UNIVERSITAT DE BARCELONA
Facultat de Matemàtiques i Informàtica

DOBLE GRAU DE MATEMÀTIQUES I ENGINYERIA INFORMÀTICA

Treball final de grau

CONFORMAL PREDICTION AND UNCERTAINTY QUANTIFICATION IN RECOMMENDER SYSTEMS

Autor: Roberto Alvarado Chamatrin

Director: Dr. Vitrià Marca, Jordi
Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, June 11, 2024

Abstract

This thesis investigates the lack of explainability in machine learning models, particularly focusing on mitigating this issue by measuring model uncertainty and adjusting outputs using conformal prediction. Conformal prediction provides a set of possible outcomes, backed by statistical analysis, to ensure a high confidence level in the predictions. The approach will be demonstrated across three domains: classification with the MNIST dataset, regression for estimating California real estate prices, and recommender systems. In recommender systems, the method will account for varying levels of uncertainty in user preferences, ensuring broader recommendation sets for users with high uncertainty and narrower sets for more predictable users.

Resum

Aquesta tesi investiga la manca d'explicabilitat en els models d'aprenentatge automàtic i se centra especialment en mitigar aquest problema mesurant la incertesa del model i ajustant la sortida del mateix utilitzant la predicció conformal. La predicció conformal proporciona un conjunt de possibles resultats, recolzats per anàlisis estadístiques, garantint un alt nivell de confiança en les prediccions. Aquest enfocament es demostrarà en tres dominis: classificació amb el conjunt de dades MNIST, regressió per estimar els preus de cases a Califòrnia i en sistemes de recomanació. En aquest últim, el mètode considera diferents nivells d'incertesa en les preferències dels usuaris, assegurant conjunts de recomanacions més amplis per als usuaris amb alta incertesa i conjunts més estrets per als usuaris més predictibles.

Acknowledgements

During the elaboration of the thesis, I got help from several people to whom I dedicate this section.

To Jordi Vitrià, my thesis supervisor, for proposing such an intriguing topic and always being there to provide guidance during difficult times. His expertise in the field is evident, and it has been a pleasure to attend his lectures and work alongside him.

To Paula Vilà, for the mutual support throughout these months of hard work and challenges.

To the rest of my friends, for being there when I needed encouragement and for providing moments of laughter during this intense journey.

Contents

1 Introduction

Artificial intelligence has undoubtedly reshaped the way we operate as a society. Tasks that were previously performed by humans are now efficiently handled by computers. More precisely, machine learning techniques enable computers to learn tasks by utilizing input data and, often, the expected output, without requiring explicit instructions as traditional algorithms do. Through this process, computers learn from the data to find a mathematical model that best represents the underlying patterns, which can subsequently be applied to automate the task.

This convenience comes with a cost, though. Machine learning models tend to behave as "black boxes": they receive input data and generate predictions without revealing the decision-making process. Even the AI/ML engineers who develop these models may not fully understand its internal functioning. This opacity raises serious challenges, especially as models can be used in sensitive applications such as determining mortgage eligibility or medical diagnosis.

For instance, a model that predicts whether someone should be granted a mortgage may inadvertently perpetuate discrimination and inequality present in historical data, denying mortgages based on race, gender or other unfair reasons[1]. This is known as a bias. Similarly, medical diagnosis being performed by a model which lacks the explainability in its decision-making process could incorrectly diagnose a patient as healthy when they are, in fact, ill, potentially endangering their life.

This thesis will address the second issue we stated, caused by the lack of explainability in machine learning models. The goal is to mitigate it by measuring the uncertainty of the model and adjusting its output. Instead of providing point (single-value) predictions, the model will provide a set of possible outcomes using a method called "conformal prediction". This set will be backed by statistical analysis, ensuring with high confidence (e.g., 90%), that the true prediction lies within that set. The set size will vary depending on the uncertainty of the model, being larger for predictions where the model is uncertain.

We will start exploring conformal prediction by addressing a classification problem using the MNIST dataset[2], and then jump into the regression problem aimed at estimating real estate prices in California.

Finally, we will address recommender systems. While conformal prediction has primarily been applied to classification and regression problems, recommender systems also inherently involve uncertainty. For instance, when recommending content to a new user on a streaming platform, the model's uncertainty will be high due to the lack of data on the user's interests. In contrast, the uncertainty will be significantly lower for a user who has watched hundreds of hours of content on the platform, as their personal interests are pretty much well known.

To achieve it, we will first try to quantify the uncertainty of a recommender system, then apply conformal prediction to ensure that the recommendation set is larger for users with a high level of uncertainty and smaller for predictable users.

2 Scientific background

Before starting to implement the paradigm we have introduced and will describe comprehensively later, we will examine the machine learning concepts necessary for the development of the thesis.

2.1 Machine learning basics

When solving a problem via machine learning, we look for a model that can accurately represent the underlying process and make reliable predictions based on input data. This involves leveraging relationships within the data that effectively solve the task at hand. In other words, the model learns from the data to solve the problem, generalizing the solution from the ground up and inferring the solution for new data.

Definition 2.1. A **model** is a mathematical representation that is trained on a dataset to make predictions or decisions without having explicit instructions on how to perform the task.

There are several ways to train a model. Some of the most used ones are:

- Supervised learning: the model is trained on a labeled dataset, meaning that each training data point is paired with an output label. The objective is to learn a mapping from inputs to outputs that can then be used to predict the labels for new, unseen data.
- Unsupervised learning: the models trains on unlabeled data, trying to learn the underlying structure or distribution in the data.
- Semi-supervised learning: a combination of the two above. As labeled data can be difficult to obtain, only a subset of all the training data is labeled.
- Reinforcement learning: the entity that tries to achieve the goal is called an agent. It learns to make decisions by performing actions in an environment, receiving rewards or penalties on that action and seeking to maximize the reward over time.

In this thesis, the models we are working with are use supervised learning so we will focus on it from now on.

In supervised learning, the model is fitted on labeled data. Each input data used for training has its corresponding label, which is the correct output. Given a labeled dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, f tries to accurately map the given inputs $\mathbf{x}_i$ to y_i.

For $i = 1, \ldots, n$ the vector $\mathbf{x}_i$ is called the **feature vector** for the i-th sample and y_i is the **label** for the i-th sample.

As such, we can represent the entire dataset in a more compact way as (X, y):

$$X = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{pmatrix} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{pmatrix}, \quad y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

where X is the feature matrix and y is the label vector. Each row in X corresponds to a data point or sample and the same row in y is its label.

For instance, the goal is to minimize the discrepancy between the true value y_i and the prediction $f(\mathbf{x}_i)$ given by the model.

Definition 2.2. A **parameter** is a value in the model that modifies the output of the function and is adjusted during the training process. We can represent the vector of parameters of a function as $\mathbf{w} = (w_1, \ldots, w_k) \in K$, where K is a field that is usually $\mathbb{R}^k$. The letter w is used as parameters are also called **weights**. As they are initially unknown, they can be initialized with random values.

Example 2.3. Imagine we are looking for a model that predicts house prices based on their square meters. We can define a very simple linear regression model $f : \mathbb{R} \to \mathbb{R}$ where

$$f(x) = w_1 x + w_0.$$

x are the square meters and $\mathbf{w} = (w_0, w_1)$ are the parameters. w_1 represents the slope of the function and w_0 is the **bias**.

A bias is any parameter independent of the input features. It's called like that because it shifts the predictions of the model regardless of the inputs. In this case, it represents the fact that, even for the smaller houses, there is always a starting fixed price w_0 (assuming it's greater than 0).

Definition 2.4. Given a labeled dataset (X, y), the **loss function** is a positive function that measures the discrepancy between the true values y and the predictions $f(X)$, for a fixed vector of parameters $\mathbf{w}$. We write it as $L(y, f(X; \mathbf{w})) = L(y, f_{\mathbf{w}}(X))$.

Example 2.5. A possible loss function is the Mean Squared Error:

$$\text{MSE}(y, f_{\mathbf{w}}(X)) := \frac{1}{n} \sum_{i=1}^{n} (y_i - f_{\mathbf{w}}(\mathbf{x}_i))^2$$

i.e., the mean of the squared differences between each true value and the prediction. The differences are squared so that the loss function is always positive, and the closer it is to 0, the better the model f for that set of data, as the errors are, on average, smaller. This loss funtion is widely used in regression problems.

As the objective is to minimize the loss function by adjusting the parameters using the training data, any machine learning problem is essentially an optimization problem, i.e., finding the minimum of the loss function.

Definition 2.6. An **optimizer** is an algorithm that updates the model parameters to minimize the loss.

One of the most famous optimizers is **Gradient Descent**, which computes the gradient of the loss function using the entire training dataset of n samples and adjusts the parameters by subtracting the gradient times a factor. This algorithm is applied iteratively:

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \alpha \cdot \nabla L(y, f_{\mathbf{w}^{(i)}}(X))$$

where (X, y) is fixed and is the entire dataset, i is the iteration index, $\mathbf{w}^{(i)} = (w_1, \ldots, w_k)^{(i)}$ is the vector of parameters in that iteration, α is the learning rate and ∇L is the gradient of the loss function. Each iteration is called a **step**.

Definition 2.7. Given $f : \mathbb{R}^n \to \mathbb{R}$ differentiable at point p, the **gradient** $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$ at point p is the vector of its partial derivatives at p:

$$\nabla f(p) := \begin{pmatrix} \frac{\partial f}{\partial x_1}(p) \\ \frac{\partial f}{\partial x_2}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \end{pmatrix}$$

From the definition, it follows that

$$\nabla L(y, f_{\mathbf{w}^{(i)}}(X)) = \begin{pmatrix} \frac{\partial L}{\partial w_1}(y, f_{\mathbf{w}^{(i)}}(X)) \\ \frac{\partial L}{\partial w_2}(y, f_{\mathbf{w}^{(i)}}(X)) \\ \vdots \\ \frac{\partial L}{\partial w_k}(y, f_{\mathbf{w}^{(i)}}(X)) \end{pmatrix}$$

Remark 2.8. As the definition says, in order to compute the gradient of the loss function, it must be differentiable at that point. For instance, it must be chosen having this condition in mind. MSE $\in \mathcal{C}^\infty$, so it can be used without any issue.

Definition 2.9. A **hyperparameter** is a parameter that defines the behaviour of the learning process and is set prior to training. As parameters are adjusted automatically during the training, this definition is needed to distinguish those parameters from the ones that are specified before the training begins.

Intuitively, the gradient points to the direction where the function grows steeper from the given point. For instance, moving the opposite way decreases the value of the function. This behaviour is informally described as "a ball rolling down the slope" (neglecting acceleration). α is a hyperparameter called **learning rate**, and it adjusts the size of the steps taken towards the minimum of the loss function. Choosing an appropriate value is important as it being too high can cause an oscillating behaviour that doesn't converge, while it being too low can make the convergence very slow and even get stuck in suboptimal values.

Another problem is that, at each step, the entire dataset has to be evaluated in order to get the output of that iteration. This can get computationally expensive. It's worth noting that the dataset can have a very high cardinality, each feature vector can be high-dimensional and, in turn, the parameter vector, so using GD is often not feasible. For this reason, an alternative called **Stochastic Gradient Descent** is used instead. The dataset is shuffled randomly, and for each iteration, one of the n samples is chosen. This is done until all the samples are used. One complete run on the entire dataset is called **epoch**. After each epoch, the dataset is shuffled again and the process continues.

This alternative is much cheaper computationally. Not only that, but although it may take more iterations to converge to a minimum, this approach introduces noise that can help escape local minima and explore the parameter space more thoroughly, potentially leading to better generalization and better final models. The algorithm is as follows:

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \alpha \cdot \nabla L(y_i, f_{\mathbf{w}^{(i)}}(\mathbf{x}_i))$$

where $(\mathbf{x}_i, y_i)$ is a unique data point chosen randomly at the current epoch, i is the iteration index, $\mathbf{w}^{(i)} = (w_1, \ldots, w_k)^{(i)}$ is the vector of parameters in that iteration, α is the learning rate and ∇L is the gradient of the loss function.

In practice, a mixed technique called Mini-Batch Gradient Descent is used, which shuffles the dataset and makes batches of size s. For each iteration, the gradient is computed using the data points of one of the batches. When all the batches are used and the epoch is completed, the dataset is re-shuffled and the process continues. Note:

- If $s = 1$, it's the same as SGD and for every epoch n iterations can be done.
- If $1 < s < n$, $\lceil \frac{n}{s} \rceil$ iterations can be done for every epoch.
- If $s = n$, one epoch equals one iteration.

Another optimizer that will be used in this thesis is Adam (Adaptive Moment Estimation). Without going into much detail, it has the advantage that it computes learning rates for each parameter and it adds momentum, which helps accelerate the optimizer in the relevant direction. This is done by maintaining an exponentially decaying average of past gradients (first moment) and past squared gradients (second moment). It can also be seen as "a ball rolling down the slope", but this time with gravity acceleration as well.

The iterative process has to stop at one point. One or more of these conditions are used for it:

- A certain number of epochs is reached.
- After a certain number of epochs, the difference between the current loss and the previous one doesn't reach a certain threshold called **tolerance**, meaning it isn't very significant. It can also be defined that this condition must not be met a consecutive number of times. This number is called **patience**.

During training, a common issue is that the model learns the training data too well, including its noise and outliers. This phenomenon is called **overfitting** and results in a model that performs exceptionally well on the training data but poorly on new, unseen data, not generalizing properly. It occurs when a model is too complex, having too many parameters relative to the number of observations. The opposite of it is **underfitting**, and happens when the model hasn't been trained enough and doesn't perform well on the training data nor on new, unseen data.

In order to detect overfitting, when given a dataset (X, y), there is a portion of it (typically from 10 to 30%) that isn't used during training and is reserved to evaluate the loss after every epoch on it as well. Such a subset is called **test set**[1], and we will write it as $(X, y)_{\text{test}} = (X_{\text{test}}, y_{\text{test}}) \subset (X, y)$. The data used for training will be written as $(X, y)_{\text{train}} = (X_{\text{train}}, y_{\text{train}}) \subset (X, y)$. A plot of the loss function for the training and test sets during overfitting looks like this:

[1] Actually, such a set is called **validation set**, while a test set is completely unseen during the training. As no such set is used in this thesis, we will simply use the term "test set" for the validation one, as they are sometimes used interchangeably.

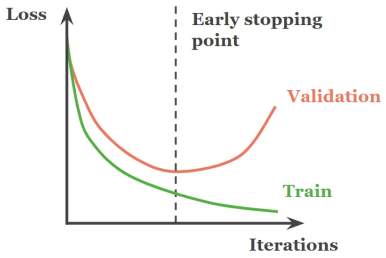

Figure 1: Plot of the loss function for training and test sets

To mitigate this, several techniques can be applied. The most straightforward one is stopping the training when performance on test data starts to degrade, called **early stopping**. Other techniques like regularization (lasso, ridge) can be used, which penalize overly complex models by adding the complexity of the model to the loss function.

2.2 Classification

One of the problems that machine learning tries to solve is classification, which is a type of supervised learning where the model learns to classify input data into one of the discrete labels or categories defined.

Given a labeled dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, y_i always falls into one of the K possible classes, which are represented by an integer:

$$y_i \in \{0, \dots, K-1\} \subset \mathbb{Z}, \ \ \forall i = 1, \dots, n$$

There are various models that deal with classification, such as Logistic Regression or Support Vector Machines. We focus on Neural Networks as that's the one we will use later, even though they are not exclusively used for classification.

Neural networks are inspired by the human brain. They consist of interconnected nodes organized in layers. Nodes receive input data and pass it to the next neuron via an edge.

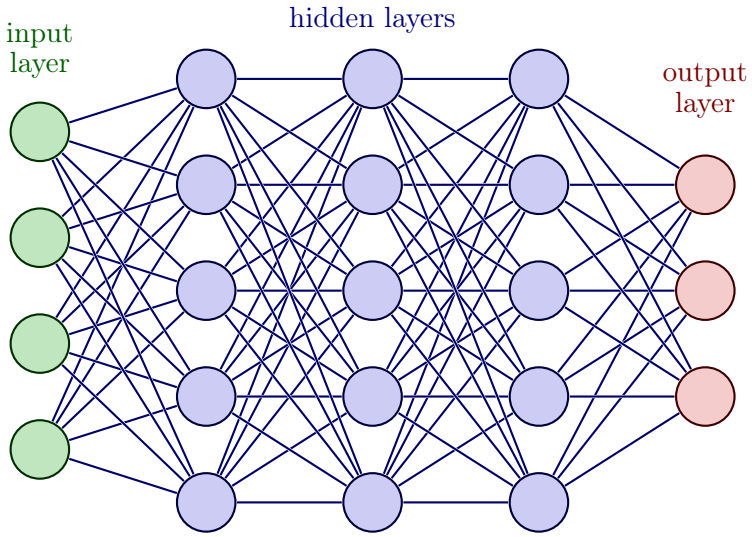

Figure 2: A neural network

In each neural network, there is an **input layer**, which receives the input (a feature vector), and an **output layer**, which returns the output. Optionally, there are **hidden layers** that add complexity to the neural network. Each edge has a **weight** associated to it, and each neuron has a **bias**. Each neuron that receives n inputs gets multiplied by the weight on the edge. All the inputs are summed and the bias is also added.

With the concepts we defined for now, all the neurons have a linear behaviour. With this lack of non-linearity, it can be proved that any such neural network, no matter the number of hidden layers, can be combined into a single layer.

Definition 2.10. An **activation function** is a non-linear function $f : \mathbb{R} \to \mathbb{R}$ that transforms the weighted sum of inputs to a neuron into an output signal. They are essential as this non-linearity is what makes neural networks capable of learning complex patterns. It's called like that because it mimics the biological behaviour of an activated/non-activated neuron, making its output relevant/non-relevant.

With the addition of the activation function, we can now formulate the output of a neuron j in layer l as follows:

$$a_j^{(l)} = f(\sum_{i=1}^{n} w_{ij}^{(l-1)} a_i^{(l-1)} + b_j^{(l)}) \tag{1}$$

where $w_{ij}^{(l-1)}$ is the weight connecting each neuron i in layer $l-1$ to neuron j in layer l, $a_i^{(l-1)}$ is the output of the neuron i in layer $l-1$ and $b_j^{(l)}$ is the bias for neuron j in layer l.

Theorem 2.11 (Universal Approximation Theorem)**.** *A neural network with at least one hidden layer of a sufficient number of neurons and a non-linear activation function can approximate any continuous function to an arbitrary level of accuracy.*

This fundamental theorem reveals the great power of neural networks and why they are widely used.

In neural networks, two key processes occur alternatingly:

- **Forward propagation** is the process by which input data is passed through the network to produce an output. Refer to equation 1 for more information.
- **Backpropagation** updates the weights and biases in the neural network to minimize the error in the predictions. This involves calculating the loss and its gradients for each parameter, using the chain rule of calculus. The loss propagates backwards through the network from the output layer to the input layer. This is used by the optimizer of choice to update the parameters. Because of this process, differentiability in neural networks is very important to be taken into account when choosing an activation function.

The choice of the activation function has evolved over time. The sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

has been the most popular as it accurately resembles the activation behaviour, because $\mathrm{Im}(\sigma) = (0, 1) \subset \mathbb{R}$ and the images quickly jump from 0 to 1:

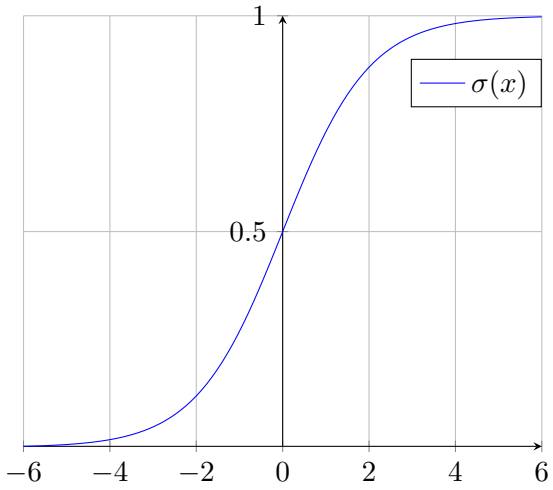

Figure 3: Plot of the sigmoid function

Moreover, it's differentiable in $\mathbb{R}$, so it behaves well when using gradient techniques to update the parameters. Nowadays, though, it's usually used only when the output has to be between 0 and 1, e.g., in the output layer for binary classification problems. The function "concentrates" in a very small range and for the rest of the domain, the slope is very close to 0. This leads to the **vanishing gradient** problem, dealing with gradients very close to 0, making the convergence very slow and the model perform poorly. Furthermore, additional problems as the **machine epsilon** (ε) (the limit to the amount of decimals that can be represented in a computer) can cause the gradient calculation to be imprecise.

For these reasons, in the decade of the 2010s, the **Rectified Linear Unit (ReLU)**[3] was introduced proving that it's a much better alternative:

$$\text{ReLU}(x) = \max(0, x)$$

This function performs the activation of neurons without vanishing the gradient, and is computationally efficient as the gradient is very easy to compute:

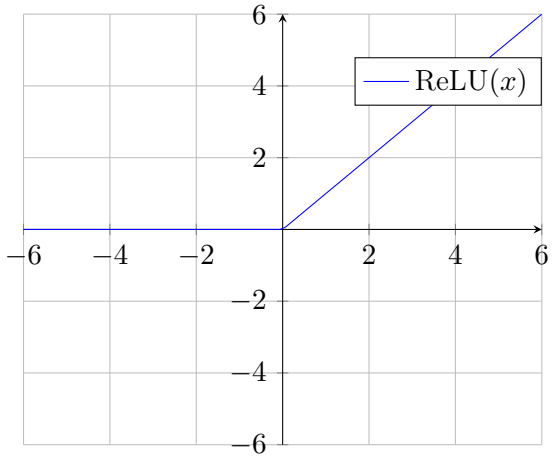

Figure 4: Plot of the ReLU function

The function isn't differentiable at 0, but it's as simple as defining the derivative at

that point as 0:

$$\text{ReLU}'(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$

Another widely used activation function is the **softmax function**, often used as the last activation function of a neural network used for classification. It's used to normalize the output and resemble a probability distribution over predicted output classes. Given a vector $\mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$, where $K \geq 1$, the softmax function $\sigma : \mathbb{R}^K \to (0,1)^K$ is defined as

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

The resulting vector $(\sigma(\mathbf{z})_1, \ldots, \sigma(\mathbf{z})_K)$ resembles the probability of each class, even though it's not calibrated as we will see later.

As for loss functions, the one we will use for classification is cross-entropy:

$$\text{Loss} = -\sum_{i=1}^{n} y_i \log(p_i)$$

where y_i is the binary indicator (0 or 1) if class label i is the correct classification, p_i is the predicted probability of class i and n is the number of classes.

An important thing to note is that softmax is used to normalize the outputs instead of a normal division of each output by the sum of them. The reason for this is that softmax "behaves better" when computing the gradient, reducing the likelihood of a vanishing gradient.

Let's see a couple of quick definitions, as the neural networks that we are working with in this thesis are of this type:

Definition 2.12. A **dense neural network** or **fully connected network** is an artificial neural network where each neuron in one layer is connected to every neuron in the subsequent layer.

Definition 2.13. A **feedforward neural network** is a neural network where the information only moves forward: from the input nodes, through the hidden nodes and to the output nodes. There are no cycles or loops in the network, meaning each layer only receives inputs from the previous layer and sends outputs to the next layer.

2.3 Regression

Another typical problem in machine learning is regression, a type of supervised learning technique used to predict continuous numerical values based on input features.

Given a labeled dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, $y_i \in \mathbb{R}$ is the continuous numerical label to be predicted.

Typical models that deal with regression are Linear Regression, Support Vector Regression and Neural Networks. Let's see the Gradient Boosting Regressor as it's the one we use later in this thesis.

Gradient Boosting Regressor (GBR) is an ensemble learning technique, meaning it combines the predictions of several base estimators (typically decision trees) to improve

overall predictive performance. The base estimators are weak prediction models, i.e., models that make very few assumptions about the data. The "boosting" part of the algorithm refers to the iterative process of training a sequence of models, each one correcting the errors of its predecessor. In each stage a regression tree is fit on the negative gradient of the given loss function. The optimizer used is, for instance, Gradient Descent.

The loss function is normally the Mean Squared Error, which we have already seen:

$$\text{MSE}(y, f_{\mathbf{w}}(X)) := \frac{1}{n}\sum_{i=1}^{n}(y_i - f_{\mathbf{w}}(\mathbf{x}_i))^2.$$

Another common loss function is RMSE, which is simply

$$\text{RMSE}(y, f_{\mathbf{w}}(X)) := \sqrt{\text{MSE}(y, f_{\mathbf{w}}(X))}.$$

2.4 Recommender systems

The problem of recommendation is another task that machine learning is dedicated to solve. In the realm of information retrieval, recommenders alter the way by which users find content. Rather than needing to actively look for it through manual search queries or browsing, recommenders adopt a proactive approach, delivering personalized suggestions for each user's unique preferences and interests[4].

These models analyze vast amounts of user data, ranging from browsing history and purchase behaviour to explicit preferences and ratings. By processing this info, recommendation systems gain valuable insights, allowing them to curate recommendations that are highly relevant and likely to resonate with each user.

We will focus on a type of models called Factorization Machines[5]. They are particularly useful for handling sparse datasets with high-dimensional feature spaces, often encountered in tasks like recommendation systems, prediction, and ranking. They were introduced by Steffen Rendle in 2010[6] and are known for their ability to model interactions between variables effectively.

In recommendation, data is sparse because the availability of it, which is typically user-item interactions or ratings, is scarce relative to the total number of possible interactions or items. In real-world scenarios, there can be millions of users and the same vast magnitude of items available for recommendation.

Furthermore, the distribution of user-item interactions often follows a **long-tail** pattern, where a small number of popular items receive a large proportion of interactions, while the majority of items receive relatively few interactions. This results in sparse data, as many items have limited or no interaction history.

New users or items, which lack historical interaction data, contribute to data sparsity. Recommendation systems face difficulties in making accurate predictions for these new entities until sufficient data is available to model their preferences effectively This is known as the **Cold Start Problem**[7].

The model structure of a factorization machine is as follows: the prediction $f(\mathbf{x})$ for a given input vector $\mathbf{x} \in \mathbb{R}^n$ is given by:

$$f(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n}\sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j \tag{2}$$

Where:

- w_0 is the global bias.
- w_i are the weights for each feature x_i.
- $\mathbf{v}_i$ is the latent vector associated with feature i.
- $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$ represents the dot product of the latent vectors $\mathbf{v}_i$ and $\mathbf{v}_j$, capturing the interaction between features i and j.

Remark 2.14. Latent vectors or **factor vectors** are used to model the interactions between pairs of features. The prediction function of an FM incorporates these factor vectors to capture the pairwise interactions without explicitly computing interactions for all possible pairs, which would be computationally prohibitive.

To evaluate the FM, **Normalized Discounted Cumulative Gain (NDCG)** will be used. It's a measure to evaluate the quality of rankings in information retrieval and recommendation systems. It assesses how well the predicted rankings of items match the actual, ideal rankings based on their relevance. It's computed as follows.

Firstly, the Discounted Cumulative Gain (DCG) is computed, which accounts for the position of the items in the ranked list:

$$\mathrm{DCG}_p := \sum_{i=1}^{p} \frac{2^{\mathrm{rel}_i} - 1}{\log_2(i+1)} \tag{3}$$

where rel_i is the relevance score of the item at position i, and p represents the rank position or the cutoff point in the ranked list. It specifies up to which position in the ranked list the evaluation should be considered.

Secondly, Ideal DCG (IDCG) is computed. This is the DCG of the ideal ranking, where the most relevant items are ranked at the top. It serves as a benchmark for normalization.

$$\mathrm{IDCG}_p := \sum_{i=1}^{p} \frac{2^{\mathrm{rel}_i^*} - 1}{\log_2(i+1)} \tag{4}$$

where rel_i^* is the relevance score of the item at position i in the ideal ranking.

Lastly, Normalized DCG (NDCG) is obtained. This is the ratio of DCG to IDCG, ensuring the score is between 0 and 1, where 1 indicates a perfect ranking. The formula is:

$$\mathrm{NDCG}_p := \frac{\mathrm{DCG}_p}{\mathrm{IDCG}_p} \tag{5}$$

3 Conformal prediction

Black-box machine learning models are routinely used in high-risk settings like medical diagnosis, which has potentially fatal consequences. Point predictions are normally generated, returning the most plausible output from the learned training data. Although this method is sufficient in most cases, it can result in a significant loss of information because it ignores the model's potential uncertainty and variability. Thus, uncertainty quantification is needed to distinguish between a certain prediction from an uncertain one.

Conformal prediction[10] is a method capable of transforming any heuristic notion of uncertainty of a model into a rigorous measure of it. This measure is then used to form statistically rigorous sets/intervals for the predictions, being smaller for certain predictions and larger when the model isn't certain. This method is independent of the model chosen and the data distribution, and has very little computational cost.

Figure 5: Prediction set examples on Imagenet. Three progressively more difficult examples of the class fox squirrel and the prediction sets generated by conformal prediction are shown.

Suppose we have a model f and a set of labeled data (X, y), from which a subset $(X_{\text{train}}, y_{\text{train}}) \subset (X, y)$ has been used to train. We take a moderate-sized (e.g., 100 samples) subset of data not seen during training which will be called calibration data:

Definition 3.1. Calibration data is a subset of n samples of the labeled dataset that haven't been used during training and will be used to conformalize the prediction of the model.

$$(X, y)_{\text{calib}} := \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subset (X, y) \setminus (X, y)_{\text{train}}$$

Intuitively, our goal is, for a new feature vector $\mathbf{x}_{n+1}$, to return a set or interval $\tau(\mathbf{x}_{n+1})$ such that we can affirm $y_{n+1} \in \tau(\mathbf{x}_{n+1})$ with high probability:

$$\mathrm{P}(y_{n+1} \in \tau(\mathbf{x}_{n+1})) \geq 1 - \alpha$$

where P is the probability and $\alpha \in [0, 1]$ is called the **error rate**. The probability chosen is called **coverage**. For example, for 90% coverage, we would choose $\alpha = 0.1$.

To achieve this, the following procedure is followed:

1. Identify a heuristic notion of uncertainty for the chosen model.
2. Define a *conformal score* function based on the heuristic.

Definition 3.2. Given $(\mathbf{x}_i, y_i) \in (X, y)$ (a feature vector and its label), the **score** function is a positive function s that measures the uncertainty of the prediction, knowing its actual label. We write it as $s(x, y)$.

3. Compute $\hat{q}$:

   Definition 3.3. $\hat{q}$ is defined as the $(1 - \alpha)$-th quantile of the scores for the n calibration samples (plus a finite sample correction):

   $$\hat{q} := \text{Quantile}\left(s_1, \ldots, s_n; \frac{\lceil (n+1)(1-\alpha) \rceil}{n}\right)$$

   where $s_i = s(\mathbf{x}_i, y_i)$ for $i = 1, \ldots, n$.

4. Form prediction sets/intervals of the form

   $$\tau(\mathbf{x}_{n+1})) := \{y : s(\mathbf{x}_{n+1}, y) \leq \hat{q}\}.$$

This was the general idea. Now, let's see how it unfolds depending on the model we are dealing with.

3.1 Classification

In classification, an alternative to point predictions is to return sets of predictions with a high level of confidence. This can be achieved with conformal prediction, determining the uncertainty of future predictions and adjust the size accordingly. This provides a more transparent view of the model's predictions, improving decision-making processes by offering a range of plausible outcomes instead of a single prediction.

One of the most commonly used scores in classification is the softmax function applied to neural networks, which simulates the probabilities of each class. However, the values it returns are not real probabilities, as they are not calibrated. This means, softmax scores do not accurately reflect the true likelihood of outcomes, and do not necessarily indicate the certainty of the predictions.

To conformalize the softmax (σ) outputs, we must find a heuristic to measure the model's uncertainty using the calibration data. One possible option is taking the values of the true class for every calibration data sample, treat 1 minus its softmax output as the score s and compute $\hat{q}$ accordingly. The problem with this approach is that, as only the true class is considered, the prediction set sizes end up not being very adaptive. This is not something we want as it wouldn't provide much information about the uncertainty of the model, even though the average set size is smaller.

Another option, which we will treat as the preferred one, is defining the score as the sum of the values of the outputs, sorted from higher to lower and up to the true class, included. Hence, higher scores (which will be closer to 1) mean that the model isn't certain and predicted other classes to be the correct ones before we actually reached the true class. This score results in more adaptive sets as the outputs of the other classes are considered as well, not only the true class. The formalization of this technique is as follows: given a calibration dataset $(X, y)_{\text{calib}}$ with n samples,

$$s(x, y) = \sum_{j=1}^{k} f(x)_{\pi_j(x)} \tag{6}$$

where $\pi(x)$ is a function dependent on the feature vector such that π^{-1} is the permutation of $Z = \{1, 2, \ldots, K\}$ and sorts the set Z by their softmax outputs, from higher to lower. k is the index of the true class y in the set of sorted softmax outputs, and f is the model. Note that k is dependent on y.

Example 3.4. Given a sample $(\mathbf{x}_i, y_i)$, if the true class has the second highest softmax output, then $\pi_2(\mathbf{x}_i) = k$ and the summation is of two elements:

$$s((\mathbf{x}_i, y_i)) = f(\mathbf{x}_i)_{\pi_1(\mathbf{x}_i)} + f(\mathbf{x}_i)_k.$$

Given a new feature vector $\mathbf{x}_{n+1}$, its prediction set will be

$$\tau(\mathbf{x}_{n+1}) = \{\pi_1(\mathbf{x}_{n+1}), \ldots, \pi_k(\mathbf{x}_{n+1})\}, \text{ where } k = \sup_{k' \in 1,\ldots,K} \left\{ k' : \sum_{j=1}^{k'} f(\mathbf{x}_{n+1})_{\pi_j(\mathbf{x}_{n+1})} \right\} + 1 \tag{7}$$

1 is added to the supremum so that the sum of the softmax outputs just exceeds $\hat{q}$.

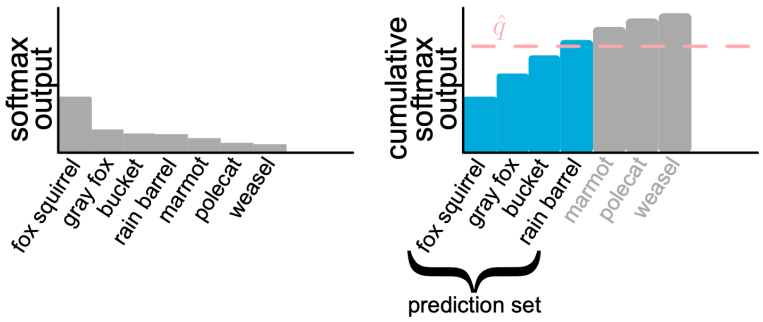

Figure 6: The left image shows the sorted softmax scores of a new input. The right image shows the classes that will be in the prediction set following the rule above.

3.2 Regression

For regression models, a similar thing happens. Instead of getting a single value prediction $f(\mathbf{x}_i) \in \mathbb{R}$, what could be more valuable is to get a prediction interval $f(\mathbf{x}_i) = (a, b) \subset \mathbb{R}$ with high confidence.

To achieve this, we are going to modify the regression model in order not to estimate the value, but the quantiles. More precisely, we will define two models that depend on the chosen value α: one, called $f_l(x)$, will estimate the $\frac{\alpha}{2}$–th quantile, and the other, called $f_u(x)$, the $(1 - \frac{\alpha}{2})$–th quantile. The letters l and u are chosen to represent each function because f_l will estimate the lower quantile and f_u, the upper one (as normally $\alpha < 0.5$). Both functions are trained by optimizing their respective pinball loss, which is:

- $L_{\frac{\alpha}{2}}(y, f_l(x)) = \begin{cases} \frac{\alpha}{2}(y - f_l(x)) & \text{if } y \geq f_l(x) \\ (1 - \frac{\alpha}{2})(f_l(x) - y) & \text{if } y < f_l(x) \end{cases}$, for $f_l(x)$
- $L_{1-\frac{\alpha}{2}}(y, f_u(x)) = \begin{cases} (1 - \frac{\alpha}{2})(y - f_u(x)) & \text{if } y \geq f_u(x) \\ \frac{\alpha}{2}(f_u(x) - y) & \text{if } y < f_u(x) \end{cases}$, for $f_u(x)$

The idea is that, e.g., for $\alpha = 0.1$, we can take intervals from the lower estimated quantile to the upper one. As they exclude the bottom and top $\frac{\alpha}{2} = 0.05$ of the values, we end up with an error rate of 0.1 and 90% coverage. The problem is that, as the fitted quantiles can be inaccurate, this coverage is only an estimation, so we have to conformalize it.

To do it, we take the score defined as the distance between the actual value y and its nearest quantile:

$$s(x, y) = \max(\{f_l(x) - y, y - f_u(x)\}). \tag{8}$$

After computing the scores on the calibration set and setting $\hat{q}$, we can form valid predictions intervals by taking

$$\tau(\mathbf{x}_{n+1}) = [f_l(x) - \hat{q}, f_u(x) + \hat{q}] \tag{9}$$

for a new feature vector $\mathbf{x}_{n+1}$. Intuitively, τ grows (if $\hat{q} > 0$) or shrinks (if $\hat{q} < 0$) the distance between the quantiles to achieve coverage.

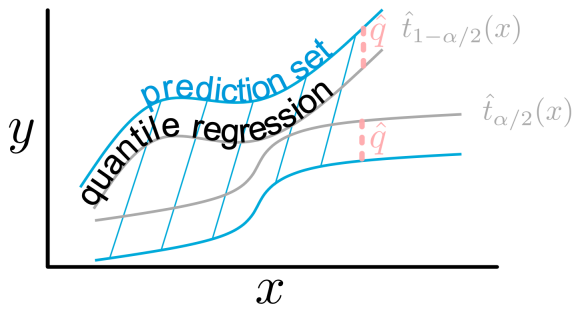

Figure 7: Quantile estimations and its conformalized version, used to get prediction intervals with guaranteed coverage.

3.3 Recommender systems

As we have seen, recommender systems are complex due to data sparsity. Users interact with only a small subset of available items, resulting in vast matrices where most entries are missing. This sparsity makes it difficult to accurately infer preferences since there is limited direct information about user-item interactions.

Measuring the uncertainty of the model's predictions could enhance the system's effectiveness. If a recommender system quantifies its confidence in a given recommendation, it provides a measure of reliability, and the recommendation set adapts to its confidence and gives a notion about how much trust can be placed in a recommendation. Users can understand when a recommendation is robust and when it is based on limited information, which can foster greater trust and satisfaction with the system. We will try to achieve this using conformal prediction. Thus, addressing the inherent complexity of recommender systems through uncertainty measurement not only improves prediction accuracy but also enhances user engagement and trust.

We focus on explicit feedback recommendation systems, which, as the name says, have explicit feedback from the user in user-item interactions. From said data, a Factorization

Machine can learn and predict the ratings of user-item interactions that haven't happened.

When recommending to a user, the recommender takes the top n predicted ratings for interactions that haven't happened and shows them to the user. To find a measure of uncertainty and apply conformal prediction, let's think about a heuristic notion that can somehow represent this.

Facing this problem, it reminds a bit of classification, as the output is a set predictions and the output is sorted from higher to lower confidence score. For instance, let's take the conformal score as the sum of the rating up to the item with the highest true rating. This way, if the first item of the prediction is the best rated one, the score consists only of the addition of one rating. However, if several items are recommended until reaching the optimal one, the ratings add up and the total score is greater.

Then, $\hat{q}$ is computed and adaptive recommendation sets are formed, being larger if more lower-rated predictions are needed to reach $\hat{q}$, and smaller if less higher-rated predictions suffice.

4 Experimentation

Let's see the different models where we tried conformal prediction.

4.1 Conformal prediction in classification: The MNIST dataset

The MNIST (Modified National Institute of Standards and Technology)[2] dataset is a collection of 28×28 pixel images of handwritten digits from 0 to 9. Each pixel value ranges from 0 (black) to 255 (white), i.e., the classical 8-bit representation of grayscale values. It consists of 60,000 images for training and 10,000 for testing, and it's commonly used for testing image classification models. It was introduced in 1998.

Let's explore the data:

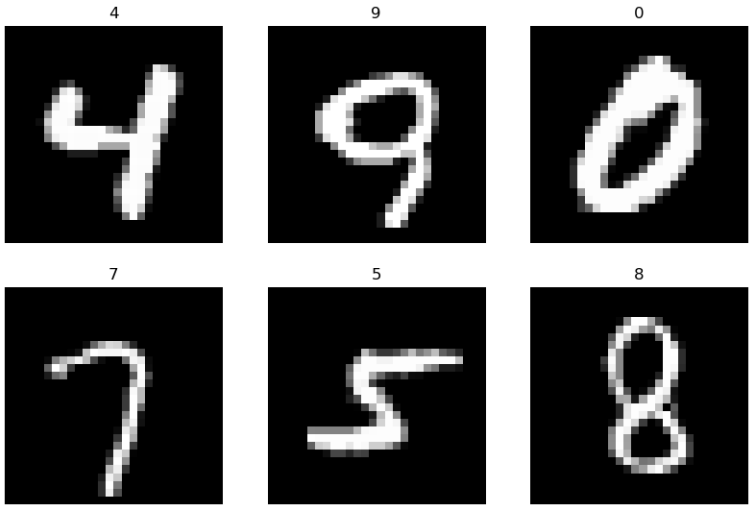

Figure 8: Some MNIST labeled samples

As we can see, the label above corresponds to that same digit. The chosen model to solve the MNIST classification will have a softmax output, so an ideal model would be a vector of 10 positions, with the value 1 in the index of the true digit and 0 for the rest. For instance, the true labels must have the same format to compute the loss function against the predicted softmax outputs. The first step we do is transform the labels of the training data so they have this same format, called **one-hot encoding**:

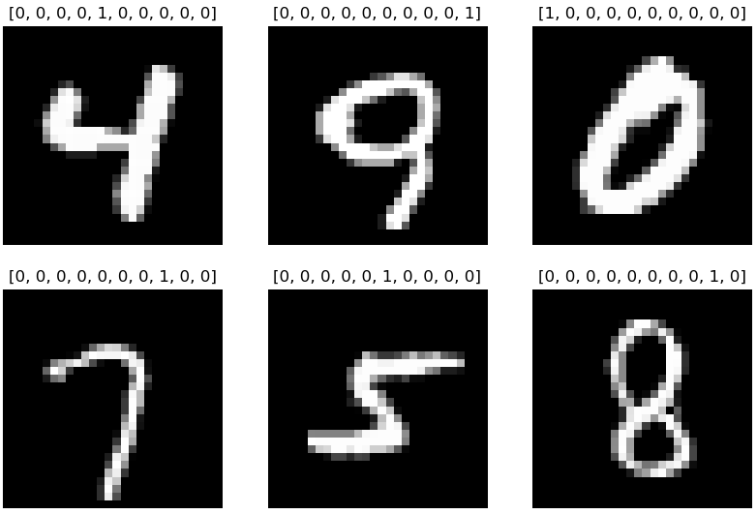

Figure 9: The same MNIST labeled samples as in Figure 8, but with one-hot encoding

This transformation is not needed for the test dataset because the prediction after training will take the index of the highest softmax value, not the raw softmax outputs.

The neural network defined takes all the $28 \times 28 = 784$ pixels as the input, has two hidden layers of 512 nodes each, and a 10-dimensional output of the softmax scores for each class. Each node in the hidden layers has ReLU as the activation function.

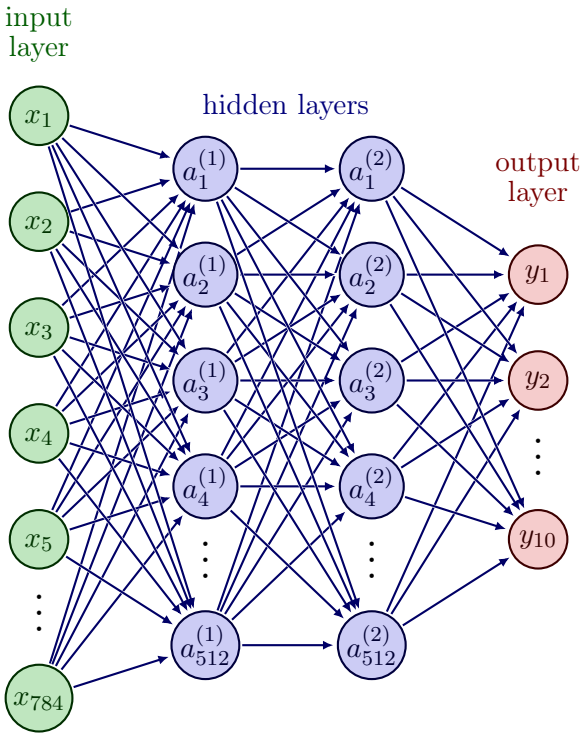

Figure 10: Representation of the neural network

The loss function is the cross-entropy loss, and the optimizer is mini-batch gradient descent with a batch size of 64. The learning rate is 10^{-1} and the optimization will stop

after 5 epochs.

After the training, we end up with an accuracy (percentage of correctly predicted classes) of 86.1% on the test set. Now, let's apply conformal prediction to form prediction sets with high confidence.

We start by choosing 100 samples not seen during training as calibration samples $(X, y)_{\text{calib}}$. We choose an error rate of $\alpha = 0.1$. We compute the confidence scores $s_1, \ldots, s_{100}$ for each sample, where each s_i is the score seen in 6.

Then, we calculate $\hat{q}$:

$$\hat{q} = \text{Quantile}\left(s_1, \ldots, s_{100}; \frac{\lceil (100+1)(1-0.1) \rceil}{100}\right) = \text{Quantile}(s_1, \ldots, s_{100}; 0.91).$$

In this case, we obtain $\hat{q} = 0.9999999$. Now, we take the rest of the samples not seen during training (9900 samples) and form new prediction sets, following the equation 7:

	true_class	prediction_list
0	6	[6, 2, 8, 0, 4, 5]
1	0	[0, 8, 2]
2	5	[8, 3, 9, 7, 0, 5, 4]
3	4	[4, 9]
4	9	[9, 4, 8, 7, 2, 5, 3, 0]

Figure 11: Some prediction sets

As we see, the first column is the true class, while the second one corresponds to the prediction set for that MNIST image. The set is ordered from higher to lower softmax score.

Knowing this, the model without conformal prediction would have chosen the first element of the list as the prediction. Thus, in this example, the model would have correctly predicted all of the images except for the third one, as the true class is 5 while the class with the highest softmax is 8. With conformal, we manage to include 5 inside the prediction set, so all the predictions are correct. Moreover, the sets seem to be adaptive, indicating more uncertainty with larger sets.

To further investigate this, an idea comes to mind. Let's see the average set size for correctly predicted classes versus incorrectly predicted ones, taking the first element of the set as the point prediction. We obtain:

- 4.27 average set size when the prediction was correct.
- 6.91 average set size when the prediction was incorrect.

This reflects the adaptive behaviour of the prediction sets, as when the model is in fact predicting correctly, the set size is smaller. When it's uncertain, it's larger.

Plotting the number of sets of each possible size and doing so for correct versus incorrect point predictions, we see a clear shift of the histogram values to the right, indicating larger set sizes for uncertain predictions:

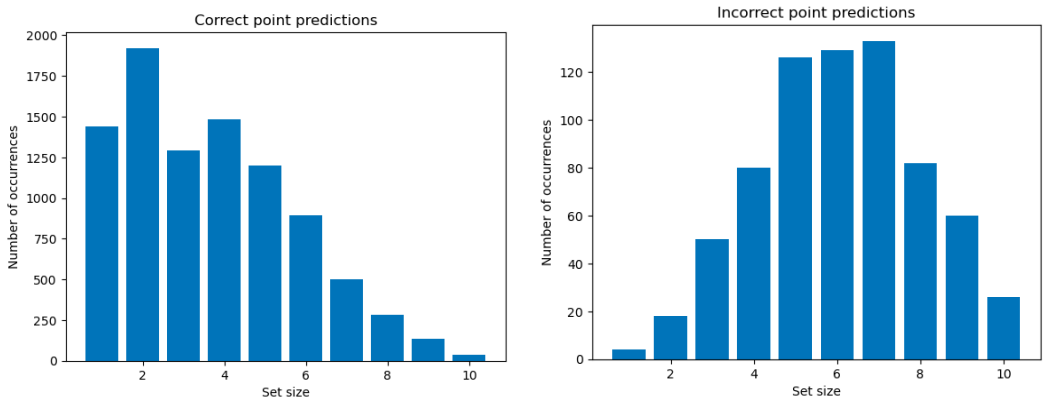

Figure 12: Histograms of set sizes for correct and incorrect point predictions

4.2 Conformal prediction in regression: California Housing dataset

The California Housing dataset studies the median house value for California districts, expressed in hundreds of thousands of dollars ($100,000), which will be our target variable.

This dataset was derived from the 1990 U.S. census, using one row per census block group. A block group is the smallest geographical unit for which the U.S. Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people).

A household is a group of people residing within a home. Since the average number of rooms and bedrooms in this dataset are provided per household, these columns may take surprisingly large values for block groups with few households and many empty houses, such as vacation resorts.

The input features used to estimate the prices are:

- Median income in the block group.
- Median house age in the block group.
- Average number of rooms per household.
- Average number of bedrooms per household.
- The block group population.
- Average number of household members.
- Latitude of the block group.
- Longitude of the block group.

As usual, 20% of the data is kept as test data. The model chosen to estimate the lower and upper quantiles is Gradient Boosting Regressor (GBR)[8], as two separate models. The loss function for both of them is the pinball loss, the optimizer is Gradient Descent and the number of estimators is 200. The learning rate is 10^{-1}.

This time, we have to choose the coverage we want before training, as the models depend on the chosen α. We choose $\alpha = 0.1$, seeking 90% coverage, so we will have

two models $f_l(x)$ and $f_u(x)$ that estimate the 0.05-th quantile and the 0.95-th quantile, respectively.

An additional third model $f_(x)$ is trained, which is a Gradient Boosting Regressor as well and predicts the mean of the price given a feature vector. We will obtain metrics from it to get an idea of how GBR is performing.

After training the three models, we can plot the actual values against the predicted ones for $f(x)$:

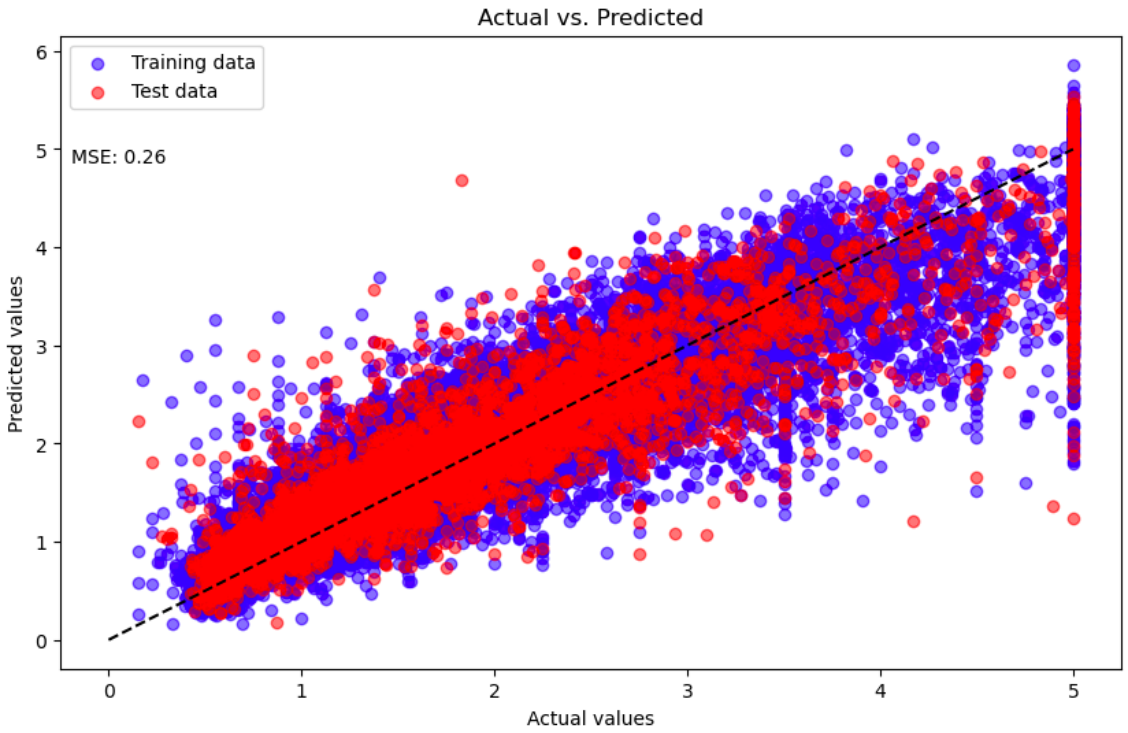

Figure 13: Plot of the actual prices against the predicted ones

It's difficult to represent the results in a clear way as the input data is 8-dimensional, but the previous plot gives us a rough estimate of the performance. As we can see, as the actual price gets bigger, the model tends to underestimate the price. The MSE for the test set is 0.26. Moreover, there is an outlier for houses with a price of \$500,000. It seems that any house of that price or above gets assigned this hard limit, so let's get rid of them by pre-processing these outliers and rerun the model:

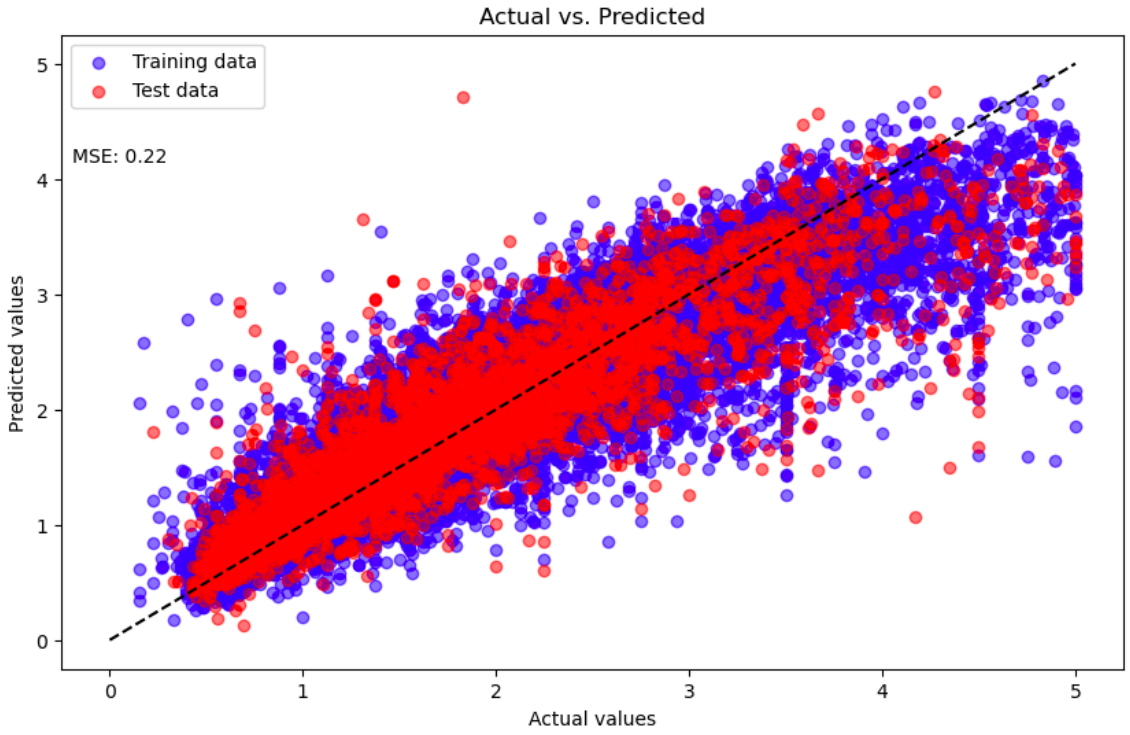

Figure 14: Plot of the actual prices against the predicted ones, getting rid of the outliers

That's better. Also, the MSE decreased to 0.22. Taking the square root of this value gives us an RMSE of 0.47. This means that, on average, the predictions deviate from the actual values by $47,000.

We choose 100 samples not seen during training as calibration samples $(X, y)_{\text{calib}}$. We compute the confidence scores $s_1, \ldots, s_{100}$ defined in equation 8 for each sample and $\hat{q} = \text{Quantile}(s_1, \ldots, s_{100}; 0.91) = -0.008$. That's very close to 0, and it shows that the models have correctly estimated the lower and upper quantiles for a 90% coverage. As $\hat{q}$ is negative, we can actually shrink the distance between the quantiles a little bit to achieve the desired coverage. For instance, applying equation 9, we end up with these prediction intervals:

$$\tau(\mathbf{x}_{n+1}) = [f_l(x) + 0.008, f_u(x) - 0.008].$$

4.3 Conformal Prediction in Recommender Systems: MovieLens

The MovieLens dataset is a widely used collection of movie ratings and metadata, compiled by the GroupLens Research Project at the University of Minnesota[9]. It's primarily used for research and experimentation in the field of recommender systems.

The dataset we use is MovieLens 1M: released in 2003, this version contained 1 million ratings from 1 to 5, from 6,040 users on 3,706 movies. The ratings are made by users of the MovieLens website, which was created to provide personalized movie recommendations to users. By using the service, users rate movies they have watched, which helps the system to recommend new movies they might like based on their preferences.

We use a Factorization Machine that learns from the already rated movies, using the

rating, the movie, the user and its occupation to predict rating for unseen movies following the equation 2.

The model is trained using MSE as the loss function and Adam as the optimizer. The number of epochs is 30 and the batch size is 2048.

With these conditions, the model yields an NDCG (see 5) of 0.896 for the top 5 recommendations. The score defined in 3.3 is calculated for 100 calibration samples, modifying the ratings to increase their distance exponentially. Otherwise, they are too similar to notice significant difference between scores. For this reason, each rating r becomes 3^r.

Then, $\hat{q}$ is computed by choosing $\alpha = 0.2$ (a lower alpha never outputs only 1 movie, so a higher α is used to achieve this behaviour):

$$\hat{q} = \text{Quantile}\left(s_1, \ldots, s_{100}; \frac{\lceil (100+1)(1-0.2) \rceil}{100}\right) = \text{Quantile}(s_1, \ldots, s_{100}; 0.81).$$

Let's see an example of the recommendations this new conformalized output makes:

```
[1401, 3246, 1097, 1958]
    movieID       pred  true
40     1401  78.706886     3
42     3246  71.478683     3
14     1097  62.265854     5
44     1958  53.032593     3
81     3949  52.458145     4
[933, 2289, 3471]
    movieID       pred  true
14      933  81.447838     4
68     2289  81.259209     3
54     3471  63.874931     5
42     1268  33.250431     3
51     2094  26.580118     3
[3435]
   movieID        pred  true
7     3435  346.681000     5
2     1260  320.110046     5
8     3364  182.051071     4
6     3788  118.671135     5
4     3481  118.041138     4
```

Figure 15: We see three recommendation sets. Below each set, the scores of each movie are shown, and its true labels. In the first set, the model is uncertain and outputs 4 movies, and in fact the movie with the highest score only shows up in the 3rd position. In the next one, the model is more certain and outputs 3 movies, while the true higher-rated movie is 3rd. Finally, in the last set, the model is very certain and only outputs 1 movie, and the true higher-rated movie is first.

5 Discussion

5.1 Future work

From now on, it would be beneficial to apply Convolutional Neural Networks (CNNs) to the MNIST dataset. CNNs are more suitable for image classification tasks, and their use should lead to higher accuracy. This approach could also provide additional insights from the data, combined with a more detailed analysis of each class independently.

For the California housing dataset, additional work can be done to understand why the model underestimates the prices of more expensive houses, and see if it gets fixed with another model or more pre-processing has to be done.

Regarding the recommender system, the new techniques that have been considered should be further formalized with precise mathematical definitions. Their effectiveness can then be studied in greater depth to evaluate the proposed scoring and the quality of the adaptive sets. Additionally, other loss functions should be tried as the ratings are not calibrated, so the MSE isn't the best option. Other options such as the Pearson correlation coefficient may be more suitable.

Overall, a more in-depth analysis of the results is possible, ensuring that the coverage really holds by generating many prediction sets/intervals and verifying its statistical properties.

6 Conclusions

In this thesis, we have successfully addressed the challenge of explainability in machine learning models by focusing on the measurement and adjustment of uncertainty in their outputs. Our primary objective was to mitigate the risks associated with the opaque nature of machine learning predictions, particularly in sensitive applications such as medical diagnosis and mortgage eligibility.

We began by exploring conformal prediction as a method to provide a statistically backed set of possible outcomes instead of single-value predictions. This approach ensures that the true prediction lies within the set with a high degree of confidence, effectively communicating the uncertainty inherent in the model's predictions.

Firstly, we applied conformal prediction to a classification problem using the MNIST dataset. Through this application, we demonstrated the method's capability to quantify uncertainty and adjust the prediction sets accordingly. This step provided a solid foundation for further exploration into regression problems.

We then extended our work to a regression problem, specifically estimating real estate prices in California. By leveraging conformal prediction, we managed to produce prediction intervals that dynamically adjusted in size based on the model's uncertainty. This not only enhanced the model's transparency but also provided more reliable and interpretable predictions for end-users.

Finally, we explored the application of conformal prediction in recommender systems, an area that inherently involves significant uncertainty, especially with new users. We successfully quantified the uncertainty of a recommender system and applied conformal prediction to ensure that recommendation sets were appropriately sized according to the user's data availability. Larger sets were provided for users with high uncertainty, and smaller, more precise sets were given to users with well-known preferences.

In conclusion, we have demonstrated that conformal prediction is a robust and versatile tool for enhancing the explainability and reliability of machine learning models. By providing a clear measure of uncertainty and conformalizing the outputs, we have taken steps toward more transparent and trustworthy ML models.

References

[1] Scantamburlo, T.; Baumann, J.; Heitz, C.: On prediction-modelers and decision-makers: why fairness requires more than a fair prediction model, http://dx.doi.org/10.1007/s00146-024-01886-3, 2024.

[2] LeCun, Y.; Cortes, C.; Burges, C. J. C.: The MNIST database of handwritten digits, http://yann.lecun.com/exdb/mnist, 1998.

[3] Nair, V.; Hinton, G. E.: Rectified Linear Units Improve Restricted Boltzmann Machines, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 807-814, 2010.

[4] Huang, C.; Yu, T.; Xie, K.; Zhang, S.; Yao, L.; McAuley, J.: Foundation Models for Recommender Systems: A Survey and New Perspectives, *arXiv*, 2024, arXiv:2402.11143.

[5] Loni, B.; Larson, M.; Hanjalic, A.: Factorization Machines for Data with Implicit Feedback, *arXiv*, 2018, arXiv:1812.08254.

[6] Rendle, S.: Factorization Machines, *Proceedings of the 2010 IEEE International Conference on Data Mining*, 995-1000, 2010.

[7] Visnovsky, J.; Kassak, O.; Kompan, M.; Bielikova, M.: The Cold-start Problem: Minimal Users' Activity Estimation, *arXiv*, 2021, arXiv:2106.00102.

[8] Friedman, J. H.: Greedy Function Approximation: A Gradient Boosting Machine, *Annals of Statistics*, 29(5), 1189-1232, 2001.

[9] Harper, F. M.; Konstan, J. A.: The MovieLens Datasets: History and Context, *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4), Article 19, 19 pages, December 2015. DOI: http://dx.doi.org/10.1145/2827872

[10] Angelopoulos, A. N.; Bates, S.: A Gentle Introduction to Conformal Prediction and Distribution-Free Uncertainty Quantification, 2022.