts to control the data quality and to monitor and correct the behavior of the ML system.[17, 19]

The second is a framework that enables humans to debug deep learning text classifiers by disabling irrelevant hidden features. The problem is that existing techniques only allow humans to provide feedback on individual predictions, and additional training examples are created based on the feedback to retrain the models. Similarly, these existing techniques allow for rectifying only errors related to examples at hand but do not provide a way to fix hidden problems in the model parameters. Local improvements for individual predictions could add up to inferior overall performance. So, they created the framework FIND (Feature Investigation aNd Disabling) to exploit an explanation method, namely layer-wise relevance propagation (LRP), to understand the behavior of a classifier when it predicts each training instance. After, it aggregates all the information using word clouds to generate a global visual picture of the model.[15]

The third is a novel explanatory debugging pipeline called HILDIF (Human In The Loop Debugging using Influence Functions), enabling humans to improve deep text classifiers using influence functions as an explanation method. They experiment on the NLI task, showing that HILDIF can effectively alleviate artifact problems in fine-tuned BERT (Bidirectional Encoder Representations from Transformers) models and increase model generalizability.[20]

2.2 Influential instances

As previously said, machine learning models are a product of training data, and every instance can impact the resulting model differently. Training instances are called "influential" when their deletion from the training data considerably changes the parameters or predictions of the model. Influential training instances are used in machine learning to "debug" the models and better explain their behaviors and predictions.

There are two main approaches for identifying influential instances: **deletion diagnostics** (the simplest one, not discussed in detail) and **influence functions**. Both are based on robust statistics, which provides statistical methods less affected by outliers or violations of model assumptions. Moreover, robust statistics provides methods to measure how robust estimates from data are (such as the weights of a prediction model or a mean estimate).

Suppose the goal is to estimate the average income of a group of workers and understand if it is influenced by a single rich or poor person. The problem can be solved by recalculating the mean value by omitting individual answers or approximated mathematically via "influence functions". With the deletion approach, the mean value is computed n times, where n is the number of workers, omitting one of the income statements each time and measuring how much the mean estimate changes. A significant change means that an instance is very influential. Also, deletion requires retraining which is slow and often unstable for larger models like NNs. The second approach upweights one of the incomes by an infinitesimally small weight, which corresponds to the calculation of the first derivative of a statistic or model. It approximates the parameter changes based on the gradients of the model. Since the mean scales linearly with single values, the mean estimate can be very strongly influenced by a single value.

Deletion diagnostics and influence functions can also be applied to machine learning models' pa-

rameters or predictions to better understand their behavior (**global** interpretability) or to explain individual predictions (**local** interpretability). With influential instances, the model is not treated as fixed but as a function of the training data.

2.2.1 Outliers vs influential instances

It is essential to understand the difference between an outlier and an influential instance. An *outlier* is an instance that is very distant from the other instances in the data set, and the distance (such as Euclidean or manhattan) to all the other samples needs to be significant. For example, a 2.5 m height person is considered an outlier in a data set of people. When an outlier influences the model, it is also an influential instance. An *influential instance* is a data instance whose removal strongly affects the trained model. When the model is retrained without a particular instance, the more the model parameters or predictions change, the more influential that instance is.

2.2.2 Influence functions

The influence function measures how strongly the model predictions or parameters depend on a training instance. Instead of deleting each instance and retrain the model multiple times, the method upweights one instance in the loss by a tiny step and approximates the loss around the current model parameters using the gradient and Hessian matrix. Influence functions have been suggested by Koh and Liang (2017) [14] to measure how an instance influences model parameters or predictions. They require access to the loss gradient with respect to the model parameters, which only works for a subset of machine learning models: Logistic regression, Neural Networks (NNs), and Support Vector Machines (SVM).

Specifically, Influence functions can be applied to models with loss functions twice differentiable with respect to its parameters (such as the logistic loss) to approximate the influence of one instance on the model parameters and the prediction.

Key idea

The key idea behind influence functions is to upweight the loss of a training instance by an infinitesimally small step ϵ, resulting in new model parameters:

$$\hat{\theta}_{\epsilon,z} \stackrel{\text{def}}{=} \arg\min_{\theta\in\Theta} (1-\epsilon)\frac{1}{n}\sum_{i=1}^{n} L(z_i,\theta) + \epsilon L(z,\theta) \tag{2.1}$$

where θ is the model parameter vector, $\hat{\theta}_{\epsilon,z}$ is the parameter vector after upweighting z by a tiny number ϵ, $L(z,\theta)$ is the loss function with which the model is trained, z_i is the training data and z is the training instance needs to be upweighted to simulate its removal. The result corresponds to the loss difference upweighting a particular example z_i from the training data by a little (ϵ) and downweight the other data instances accordingly.

The influence function of the parameters

The influence function of the parameters, i.e., the influence of upweighting a training instance z on the parameters, can be calculated as follows:

$$I_{\text{up,params}}(z) \stackrel{\text{def}}{=} \left.\frac{d\hat{\theta}_{\epsilon,z}}{d\epsilon}\right|_{\epsilon=0} = -H_{\hat{\theta}}^{-1}\nabla_\theta L(z,\hat{\theta}) \tag{2.2}$$

where the expression $\nabla_\theta L(z,\hat{\theta})$ is the loss gradient with respect to the parameters for the upweighted training instance, while $H_{\hat{\theta}}^{-1}$ is the inverse Hessian matrix. The gradient is the rate of change of the loss of the training instance and represents how much the loss changes when the model parameters $\hat{\theta}$ are changed by ϵ. A positive entry in the gradient vector means that a small increase in the corresponding model parameter increases the loss, a negative entry means that the increase of the parameter reduces the loss.

The Hessian matrix

The first part $H_{\hat{\theta}}^{-1}$ is the inverse *Hessian matrix* (second derivative of the loss with respect to the model parameters). The Hessian matrix is the rate of change of the gradient, or expressed as loss, it is the rate of change of the loss. It can be estimated using:

$$H_\theta = \frac{1}{n}\sum_{i=1}^{n} \nabla^2_{\hat{\theta}} L(z_i, \hat{\theta}) \tag{2.3}$$

which informally corresponds to how curved the loss is at a certain point. The intuition behind this influence function formula is that it is unknown how exactly the loss of instance z will change when it is removed/upweighted. The function is approximated locally by using information about the steepness (the gradient) and the curvature (the Hessian matrix) at the current model parameter setting. The Hessian matrix is usually cached for better performances when used in real-world scenarios.

The new parameters

Using Eq. 2.2 it is possible to calculate what the new parameters would approximately look like if the instance z is upweighted:

$$\theta_{-z} \approx \hat{\theta} - \frac{1}{n} I_{\text{up, params}}(z) \tag{2.4}$$

The resulting vector is the original one minus the gradient of the loss of z (decreasing the loss is the objective) scaled by the curvature (since it is multiplied by the inverse Hessian matrix) and scaled by $-\frac{1}{n}$, because that is the weight corresponding to *deleting* an instance from the training set.

To find out how the *prediction* changes when a training instance z is upweighted there are two ways. The first is to calculate the new parameters and then make predictions using the new model. Otherwise, calculate the influence of instance z on the predictions directly, since the influence can be calculated by using the chain rule:

$$\begin{aligned} I_{\text{up, loss}}(z, z_{\text{test}}) &= \left.\frac{dL(z_{\text{test}}, \hat{\theta}_{\epsilon,z})}{d\epsilon}\right|_{\epsilon=0} \\ &= \nabla_\theta L(z_{\text{test}}, \hat{\theta})^\intercal \left.\frac{d\hat{\theta}_{\epsilon,z}}{d\epsilon}\right|_{\epsilon=0} \\ &= -\nabla_\theta L(z_{\text{test}}, \hat{\theta})^\intercal H_\theta^{-1} \nabla_\theta L(z, \hat{\theta}) \end{aligned} \tag{2.5}$$

The first line measures the influence of a training instance on a certain prediction z_{test} as a change in loss of the test instance after upweighting the instance z and getting the new parameters $\hat{\theta}_{\epsilon,z}$.

In the second line the chain rule is applied obtaining the derivative of the loss of the test instance with respect to the parameters times the influence of z on the parameters. The third line replaces the expression with the influence function for the parameters. The first term $\nabla_\theta L(z_{\text{test}}, \hat{\theta})^\intercal$ is the gradient of the test instance with respect to the model parameters.

Informally, the formula $I_{\text{up, loss}}(z, z_{\text{test}})$ states that the influence function of the training instance z on the prediction of an instance z_{test} is "how strongly the instance reacts to a change of the model parameters" multiplied by "how much the parameters change when the instance z is upweighted".

Influence functions - Pros and Cons

The influence functions approach differs significantly from the other model-agnostics approaches since the latter only focus on feature-perturbation. Instead, influential instances emphasize training data's role in the learning process. This aspect makes influence functions one of the best debugging tools for machine learning models since they directly help identify the instances that should be checked for errors.

Firstly, as previously said, it is **much quicker than the deletion diagnostics** approach since it does not need to retrain the model multiple times. Then, influence functions via derivatives can also be used to **create adversarial training data**. These are instances that are manipulated in such a

way that the model cannot predict certain test instances correctly when the model is trained on those manipulated instances. Influence functions can be used to **compare different machine learning models** and better understand their different behaviors beyond only the predictive performance. Finally, it is also possible to **identify groups of influential instances** [16].

However, influence functions have some drawbacks and are not applicable to every model. **Only models with a second-order differentiable loss function with respect to its parameters**, such as neural networks, can use this approach to identify influential instances. They do not work for tree-based methods like random forests, boosted trees, or decision trees. Then, the models with a loss function non-differentiable could apply Influence Functions through a trick, namely substituting the loss with a differentiable one for calculating the influence for the influence functions, and train the model with a non-differentiable loss (like the Hinge loss). Next, **influence functions are an approximation** because the approach forms a quadratic expansion around the parameters. There is no guarantee that the approximation will be close to the ground truth. In conclusion, **there is no explicit cutoff of the influence measure at which an instance can be called influential or non-influential**. The instances can be sorted by influence, but a threshold is essential to bipartite them. The bipartition could be valuable or fundamental in some situations, but require the addition of an hyperparameter to the model.

2.3 CINCER

This chapter focuses on CINCER, the starting point, and the protagonist of the Evaluation environment development. As previously said, label noise is a significant and constant issue in machine learning as it can lead to unreliable models and compromised predictive performances. Label noise is often tackled by monitoring incoming examples that are likely mislabeled, aka suspicious examples, and asking to a supervisor to provide clean (or at least better) annotations. Using this strategy, however, implies degradation in the model's quality and prevents spotting future mislabeled examples that fall in regions affected by noise. It happens because the noisy examples that do not undergo the cleaning step (e.g., those in the initial bootstrap data set) or that manage to elude it are left untouched, since there is no way of updating them. CINCER stands out compared to existing approaches which focus solely on cleaning the incoming examples, because it identifies a counter-example, i.e., a training example that maximally supports the machine's suspicion, and then asks a human annotator to update the suspicious example, the counter-example, or both. In other words, if an incorrect example has already been embedded in the data set after the initial training or was not identified as suspicious, CINCER could locate the fallacy whilst other algorithms could not.

CINCER (Contrastive and InflueNt CounterExample stRategy) is a new explainable interactive label cleaning algorithm. A **label cleaning algorithm**'s goal is to identify the data points that generate noise and are responsible for derailing the model from better performances. Furthermore, CINCER is **interactive** and works in settings where a human supervisor, usually a domain expert, can be asked to double-check and relabel any potentially mislabeled example. Finally, CINCER is **explainable**; why an incoming example is mislabeled must be clear to the supervisor from the selected counter-example displayed.

The data inconsistency is represented by the pair example/counter-example. The inconsistency is resolved by asking the supervisor to relabel the example, the counter-example, or both. The goal is to acquire a clean data set and a high-quality predictor while asking a few relabeling queries to keep the interaction cost under control. It is important to notice that usually there is no possibility of picking only the interesting counter-examples and the only strategy is to check the entire data set. It can be easily deduced that the interaction in these environments, where the annotator need to check the whole training set, is tedious and not implemented.

2.3.1 Other approaches

The two approaches discussed in [18] are Skeptical Learning (SKL) and Learning from Weak Annotators (LWA). The first learns from unreliable users, and, for each example it receives, the machine compares (an estimate of) the annotation quality with that of its prediction. The type of label quality estimation depends on the implementation. If the prediction looks more reliable than the annotation

by some factor, SKL asks the user to double-check his/her example. LWA, on the other hand, focuses on querying domain experts. The most recent approach identifies suspicious examples that have a significant impact. Moreover, it estimates the noise rate directly since it jointly learns a prediction pipeline composed of a classifier and a noisy channel.

As previously said, focusing on cleaning the incoming examples only (like SKL) is not an excellent solution to remove noise from the data set. The significant issues with label noise are detrimental to the model performance and the ability to identify suspicious examples. Label noise is inserted in two ways: adding to the training set examples that manage to elude the cleaning steps (widespread in the initial stage of the model) and using the initial bootstrap data set.

2.3.2 The algorithm

It is considered a general class of probabilistic classifiers $f : \mathbb{R}^d \to [c]$ of the form $f(x;\theta) \stackrel{\text{def}}{=} \arg\max_{y\in[c]} P(y \mid x;\theta)$, where the conditional distribution $P(Y|X;\theta)$ has been fit on training data by minimizing the cross-entropy loss $L((x,y),\theta) = -\sum_{i\in[c]}\{i = y\} \log P(i \mid x,\theta)$. In the implementation, P is assumed to be a neural network with a softmax activation at the top layer, trained using some variant of SGD and possibly early stopping [18].

Algorithm 1 Pseudo-code of CINCER. **Inputs**: initial (noisy) training set D_0; threshold τ

```
fit θ_0 on D_0
for t = 1, 2, ... do
    receive new example z̃_t = (x_t, ỹ_t)
    if μ(z̃_t, θ_{t−1}) < τ then
        D_t ← D_{t−1} ∪ {z̃_t}                          ▷ z̃_t is compatible
    else
        find counterexample z_k using Eq. 2.11          ▷ z̃_t is suspicious
        present z̃_t, z_k to the user
        receive possibly cleaned labels y'_t, y'_k
        D_t ← (D_{t−1} \ {z_k}) ∪ {(x_t, y'_t), (x_k, y'_k)}
    fit θ_t on D_t
```

CINCER works as follows: at first the machine acquires a training set $D_{t-1} = \{z_1, ..., z_{t-1}\}$ and trains a model with parameters θ_{t-1} on it, then the for loop begin. D_{t-1} is the initial training set that might already contain some noise. Then, for every new example the machine receives, the algorithm needs to decide if it is compatible (and trust it) or not. To do this, CINCER follows the skeptical learning approach computing the margin $\mu(\tilde{z}_t, \theta_{t-1})$, namely the difference in conditional probability between the model's prediction $\hat{y}_t \stackrel{\text{def}}{=} \arg\max_y P(y \mid x_t, \theta_{t-1})$ and the annotation $\tilde{y}_t$.

$$\mu(\tilde{z}_t, \theta_{t-1}) \stackrel{\text{def}}{=} P(\hat{y}_t \mid x_t, \theta_{t-1}) - P(\tilde{y}_t \mid x_t, \theta_{t-1}) \tag{2.6}$$

The margin estimates the incompatibility between the model and the example: the larger the margin, the more suspicious the example. Here comes into play the only hyperparameter of the algorithm, the threshold τ. An incoming example $\tilde{z}_t$ is marked as suspicious if its margin (Eq. 2.6) is greater than τ, and compatible otherwise.

If $\tilde{z}_t$ is compatible, it is directly added to the data set. Otherwise, CINCER find a counterexample $z_k \in D_{t-1}$ that maximally supports the machine's prediction using Eq. 2.11 discussed in the following section. Then, CINCER asks the annotator to double-check the pair $(\tilde{z}_t, z_k)$ and relabel the suspicious example, the counter-example, or both, updating them accordingly and inserting them in the data set, beginning again the loop.

2.3.3 Counter-example selection

Counter-examples are the training examples that maximally support the machine's suspicion given an incoming suspicious example. Counter-examples are meant to illustrate why a particular example $\tilde{z}_t$ is chosen and help to provide a useful corrective feedback from the supervisor to the machine. CINCER aims to find counter-examples that should be:

1. **Contrastive**: z_k should explain why $\tilde{z}_t$ is considered suspicious by the model, thus highlighting a potential inconsistency in the data
2. **Influential**: if z_k is mislabeled, correcting it should improve the model as much as possible, so to maximize the information gained by interacting with the annotator
3. **Pertinent**: it should be clear *to the user* why z_k is a counter-example for $\tilde{z}_t$

The following sub-sections discusses in details the meaning of each desideratum and how they are obtained.

Contrastive counter-examples

The first desideratum heavily involves influence functions discussed in Section 2.2.2. How does CINCER obtain contrastive counter-examples? Formally, let θ_{t-1} be the parameters of the current model. Given a suspicious example $\tilde{z}_t$, $z_k \in D_{t-1}$ is a contrastive counter-example if *removing* it from the data set and retraining leads to a model with parameters θ_{t-1}^{-k} that assigns *higher* probability to the suspicious label $\tilde{y}_t$. The goal is to find the counter-example that maximally affects the change in probability, namely the maximum difference between the conditional distribution $P(\tilde{y}_t \mid x_t; \theta_{t-1}^{-k})$ and $P(\tilde{y}_t \mid x_t; \theta_{t-1})$. The most contrastive counter-example is:

$$\arg\max_{k \in [t-1]} \{P(\tilde{y}_t \mid x_t; \theta_{t-1}^{-k}) - P(\tilde{y}_t \mid x_t; \theta_{t-1})\} \tag{2.7}$$

As discussed in section 2.2, retraining the model $|D_{t-1}|$ times to optimize Eq. 2.7 omitting one example each time and measuring how much the mean estimate changes (deletion diagnostics) is impractical for real size data sets. On top of that, the scenario considered is interactive, so the whole process should be computed for every incoming suspicious example.

The problem is solved exploiting influence functions (IFs), an alternative to deletion diagnostics that measures how strongly the model predictions or parameters depend on a training instance. The method upweights one instance in the loss by a tiny step and approximates the loss around the current model parameters using the gradient and Hessian matrix, without retraining.

Firstly, the parameter vector θ_t after upweighting z by a tiny number ϵ ($\theta_t(z, \epsilon)$) is computed with equation 2.1. The result is then used to compute the so-called influence function $\mathcal{I}_{\theta_t}(z)$. If the loss is strongly convex and twice differentiable, the IF can be written as Eq. 2.2, where the curvature matrix (the Hessian matrix, Eq. 2.3) is positive and invertible. Finally, applying the chain rule (Eq. 2.5) it is shown that the most contrastive counter-example (Eq. 2.7) can be approximated to:

$$\arg\max_{k \in [t-1]} \nabla_\theta P(\tilde{y}_t \mid x_t; \theta_{t-1})^\intercal H(\theta_{t-1})^{-1} \nabla_\theta L(z_k, \theta_{t-1}) \tag{2.8}$$

Eq. 2.8 can be efficiently solved by caching the inverse Hessian-vector product (HPV) $\nabla_\theta P(\tilde{y}_t \mid x_t; \theta_{t-1})^\intercal H(\theta_{t-1})^{-1}$, so that evaluating the objective on each z_k becomes a simple dot product, and the inverse HPV is approximated with an efficient stochastic estimator like LISSA. This create an algorithm for computing contrastive counter-examples [18].

Influential counter-examples

The second desideratum is obtained producing the model by optimizing the cross-entropy loss. Therefore, the contrastive counter-examples are highly influential if $L(z, \theta) = -\log P(y \mid x; \theta)$. Using this kind of loss (twice differentiable with respect to its parameters), under the assumption and as long as the model satisfies $P(\tilde{y}_t \mid x_t; \theta_{t-1}) > 0$, Eq. 2.7 is equivalent to:

$$\arg\max_{k \in [t-1]} -\nabla_\theta L(\tilde{z}_t, \theta_{t-1})^\intercal H(\theta_{t-1})^{-1} \nabla_\theta L(z_k, \theta_{t-1}) \tag{2.9}$$

This shows that, for the large family of classifiers trained by cross-entropy, highly influential counter-examples are highly contrastive and vice versa, so that no change to the selection algorithm is necessary [18].

Pertinent counter-examples

The last desideratum is peculiar, since it does not involve elaborate formulas. To make counter-examples pertinent, CINCER restricts the choice of possible counter-examples. CINCER restricts the search to counter-examples whose labels in the training set are the same as the prediction for the suspicious example, i.e., $y_k = \hat{y}_t$. This way the counter-example can be interpreted by the annotator as being in support of the machine's suspicion. If the counter-example is labeled correctly (the machine is right), then the machine's suspicious is likely right and the incoming example needs to be cleaned. Otherwise, if the counter-example is incorrectly labeled (the machine is wrong) and the suspicious example is not mislabeled, it is likely that it is the counter-example (which backs the machine's suspicions) that needs cleaning. This desideratum is the one that motivates the experiments of this thesis.

2.3.4 Counter-example selection with the Fisher information matrix

Influence functions (IFs) estimates are fairly accurate for shallow networks, while for deeper networks the estimates are often erroneous in practice [11]. Being unstable reflects on the quality and reliability of the counter-examples identified by IFs. The issue is that, for the common use case of non-convex classifiers trained using gradient-based methods (and possibly early stopping), θ_{t-1} is seldom close to a local minimum of the empirical risk, rendering the Hessian non-positive definite [18].

For this reason, a different approach is taken by substituting the Hessian matrix with the Fisher information matrix (FIM). The FIM $F(\theta)$ of a discriminative model $P(Y \mid X, \theta)$ and training set D_{t-1} is:

$$F(\theta) \stackrel{\text{def}}{=} \frac{1}{t-1} \sum_{k=1}^{t-1} \mathbb{E}_{y \sim P(Y|x_k,\theta)} \left[\nabla_\theta \log P(y \mid x_k, \theta) \nabla_\theta \log P(y \mid x_k, \theta)^\intercal \right] \tag{2.10}$$

The FIM is a good approximation of the Hessian, since it captures much of the curvature information encoded into the Hessian. Moreover, the FIM approximates the Hessian if the model approximates the data distribution. For these reasons, Eq. 2.9 can be rewritten as:

$$\arg \max_{k \in [t-1]} -\nabla_\theta L(\tilde{z}_t, \theta_{t-1}) F(\theta_{t-1})^{-1} \nabla_\theta L(z_k, \theta_{t-1}) \tag{2.11}$$

This formulation have several advantages. At first, the FIM is positive semi-definite by construction, making the computation of Eq. 2.11 much more stable. Second, this optimization problem admits caching the inverse FIM-vector product (FVP), which makes it viable for interactive usage.

3 Experiment description

As previously stated, the objective of the thesis is to evaluate the CINCER algorithm via a survey with real users, using the Prolific platform, and compare it against IF (Influence Function) and 1-NN (Nearest Neighbour). This chapter describes the experiment and the choices made to build it.

Data set The experiment has been built on a noisy MNIST data set. MNIST is a database of handwritten digits, black-and-white, 28×28 with pixel values normalized in the $[0, 1]$ range. They are available from [7], where training set of 60.000 examples and a test set of 10.0000 examples are available. MNIST is a subset of a larger set available called NIST.

Research participants The 100 study participants have been selected through the Prolific platform, discussed in chapter 4.1. They did not require any peculiar trait, just being older than 18 years old.

3.1 The process

In this paragraph the process is high-level described, focusing on the decisions taken and the faced issues, without going into the details about the programming procedures and specifications.

3.1.1 Suspicious images selection

At first, from the data set some suspicious images need to be identified. To do so, the model is trained using a subsample of the training set. Then, 82 suspicious examples are identified by CINCER, and the three selection strategies are applied to find, inside the training set, one counter-example for each of the 82 suspicious examples selected. At first, the suspicious examples were only 42, but to obtain a greater number of them, every example is taken twice. It is important to notice that, even if the same suspicious examples is present two times, it does not mean that the counter-examples associated with it will be the same. Therefore, both examples can be used in the surveys without distinctions.

3.1.2 `select_counterexamples()`: old and new

The suspicious images selection is programmed inside the `select_counterexamples()` function, stored into the CINCER code repository [1]. This function generates a set of illustrations representing a suspicious example and the relative counter-examples selected from CINCER, 1-NN and IF.

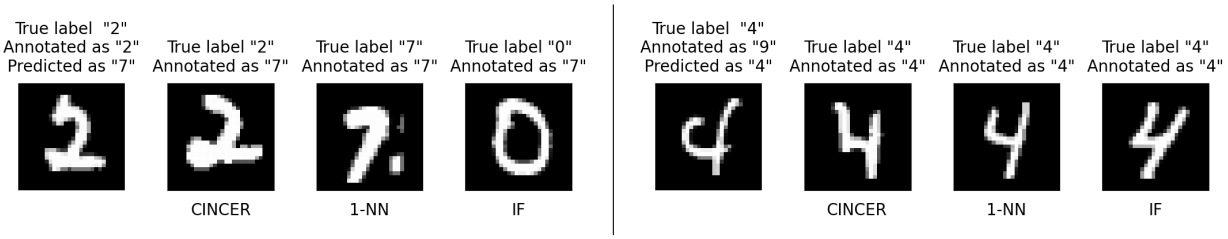

Figure 3.1: Suspicious example and counter-examples selected using (from left to right) CINCER, 1-NN and influence functions (IF), on a noisy MNIST. **Left**: the suspicious example is mislabeled, the machine's suspicion is supported by a clean counter-example. **Right**: the suspicious example is not mislabeled, the machine is wrongly suspicious because the counter-example is mislabeled [18]

The first image is a suspicious example from the noisy MNIST data set. On top of it, there are three labels:

- **True label**, which is the correct label of the image
- **Annotated as**, which is the label assigned to it by the supervisor and may be incorrect
- **Predicted as**, which is the machine prediction label on the example, and may also be incorrect

The following counter-examples are selected with (from left to right) CINCER, 1-NN, and influence functions (IF) with the relative "True label" and "Annotated as" tags from the same noisy MNIST data set.

The new module developed on top of it, instead, aims to generate for each instance six different images, each composed as 3.1 (suspicious example + three counter-examples), but with different information displayed (Figure 3.2). It generates six of them because it becomes valuable in the survey generation stage to remove question order biases. Each counter-example set order is a permutation of all the six possible permutations.

The pictures 3.1 and 3.2 (and consequently the different versions of `select_counterexamples()`) have diametrically opposed goals. The first tends to inform, marking each counter-example with the relative algorithm used to find it and appending labels to have a more extensive understanding of the choices. The second aim is to deceive which algorithm found which image. The intrinsic information of the image only explains 1) what the machine thinks and 2) which are the counter-examples, separating them from the first (the suspicious one) using a divider.

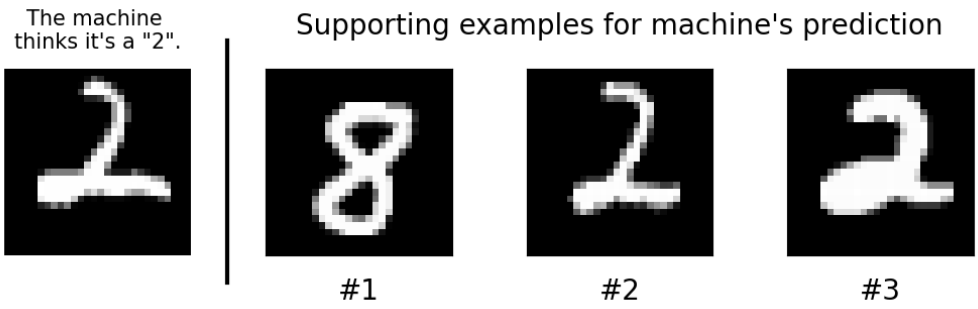

Figure 3.2: Suspicious example and counter-examples selected using CINCER, 1-NN and influence functions (IF), on a noisy MNIST, without labels and corresponding algorithm

3.1.3 Internal subdivisions

The suspicious examples are then splitted in four different groups, based on two factors: being correct (or incorrect) and ambiguous (or unambiguous). The images where the true label corresponds to the predicted label are considered **correct**. The pictures where the true label conforms to the annotation and not the prediction are considered **incorrect**. The images are considered **unambiguous** when the suspicious example image is with high certainty the representation of a number. Finally, the pictures are considered **ambiguous** if the number represented can be associated to more than one number (for example 1 and 7), or the number does not have a clear representation.

Meanwhile the first split can be automatized checking the true label compared to the "Annotate as" label and "Predicted as" label, the second needs a human evaluator to decide which (in his opinion) is ambiguous and which not. For this reason, the second split could be different based on the evaluator charged with the task.

The internal subdivision operation enables the survey generator (Section 4.3) to take various pictures from each of the four subfolders (`correct_ambiguous`, `correct_unambiguous`, `incorrect_ambiguous`, `incorrect_unambiguous`) to create more complete surveys, visualizing the differences in efficiency and behavior between the three algorithms in the different setups.

3.1.4 Survey generation

After the examples generation and the internal subdivisions, the images can be finally used as examples inside the survey. The survey is composed of n different forms, whose structure is described in Section 3.2. It has been decided to utilize multiple forms instead of a singular one, to increase the number of examples, while maintaining the forms short not to overwhelm or tire the users. Although the singular form approach would have been sufficient and less laborious to build for shorter and smaller surveys, it was essential to define specific standards in this occasion.

In the first place, **generate multiple forms** instead of a single one and **assign them randomly to every respondent**. Since Prolific does not permit the possibility to link different form to a survey, a whole new process is needed. The first problem is handled by the survey generator (Section 4.3), while the second by the web app (Section 4.4).

Secondly, **the randomness of examples and order of questions asked**. A simple approach already exists in Google Forms that simply shuffles the responses. The problem is that, in this case, each response indicates the index of the solution chosen and does not have an independent meaning. For example, given an image like Figure 3.2 where the possible responses are `#1`, `#2`, `#3`, and `None of them`, shuffling them does not change the counter-examples order they refers. Furthermore, the statistics generated on a single Google Form would result on less interesting and reliable evaluations.

Finally, **the generation of multiple surveys**, with (possibly) different examples, different amount of Google Forms, and different survey names. The reason why this feature has been implemented is because, other than the official survey generation, internal trials have been created and shared with research groups to test the operating and functioning of the evaluation environment.

3.2 Form structure

Here it is described the form structure, how it is composed and the decisions made to build it. The form has been built as a Google Form since it is free and widely used. It is composed by a static part, equal for all the surveys, and a dynamic portion: the examples.

The static part encloses different sections:

- the introduction of the experiment
- the consent form
- the survey instructions
- the end of survey instructions

The dynamic part instead is composed of 12 examples randomized, three for each group of the previous internal subdivisions. An example section is a multiple-choice (MC) question composed as follows:

- the title, written as "Example" followed by the index of the example
- the `.png` image of the example
- the question, "Which counter-example (s) is(are) more convincing for you?"
- the possible choices (`#1`, `#2`, `#3`, or `None of them`)

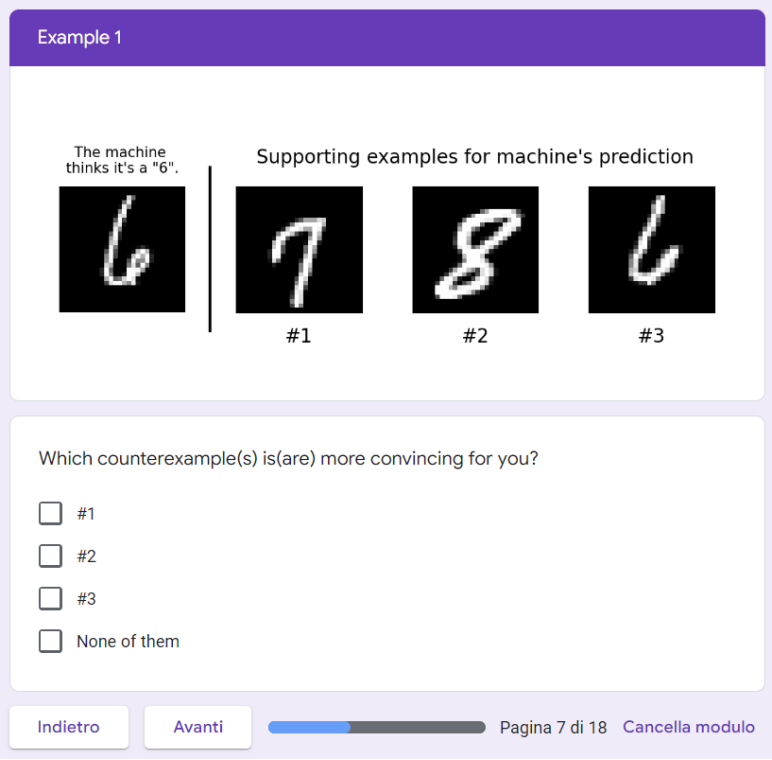

Figure 3.3: Structure example asked inside the Google Form

Utilizing a MC question instead of others is not straightforward. The single-choice question has no logical problems, but it does not permit the possibility of inserting more than one answer, which is essential. Assigning a confidence score for each image is a clever way to observe the case when two counter-examples convince the user, but one is more convincing than the other. The disadvantage here is a lack of clarity for the respondent given the personal question. Moreover, assigning a confidence score is more complex than deciding if one counter-example is relevant or not.

The MC question is therefore the best one in this situation, but has its own drawbacks. The respondent can commit two types of mistakes: skipping the question or selecting `None of them` along with another response. While the first is effortlessly implementable requesting the selection of at least one response to proceed with the questions, the second is not due to a lack of logic conditions complexity implementable in Google Forms. So, in the data gathering process, all the responses containing `None of them` and others have been invalidated.

4 Evaluation environment

This Chapter explains the evaluation environment and the ability to generate, process, distribute, evaluate and display statistics of a series of surveys.

Altogether it is summarized in 5 different modules:

1. The **supporting example creation tool** on top of the CINCER algorithm to generate the example images
2. A **survey generator** to create various surveys with the corresponding responses and links file
3. A **web app** to randomly access the survey generated from the previous algorithm
4. A **data gathering process** to handle the users' answers
5. A **statistics generator** to generate plots, namely visual statistics

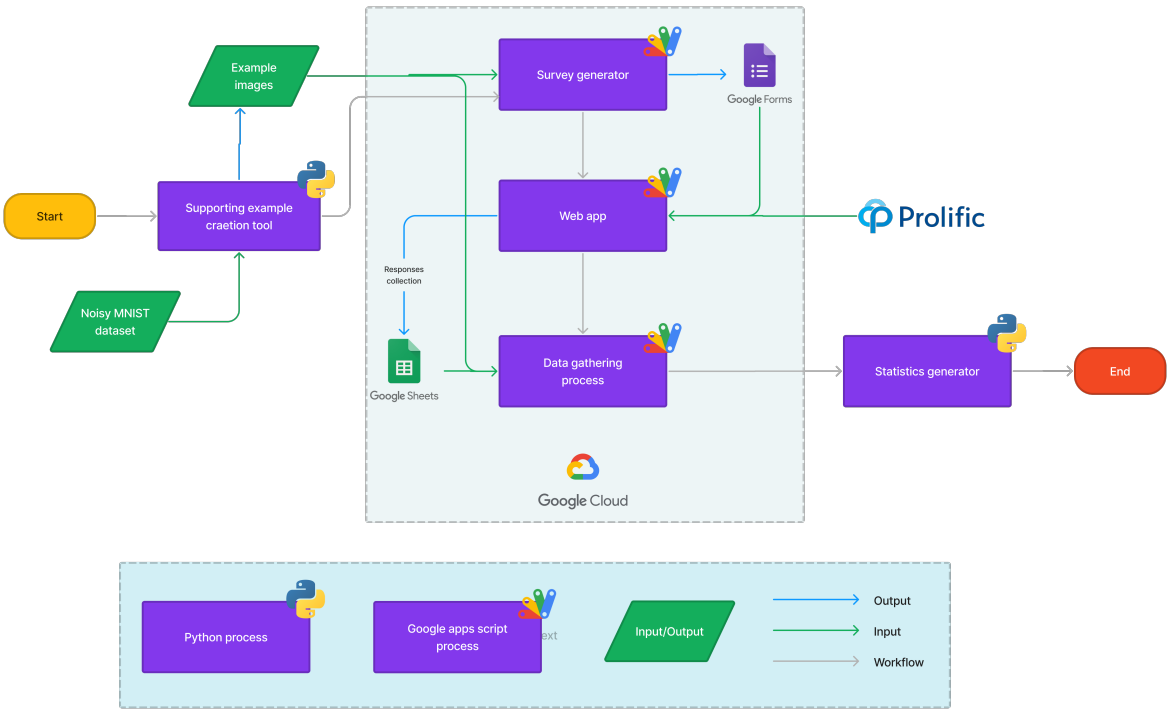

Figure 4.1: Evaluation Environment workflow

4.1 Programming languages, libraries, and platforms

The entire project has been built using Python and a proprietary language based on JavaScript, including ECMAScript 5 API, Google Apps Script (file extension `.gs`). Google Apps Script is a scripting platform developed by Google for light-weight application development in the Google Workspace platform, and it runs server-side on Google's infrastructure [3].

A `.gs` files can perform various actions on Google products such as Google Form and SpreadSheet. They can open, create, modify, and link many services correlated to a specific account Google and the relative Google Drive space. The whole project has been developed inside a UniTn Google Drive folder.

4.1.1 Data analysis tools

Numpy, Matplotlib, and Pandas are just a few of the libraries used inside the python scripts. Numpy adds support for large, multi-dimensional arrays and matrices and include an extensive collection of high-level mathematical functions to operate on these arrays. It has been used mainly in the data-gathering process discussed in Chapter 4.5.1, given the ease of import and export methods for CSV and XLSX files and the possibility to remove and convert DataFrame columns by default [8]. Matplotlib is a comprehensive Python library for creating static, animated, and interactive visualizations [6]. Lastly, Pandas offers data structures and operations for manipulating numerical tables and time series and is built on top of Numpy [9]. These open-source software libraries are widely used for data science/data analysis and machine learning tasks. Finally, file and folder handling libraries like `os` and `glob` have permitted creation, renaming, moving, and so on inside the project.

For the data-handling part, instead of a Python file I decided to use a Jupyter Notebook [4]. Jupyter Notebooks are widely used in Machine Learning, and Data processing since the code of a file is subdivided into smaller blocks. They structure the code in a modular way permitting to remove the execution of previous parts already coded, tested, and executed. For example, if a CSV file needs to be imported, cleaned, and saved into an XLSX file, to then process the latter, there is no need to reproduce the first step at every test of the second. Jupyter notebooks have been designed to be run in JupyterLab, the latest web-based interactive development environment for notebooks, but have been developed on a local machine.

4.1.2 Prolific

Ultimately, the experiment was carried on Prolific, a virtual platform that helps researchers recruit participants for their online research. It enables fast, reliable, and high-quality data collection by connecting diverse people worldwide while offering ethical pay to participants. Researchers create a study and upload it to Prolific. They estimate the time to complete the study, set a reward per hour, and add participant criteria. Since there was no possibility of uploading different surveys and assigning them randomly to users requesting to participate in the study, the web app discussed in Chapter 4.4 has been developed as a workaround [10].

4.2 Supporting example creation tool

The first block of the process is the Supporting example creation tool. The python module aims to generate the supporting example images (in `.png` format) used in the subsequent procedure. It has been written on top of the function `sample_counterexamples()` in the CINCER code repository [1]. The different output is described in Section 3.1.2, while here it is described the code.

4.2.1 Suspicious examples selection

The first step is to build the subsampled noisy data set from the original data set. Then the model is trained on the training set previously built for 100 epochs. After that, the selection of suspicious images takes place. As described in 2.3.2, the margin is the turning point for the selection, so it is used to split the compatible by the suspicious examples. The suspicious list is then used in two internal arrays, `uncertain_mistakes` and `certain_mistakes`, which are subsequently concatenated as the final suspicious examples `selected`. The first represent the top n images in which the model predicts a different class than the one assigned to the training set (that could be wrong) with high confidence, the second the top n pictures that has lower confidence. Since both of them are generated

from the same array, the suspicious examples can be duplicated on those lists. To have a greater pool of images as examples for the surveys, n has been decided to be the number of suspicious examples, therefore every image is taken twice by default. Even if this passage is useless in this application, usually the number of selected suspicious examples is lower than the one selected from the machine, resulting in the most certain and uncertain suspicious examples.

4.2.2 Image metadata

As seen in Figure 3.2, there should be no reference of which algorithm found which counter-example. For this reason, the metadata containing the details about the image has been stored in the title. The metadata of each image, which become helpful in the data-gathering step in Chapter 4.5.1, has the following format `%T%L%P_%A%T%L_%A%T%L_%A%T%L.png` where:

- `%T` is the actual label
- `%L` is the label annotated by the supervisor
- `%P` is the predicted label
- `%A` is the the first letter of the algorithm's name used to find the counter-example

Each aggregation of labels separated by an underscore "_" character refers to the image placed at the corresponding position. This strategy allows the metadata integration for the statistics generation part and hides it in the survey.

4.2.3 Internal subdivisions

Each group of images is then saved inside a folder named as two indexes separated by two underscores "__". The first is the index of the example inside the MNIST data set, and the second is the index of the list of the surrogate internal suspicious images list `selected`. As described in 3.1.3, the examples with the same suspicious image (same first index), will have different counter-examples associated.

Finally, instead of generating a simple collection of folders, each new folder is automatically saved in a `correct` or `incorrect` subfolder. The images where the true label corresponds to the predicted label are considered correct and saved in the related folder. The pictures where the true label conforms to the annotation and not the prediction are considered incorrect and stored in the related folder.

After the whole generation, a manual procedure is needed to further divide the folders into "ambiguous" and "unambiguous" from a human observer, based on the appearance of the shape of the suspicious image numbers.

4.3 Survey generator

The supporting example generation tool results are then used in the survey generator algorithm. The goal of the Google Apps Script is to generate a finite number of Google Forms linked to a Google Spreadsheet file, each connected to a different sheet of the file, to later obtain the answers from the users and store them in an organized and automated manner.

4.3.1 Initial idea

The initial idea was to generate a completely new Google Form for each user, to maximize the randomization and minimize research participants' biases. The issue using this methodology was the generation time, approximately 40 seconds. Various heuristics have been implemented to reduce or remove the waiting time for the respondent.

Firstly, **relying on the structure of a Google Form template** containing the static info required in every new form. The template details are clustered in sections (the building blocks of a Google Form): explanation of the experiment, consent form, execution instructions for the participants, and end of survey. The idea is to copy the default template to start from it the final Google Form generation, instead of begin from scratch every time.

Secondly, **postponing the entire generation** of the Google Form while the user reads the instructions and concludes the first questions. Since the Google Form can be edited while a user is

filling it, the idea could work unless they were quicker than the generation server-side. In that case, the user would have finished the survey without an answer to some of the final questions.

In summary, having a long waiting time for the user or a possible bugged product has led to another idea, namely the generation of a finite number of Google Forms and randomly redirect each user to one.

4.3.2 Survey internal structure

The survey generation is, in turn, composed of three submodules:

- **the forms generation**, which generates the Google Forms inside a `Forms` folder
- **the responses linking** from each form to a Sheet of a `Responses` Google SpreadSheet file
- **the Google Forms links generation** on a `Links` Google SpreadSheet file used in the Web app process to access random surveys

Forms generation

The form generation is the core submodule of the whole survey generator module. It starts creating a new Google Form and assigning some default settings to the file. The settings have various purposes, from simple visualization like color or progress bar add-on to more important ones like the `requireLogin`. The latter specifies if the user requires a Google profile to access the Google Form and, in this case, has been set to `not required`. Therefore, no prerequisites from the research participants are demanded other than a minimum age of 18 (checked in the approval consent section).

Sequentially, the generation of the static sections is needed. Since there is no waiting time for the respondents to the survey, the heuristic used in the initial idea was no more needed; a programmatically way of generation has been followed. The entire form generation process can now be integrated with some logic. Usually, one button (`Back` or `Next`) must be pressed to go to an adjacent section. Rather, indicate the acknowledgment or not of participation and the understanding of the experiment need to have different behavior than the default indeed. If the user does not indicate consent in the Acknowledgment section, a termination without consent section (without reward for the user, too) is displayed. Instead, in the experiment list rules section is required to select the toggle "I have understood" checkbox to start the experiment.

As previously said, one of the experiment goals is to create complete surveys and visualize the differences in efficiencies and behavior between the three algorithms (CINCER, 1-NN, and IF) on different types of examples. For that, the algorithm takes the folders created from the supporting example creation tool (supported by the manual division into "ambiguous" and "unambiguous") to display the images. For each group of folders, three distinct ones are selected, and for each, a random image is imported. Subsequently, the list of the selected image is shuffled to obtain randomization in the order of the groups. Afterward, given as input an image, an example section block is created.

To conclude, a simple static final section is added to end the survey, explaining the steps to submit the Google Form and acquire the payment from Prolific's platform.

Responses linking procedure

The responses linking procedure is responsible for linking every Google Form generated from the previous procedure to a Google Spreadsheet file, which will then be manipulated for the statistics generation. After creating the file, each Google Form needs to be linked to a Sheet of a Google Spreadsheet file and perform some operation on the corresponding Sheet. A mutual exclusion strategy is implemented to limit the access to the resource to only one modification at a time, using:

- `LockService.getScriptLock()` to obtain a lock object
- `lock.waitLock(waiting_time)` to wait for the release of the lock object
- `lock.releaseLock()` to release the lock and pass it to another thread

After renaming the Sheet to `Sheet #n`, where `n` is the index of the Sheet, during the linking phase, the responses are automatically added to the SpreadSheet file in the first line, preceded by a `timestamp` field. The application of this method is enough for most surveys since for every Google Form submitted, it automatically registers a line with the timestamp of the submission followed up by the responses selected from the user.

However, that is not the case. As previously said, the question of each example is "Which counter-example(s) is(are) more convincing for you?" and the possible answers are: `#1`, `#2`, `#3`, `None of them`, or a combination of them. There is no way to assign a value to the responses since the questions and answers do not have any kind relation. The responses are correlated to the image through the metadata information. For this reason, substituting the questions with meaningful values is required.

After the linking phase, the first row of the Sheet is overwritten. The image's title correspond to the relative question and the folder's title containing the image are concatenated to give meaning to the questions and add to relate to the image's folder.

For example, take into consideration Figure 4.2: on the left the starting image, on the right the produced one using (from left to right) 1-NN, CINCER, IF. They are both saved in the folder `Images/corr_unamb/470__0`. The string `868_188_C08_I68 470__0` is the resulting question generated.

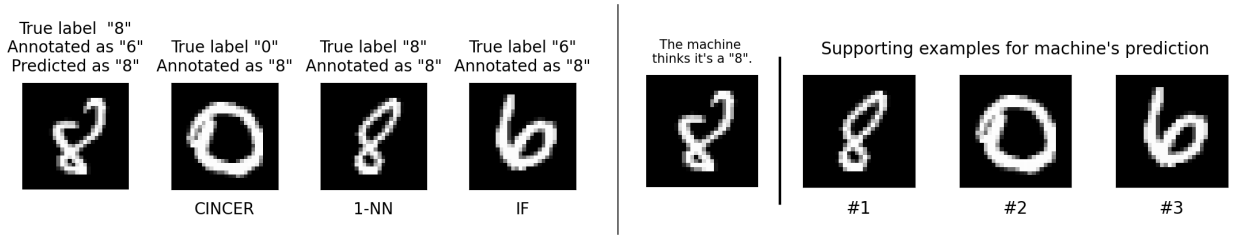

Figure 4.2: Example **Left**: the main image produced from CINCER **Right**: one of the six possible permutations generated from 4.3.2

Lastly, a counter for correctly submitted forms is added, replacing the `timestamp` column name label. So, after the question replacement, a "0" value is added as the first cell, and a trigger function automatically increases it at every valid form submission. The result is a live counter of correctly submitted results, valid for a quick understanding of the number of people previously submitted. It is a helpful value for the data gathering process discussed in Chapter 4.5.1.

SpreadSheet links generation

The SpreadSheet links generation is a process to speed up the access and reduce the delay of the web app execution explained in Chapter 4.4. The objective of this step is to automatically generate a SpreadSheet file incorporating the free access links of the Google Forms.

After the `Links` file creation and the shifting to the survey folder, the first column is filled with Forms access links. This step removes the otherwise constant link generations at every web app usage.

4.4 Web app to access a random survey

The web app goal is to redirect to a random Google Form from the `Links` file inside the survey folder for every applicant requesting access to the survey from the Prolific platform. After the deployment, the web application link is the one that will be inserted into the Prolific platform as the URL of the study.

Google Apps Script permits building user interfaces for a script to publish as a web app. A script can be published as a web app if it meets the following requirements:

- it contains a `doGet(e)` or `doPost(e)` function
- the function returns an HTML file

Once the web app is published, it can retrieve parameters, run some code and return an HTML file. In this case, it has been decided to utilize a `doGet(e)` function. It is needed to insert the name

of the survey experiment as a parameter into the URL, and only a GET request can do that (while in a POST request, the parameters are hidden and not integrable in the URL).

The process of this module is straightforward. Once the name of the experiment is extracted, the experiment's folder is found inside the Google Drive account using the name as objective key. For this reason, it is crucial in Chapter 4.3 to name the surveys with different names, to avoid anomalous behaviors. The Form generator module has been used only for a few survey creations; checking other surveys with the same name before the actual generation has not been implemented but could be done in a future version. After finding the relative `Links` folder, it only needs to read a random value from the links column to redirect the user to the relative Google Form. The issue is that it does not exist a Google Apps Script function that evaluates how many rows of a column are not empty.

Two workarounds are possible: reading every cell sequentially until an empty one is found or finding how many forms have been previously generated. The second alternative has been chosen, returning the number of files inside the `Forms` folder. Then, a random index from zero to the number of Google Forms is returned and the corresponding link retrieved. An elementary HTML page is generated, only containing a JavaScript `windows.location` method to redirect to the Google Form.

4.5 Data gathering and Statistics plots generation

This Chapter focuses on the data gathering process, including conversion of the SpreadSheets files and the data process, concluding with the statistics plots generation using MatplotLib.

4.5.1 Data gathering

At first, a Google Apps Script file receives the survey name in question to find the affiliated survey folder inside the Google Drive account. Then, following the same structure discussed in the previous Chapters, opening the "Forms" folder inside the current, it selects and converts the Sheets section inside the `Responses.xlsx` file into multiple standard `.csv` files. The name of each file is the corresponding one on the Sheet, generated in a directory called `CSV files` (neighbor of the current one) to provide a proper separation. It has been decided to use `.csv` files instead of other formats because the Pandas python library used to handle complex data formats is well optimized for this kind of data. Moreover, CSV files are supported by nearly all data upload interfaces and are heavily used in data processing and manipulation so that they can be readily available for future usages.

	A	B	C	D	E	F	G	H
1	12	Please indicate your consent before proceeding:	All clear?	828_I98_188_C68 329__31	315_135_C65_I65 857__21	118_I08_118_C28 659__4	226_C26_126_I16 1255__18	656_I76_166_C
2	5/10/2022 21:02	☑ I consent, begin the study	👍 I have understood	#2	None of them	None of them	None of them	#2
3	5/16/2022 16:02	☑ I consent, begin the study	👍 I have understood	#2	None of them	#2	None of them	#2
4	5/16/2022 16:03	☑ I consent, begin the study	👍 I have understood	#2	#1	#2	#1, #2	#2
5	5/16/2022 16:03	☑ I consent, begin the study	👍 I have understood	#2, #3	#3	None of them	#1	#2
6	5/16/2022 16:03	☑ I consent, begin the study	👍 I have understood	#2	#2	#1	#2	#2
7	5/16/2022 16:03	☑ I consent, begin the study	👍 I have understood	#2	#1	#2	#2	#2
8	5/16/2022 16:03	☑ I consent, begin the study	👍 I have understood	#2	#1	#2	#2	#2
9	5/16/2022 16:04	☑ I consent, begin the study	👍 I have understood	#2	#1	#2	#2	#2
10	5/16/2022 16:04	☑ I consent, begin the study	👍 I have understood	#2	#1	#3	#1	#2
11	5/16/2022 16:04	☑ I consent, begin the study	👍 I have understood	#2	#1	#2	#2	#2
12	5/16/2022 16:04	☑ I consent, begin the study	👍 I have understood	#2	#1	#2	#1, #2	#2
13	5/16/2022 16:07	☑ I consent, begin the study	👍 I have understood	#2	#1	#2	#1, #2	#2
14								

Figure 4.3: CSV example obtained from the data gathering process

4.5.2 Data manipulation

From now on, the Google Apps Script application terminates to switch to Python and his data science libraries. Every CSV file is imported and stored inside a for loop in a Pandas DataFrame. A DataFrame is the primary Pandas data structure, and it is two-dimensional, has labeled axes (rows and columns), and is excellent for data manipulation. Next, a data cleaning step is needed since only some of the corresponding columns of each DataFrame are important. For this reason, the first three columns are deleted, those representing `Timestamp`, `Consent approval`, and `Instruction understanding`.

The first value corresponds to the response submission time, which is useless for this experiment. The second fits the answer identifying the acknowledgment of the data management agreement, worthless because no one responded negatively to the question. Nevertheless, even if some respondents did,

the subsequent cells would have had an empty value, handled in a next step. The third matches the understanding of the instructions, but since it is required and there are no other options, it has a constant value between all responses.

Next, the DataFrames are concatenated in a single CSV file along the columns axis. The result is a series of columns with questions as header and the corresponding answers as values, placed back-to-back.

As the last and most crucial step, the responses need to assume a meaningful value to be successively used to generate statistics. Please note that, from the previous module, the saved responses were numeric indexes (`#1`, `#2`, `#3`) with the addition of `None of them`. The indexes in this state have different meanings based on the related question, so no global usage is possible. The DataFrame needs a normalization procedure, converting the responses indexes to the algorithm's initial letter they indicate. Furthermore, it is crucial to consider that the answers must have the same value when referring to the same choice. For example, let us assume two different questions where the algorithm order were `CINCER`, `IF` and `1-NN` and in another `IF`, `CINCER` and `1-NN`. In addition, a response `#1`, `#2` has been selected in both. Naively mapping the responses to `C`, `I` for the first and `I`, `C` for the second would result in two different values but same answer's type. So, in the case of multiple answers, a sorting function based on the ASCII table order is applied to circumvent the problem.

Once the resulting CSV has been processed, normalized, and saved, a splitting operation is carried out. The goal is to split it into six question subsets, namely `correct`, `incorrect`, `correct_ambiguous`, `correct_unambiguous`, `incorrect_ambiguous`, and `incorrect_unambiguous`, to generate the same statistics for each group. It is important to recall that the answers are composed of the question's metadata, concatenated to the index folder where the image has been picked. In addition, the images are already splitted into subfolders corresponding to each group. Firstly six arrays are generated, four picking all the indexes of one subfolder and two picking the indexes of two subfolders (correct and incorrect). Then, from the total CSV built in the previous step, four other DataFrames are produced and saved in the same `CSV files` folder. Here the data gathering and manipulation process end since all seven CSV files are available and ready to be used as inputs for statistics generations.

4.5.3 Statistics plots generation

The statistics plots generation part needs to input the five CSV files produced in the previous step to generate some plots. A plot is a state-based interface to a Matplotlib module which provides a MATLAB-like interface. Various plots can be used in Matplotlib, such as Line Plot, Countour, Histogram, Scatter, and 3d Plot. Fancy illustrations are not required in this case, so a simple histogram represents the analytics.

We focused on the **mean** of the possible choices for each CSV file. The possible choices were: a particular choice (`I`, `C`, `1`), a pair of choices, all of them or none of them. To associate each choice to a proper column label inside the plots, a static Python dictionary is used. The response is the key and the corresponding label the value. For example: `None of them:‘‘No method wins"` or `1:‘‘1-NN wins"`.

Next, for each column, the mean is computed, summed up and lastly divided by the number of columns, obtaining the total mean of each possible choice, for each group. In the end, the histograms are built using Matplotlib functions and saved in the `Plots` folder.

5 Results

In this chapter the results obtained from the Evaluation environment are exposed, described and commented. The experiment questions are:

1. Do counter-examples selected via IF, NN, or FIM satisfy desideratum D3 from Section 2.3.3? Is it clear *to the user* why an example is a counter-example for a particular example?

2. What are the algorithm selection's frequencies compared to the hypothesis in 5.1?

All the generated plots have the same structure, displaying the frequencies of each algorithm selection. They can be grouped in three categories, based on whether a user selects:

- "None of them" (no method wins)
- a specific algorithm
- two algorithms

The option where all three algorithms are selected simultaneously is not displayed since there were zero occurrences of this particular case.

5.1 Hypothesis

Before showing the plots of the algorithm selection's frequencies, it is important to highlight the assumptions made about the experiment's outcome.

1-NN should find the closest counter-examples for a human eye. The Nearest-Neighbor algorithm stores all available cases and classifies the new data on the basis of a similarity measure. The measures could be different depending on the goal of the algorithm, but in this case the analogy is based on the Euclidean distance image's pixels. Humans tend to assign a higher correlation to similar images rather than focusing on other types of resemblances (which could be more useful or meaningful to the machine).

CINCER and IF, instead, focus on finding counter-examples more useful to the machine that maximally supports the incoming examples, trading off similar appearance. The difference is that CINCER counter-examples are expected to be more interpretable, making it preferred compared to IF's selections, which are numerically unstable.

In summary, the algorithm's selection frequencies ascendant order is supposed to be 1-NN, CINCER and IFs.

5.2 Correct vs incorrect examples

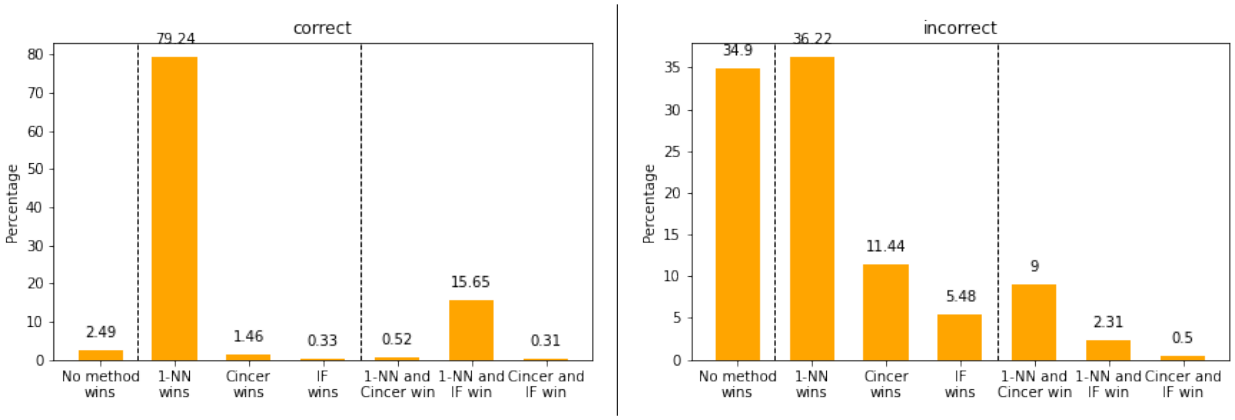

Figure 5.1: Algorithm selection frequencies on **Left**: correct examples, **Right**: incorrect examples

Figure 5.1 shows the algorithms selection on the correct vs incorrect examples. Both follows the assumptions perfectly (in the single choice group), where CINCER produces better counter-examples that the user considers more reliable than IF's and 1-NN wins in first place. In the double choice group, instead, 1-NN and IF wins in the correct plot, while 1-NN and CINCER wins in the incorrect one. The latter follows the assumptions even in the double choice group, whilst the first does not. The biggest difference between the two is in the "No method wins" bar, which is $\frac{1}{3}$ of the total answers in the incorrect case and basically absent in the correct one.

In the "correct" plot is clear that 1-NN outperforms the opponents 80% of the times. This indicates that, if the model correctly predict a counter-example, 1-NN is enough to satisfy the users. Instead, if the model incorrectly predict a counter-example (incorrect plot), 30% of the times no method work.

Two examples of this interesting case are shown in Figure 5.2, where every user selected "None of them". It is clear that no counter-example selected truly represents the suspicious example.

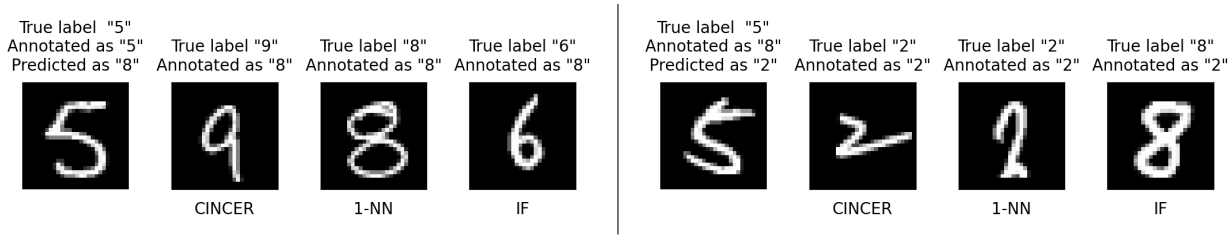

Figure 5.2: Incorrect examples where the option "None of them" won in every form

5.3 Ambiguous vs unambiguous examples

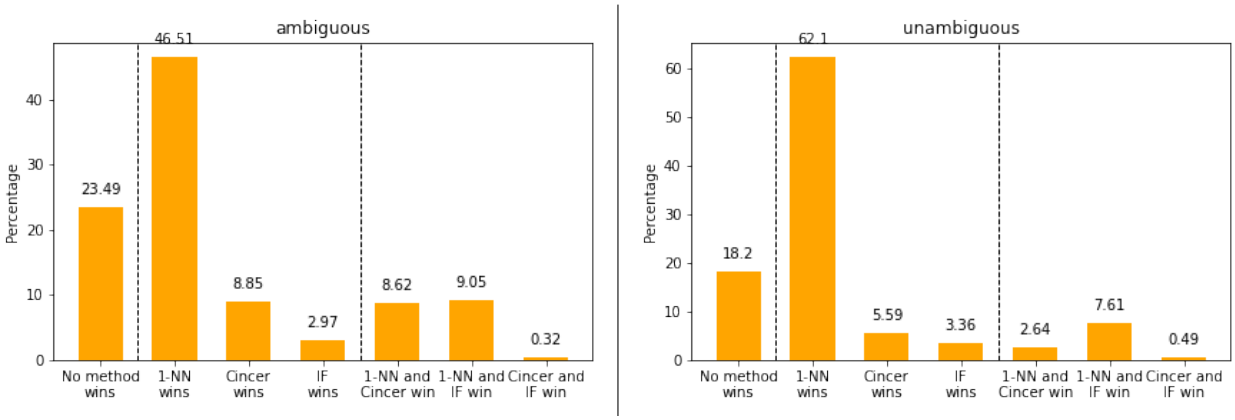

Figure 5.3: Algorithm selection frequencies on **Left**: ambiguous examples, **Right**: unambiguous examples

Figure 5.3 shows the selection of the algorithms in ambiguous vs. unambiguous examples. The impact in this setting is much less compared to the correct vs. incorrect comparison. 1-NN seems to be less persuasive on the ambiguous examples, but in general the frequencies are similar. Both follow the assumptions in the single choice group, while in the double choice 1-NN and IF wins over 1-NN and CINCER.

5.4 Experiment answers

To answer to the first question, the focus must be on the algorithm frequencies of each particular subset. The counter-example is selected by following the algorithm's logic path, aiming to exploit the result as the best advantage to cleaning the data set and performing better in the future. The problem is that a certain counter-example could be counter intuitive or wrongly associated to the starting image from the user perspective. Figures 5.3 and 5.1 show that, generally speaking, approximately four over five times at least one algorithm finds a clear counter-example from the user perspective.

The hypothesis, instead, are generally followed in each subset. 1-NN is always the algorithm chosen the highest number of times, CINCER the second, and IF the last. In the double choice subgroup,

instead, the hypothesis are only sometimes followed.

6 Final considerations

In this thesis, I have discussed how I approached the challenge of developing an evaluation environment to compare CINCER, 1-NN and IF finding the "best" counter-example given a suspicious example with real users. Alongside the specification, I provided detailed insights on the development process, pointing to the major challenges tackled and the solution that I have implemented. At the current moment, the Evaluation Environment is able to:

- find several suspicious MNIST examples with correlated counter-examples
- generate multiple images for the survey
- generate multiple forms including different examples, producing a survey
- redirect every user to a random form from the same link
- automatically store the collected data returning meaning to questions where the metadata is not explicitly displayed in the demand
- collect the data in CSV format
- generate statistics

6.1 Future work

The evaluation environment has been implemented specifically to evaluate CINCER versus 1-NN and IF on the MNIST data set. On top of that, no experimental psychologist has been consulted about the survey structure and types of questions. Consequently, various upgrades can be made to the process, from generalizing it on other types of surveys to enhance the quality and trustworthiness of the forms.

- *Generalize the Survey Generator tool.* To the current state, the process is not general and cannot be used to build other types of surveys or include different questions. However, the kind of survey implemented could be useful on many other structured and professional surveys with specific and not directly available requirements like ours. For example, generating different surveys with mixed or different questions and randomly assigning them to several survey respondents could be a feature interesting on surveys about any topic. Another feature implemented that could be useful in general surveys is the usage of metadata as the title of an image. In all those cases where the responses refer to the image displayed (and not the text question), the linked Spreadsheet that gathers the responses has meaningless answer-response correlation. As a workaround, the metadata of the image (written in the title) could replace the fictitious question, returning meaning on the collected data.
- *Generalize the Evaluation environment.* Other than only exploiting the Survey Generator tool 4.3 for advanced general surveys, the whole process could be generalized for other kinds of evaluations. Skipping the Supporting example creation tool 4.2, far too specific and complex for generalization, given some data set of images structured in a particular way, the Google Form static sections and other survey specifications, the process could be reused for several assessments. At the moment the modules are independent and there is no kind of global workflow, but with the right goals and requirements it could be fully automated for several types of evaluations.
- *Inclusion of experimental psychologists.* Lastly, the survey could be restructured by experimental psychologists to improve the reliability of the test and include a scientific psychological approach.

Bibliography

[1] Cincer github repository. https://github.com/abonte/cincer.

[2] Explainability and auditability in ml: Definitions, techniques, and tools. https://neptune.ai/blog/explainability-auditability-ml-definitions-techniques-tools. last access 29/06/2022.

[3] Google apps script - wikipedia. https://en.wikipedia.org/wiki/Google_Apps_Script. last access 29/06/2022.

[4] Jupyter - official site. https://jupyter.org/. last access 29/06/2022.

[5] A little talk on label noise. http://knowdive.disi.unitn.it/2018/09/a-little-talk-on-label-noise/. last access 29/06/2022.

[6] Matplotlib - official site. https://matplotlib.org/. last access 29/06/2022.

[7] The mnist database. http://yann.lecun.com/exdb/mnist/.

[8] Numpy - official site. https://numpy.org/. last access 29/06/2022.

[9] Pandas - official site. https://pandas.pydata.org/. last access 29/06/2022.

[10] Prolific - official site. https://www.prolific.co/. last access 29/06/2022.

[11] Samyadeep Basu, Philip Pope, and Soheil Feizi. Influence functions in deep learning are fragile, 2020.

[12] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv: Machine Learning*, 2017.

[13] Benoit Frenay and Michel Verleysen. Classification in the presence of label noise: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 25(5):845–869, 2014.

[14] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions, 2017.

[15] Piyawat Lertvittayakumjorn, Lucia Specia, and Francesca Toni. FIND: Human-in-the-Loop Debugging Deep Text Classifiers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 332–348, Online, November 2020. Association for Computational Linguistics.

[16] Christoph Molnar. *Interpretable Machine Learning*. 2 edition, 2022.

[17] Emanuel Slany, Yannik Ott, Stephan Scheele, Jan Paulus, and Ute Schmid. Caipi in practice: Towards explainable interactive medical image classification. 04 2022.

[18] Stefano Teso, Andrea Bontempelli, Fausto Giunchiglia, and Andrea Passerini. Interactive label cleaning with example-based explanations. *arXiv preprint arXiv:2106.03922*, 2021.

[19] Stefano Teso and Kristian Kersting. Explanatory interactive machine learning. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, AIES '19, page 239–245, New York, NY, USA, 2019. Association for Computing Machinery.

[20] Hugo Zylberajch, Piyawat Lertvittayakumjorn, and Francesca Toni. HILDIF: Interactive debugging of NLI models using influence functions. In *Proceedings of the First Workshop on Interactive Learning for Natural Language Processing*, pages 1–6, Online, August 2021. Association for Computational Linguistics.