Inferring Interaction Network from Sensor Data

BY

ETTORE RANDAZZO
B.S, Politecnico di Milano, Milan, Italy, 2016

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2016

Chicago, Illinois

Defense Committee:

Tanya Berger-Wolf, Chair and Advisor
Bhaskar DasGupta
Pier Luca Lanzi, Politecnico di Milano

To my uncle Gianni, my memory of you will always inspire me to achieve all my goals in life, just as you did.

ACKNOWLEDGMENTS

I would like to thank my parents, Adriana and Marcello, who made this dream come true, and my brother Nestore, who has always been there for me when I needed.

I believe both Politecnico di Milano and University of Illinois at Chicago formed me the way I am now, and I am really grateful for that. In particular I would like to thank Tanya Berger-Wolf, Pier Luca Lanzi and Robert Kenyon who also helped me with my mental growth, doing and caring about me much more than what is usually expected from a professor.

ER

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

CHAPTER **PAGE**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (continued)

LIST OF ABBREVIATIONS

GPS	Global Positioning System
DNA	Deoxyribonucleic Acid
WAAS	Wide Area Augmentation System
BLE	Bluetooth Low Energy
RSSI	Received Signal Strength Indication
SVM	Support Vector Machine
NN	Neural Network
MPU	Magnetic Pick-Up
FSM	Finite-State Machine
ReLU	Rectified Linear Unit
OFAT	One-Factor-At-a-Time
DT	Decision Tree

SUMMARY

Being able to observe how animals interact among themselves has always been a crucial requirement for behavioral scientists who study social species. Physically watching them to see when interactions occur is extremely time-consuming, results in many missed observations and it becomes extremely difficult to do for a very extended period of time. This is especially true for animals who live in large territories and for animals who behave differently when nearby human beings. To overcome these problems, scientists are accustomed to use several different kinds of sensors which are usually attached to the target animals to record some kind of raw data which, once collected, is used to analyze the behavior of their hosts.

The sensors have to be as least invasive as possible for the animal to behave as if it wasn't wearing them. This implies that sensors have to be light with respect to the animal, they must not emit a considerable amount of heat and the wavelength they use must not be perceptible by the animals near them.

The most popular type of data extracted from sensors attached to animals is Global Positioning System (GPS) data. GPS data is very easy to extract and it can be used to efficiently track the positions of entire groups of animals. However, when we are interested in pairwise interactions between animals and not in their positions, GPS data is not very reliable either because of its low accuracy and because it might not be in line of sight with the satellites it relies on.

SUMMARY (continued)

In this study, we introduce synthesized sensor data based on a type of non-invasive short range proximity sensor in order to understand whether some animals interact among themselves at a given time. It is essential to label the data in order to be aware of the interactions as they occur during the traning phase.

The sensors whose data we are interested in synthesizing can be used on animals that are capable of wearing them due to weight or size constraints and as long as the sensor range contains the interaction range of the animals we are interested in. The models we analyze in this work don't assume either that all of the animals behave uniformly amongst each other and also the independence amongst interactions.

We present a framework to synthesize proximity, location and speed data extracted from sensors with several different configurations and we present models to infer animal interactions.

Finally, we evaluate the methodology we use by proposing a case study where we perform different analyses to understand when our models are fit for the inference task, what are the most critical parameters impacting their performances and when we should start assuming independence conditions to simplify the task.

CHAPTER 1

INTRODUCTION

Humans have been observing animals since ancient times. The reasons have been very different from being as effective as possible for hunting and to avoid dangerous predators to pure interest in the everyday life of animals.

After the work of Charles Darwin and the birth of the theory of Evolution [1], observing animals has become extremely interesting because of the connection of animals with humans. Moreover, with the discovery of the Deoxyribonucleic Acid (DNA) in 1869 and its molecular structure's identification in 1953 [2], the study of animal behavior has become important to also understand human behavior.

However, observing animals is not always an easy task, either because of the time it takes to gather significant information from animals and because sometimes it is very hard to follow some types of animals, such as predators whose territory is usually very vast. The advent of the technology revolution of the most recent years gave biologists a new way to observe animals.

By exploiting sensors attached to several individuals of a given species, we can observe their movements for a very long period of time. This gives us insight of what certain animals do, where they go, what their usual route is, for example for hunting, and ultimately how they interact with their environment and companions even when it would not be possible to be observed by a scientist.

The most used sensor data for these tasks nowadays is GPS data. It is a space-based navigation system which gives information about time and position, given in longitude and latitude, and can be used anywhere in the world as long as there are four or more GPS satellites that have an unobstructed line of sight to the sensor used. This makes GPS data available almost anywhere on Earth and extremely efficient in tracking the movements of animals in their habitat, especially when they live in a vast area free of obstructions, basically anywhere but thick forests.

However, GPS data has its shortcomings; the standard accuracy of an average GPS sensor is about 15 meters (49 feet), which can be improved to 3 meters (9.8 feet) by using Wide Area Augmentation System (WAAS). The accuracy of a sensor is how confident we are about the position we are recording. This means that having an accuracy of 3 meters implies that the real position of the sensor can be anywhere in a circle with a radius of 3 meters. Despite being acceptable for tracking animal movements in large areas, this is not the case when we want to spot interactions between two individuals.

In fact, animals tend to have topological interactions instead of metric interactions [3]; they usually interact with a small, hardly varying number of the closest animals instead of every animal closer than a threshold. Thus, being uncertain of a distance by four times the accuracy of a single sensor between two animals (twice the accuracy per animal), makes GPS data by itself, without great redundancy, unfit for the task. Moreover, for smaller animals, the redundancy needed to have usable data might make the task impossible due to weight as well as cost constraints.

An alternative to GPS data is to use Bluetooth Low Energy (BLE) transmitters and receivers so that the receiver is able to compute an Received Signal Strength Indication (RSSI) value, which stands for the strength of the signal between the two sensors. This is far more accurate than GPS data, however it does not give either absolute positions and accurate absolute distances between sensors, because of the many reasons a signal could be weaker than expected. Still, it is sufficient when we want to understand interactions between animals, which is very dependent on their relative distances

1.1 Thesis Objective

This work will exploit RSSI data, as well as accelerometer data to extract interactions among groups of animals. We will work with different configurations of sensors, creating a framework able to synthesize such networks to extract the best model for a given instance and test its performances with different parameters to understand when it does and does not perform well.

When scientists are interested in interactions between animals, what they *mean* by interaction may well vary from one to another. Some might consider an interaction when two animals are sufficiently near one another, others when one approaches the other and the interaction might be only an instant despite them being close for a longer period. Others might think an interaction to be when two animals do something together, others when one follows another one. Given that the word "interaction" is so arbitrary, in order to train any model to fit the specific definition of any given instance, it implies the need to have *labeled* data.

Labeled data means that along with the raw input data which comes from the sensors, we are given the interactions that occur among the animals we are considering at any given time

period we are interested in sampling. This is necessary for most models that want to learn how to predict an output (in our case the interactions between our animals) given an input (in our case our raw data collected from sensors).

Unfortunately, this means that scientists still have to manually annotate interactions of animals for a fairly large amount of time; the longer they annotate interactions, the more robust the models will be. This might seem counterintuitive; we want to build a model to *not* manually annotate interactions, and we require them for our model to work. However, the *amount* of data we have to label is significantly different; ideally, once we labeled a sufficient amount of data, we would not need to label it anymore in the future, regardless of the number of instances we would be able to sample with our sensors. Suppose we had a month of recordings of interactions of animals, we would probably need to label only about a few days or even a few hours of interactions to be able to build a reasonably good model, which would then label all the remaining time.

The models we analyze in this work take into account uniquely identifiable animals, that is every animal is considered differently from every other one, and if an interaction occurs between two animals, it may impact the likelihood of other interactions. These are probably some of the most nonconstraining assumptions we can make, however such models have their shortcomings; they require *a lot* of data, w.r.t. other possibilities.

We estimate that the ideal model to spot interactions depends on the rarity of the interactions themselves (Figure 1); when we have enough data to spot every single possible pairwise interaction a fair amount of times, we expect our model to perform extremely well. However,

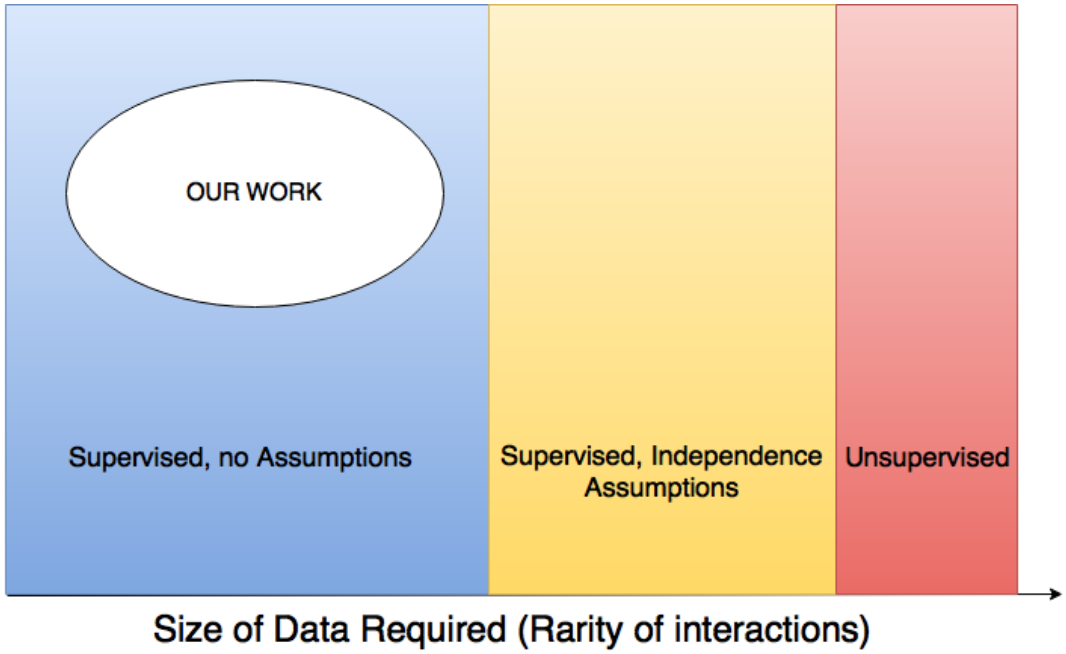

Figure 1: Partitioning of the best model to infer interactions with respect to the rarity of them.

when interactions become extremely rare and we only see a bunch of them, and not for every possible pair (which of course scales also with respect to the number of animals), we would need to drop some assumptions so that every label can be used to train *any* pairwise prediction.

Finally, when spotting interactions becomes extremely time consuming, we can decide to automatically detect them without training any model. This is called unsupervised learning, and it should be used only when no other options are possible, because it is most likely going to give either an extremely large number of false positives or an extremely large number of false negatives, and the worst part of it is that, even if we can reason about the quantity of false

positives, we *cannot* do so for false negatives: we wouldn't know how many interactions we are missing unless we started manually labeling all our data.

1.2 Structure of the Thesis

Chapter 2 will briefly give an insight of the related work. Interestingly enough, supervised learning has hardly ever been considered for animal interactions, most likely because of the time it would need to be done well. Chapter 3 will give necessary background of most of the approaches we will use to build our framework. This chapter is intended to aid a lack of knowledge in a specific area and it is not necessary whenever the reader already has a sufficiently large background on a specific subsection. We still recommend to quickly look at it even in the latter case, to better understand how we are going to use such concepts later on. Chapter 4 will explain the proposed approach. First, it will formally define our tasks, specifying our sensors and our configurations, second, it will explain our models and framework. Chapter 5 will show a case study where we will perform some sensitivity analysis and discuss the obtained results. Chapter 6 will give conclusive observations and talk about possible future work.

CHAPTER 2

RELATED WORK

Whenever we are interested in animals, we are interested in their behavior. Behavior is what makes an animal different from another, behavior is what leads an animal to do what they do. Whenever biologists are interested in studying animal behavior, they need to observe the behavior of animals not only considering single individuals, but how they interact among themselves as well.

Although observing animals manually can be done [4; 5], it can only be done for a small amount of time and usually within small areas.

When interested in small animals constrained to live in a controlled environment such as fish or insects, video tracking technology can seriously help biologists [6; 7; 8].

When facing larger animals, we can sense various information such as their location, movement and orientation thanks to different wireless sensors [9; 10; 11; 12; 13; 14].

GPS data is customary to be used to track animal positions, a well known example is where researchers at Princeton built ZebraNet [15], a tracking system which involves the usage of tracking collars worn by some zebras in Kenya which collect GPS data. Tracking collars were also used to track deer movements over a seven-year time period [16].

However, GPS data is not accurate enough to be successfully used to spot animal interactions [17].

Video or RFID [18] can be used to better detect proximity among animals when it is possible to cover vast areas with location systems. However, it is hardly the case. Ultrasonic tracking could still be used for some animals [19], but it may disturb several other species, making it impossible to be deployed in most cases.

Regarding inferring animal interactions, researchers recently exploited GPS data to track baboons movements [20] in order to study shared decision making movements, dominance interactions by first automatically extracting and then manually validating them as well as extracting dominance hierarchies.

Scientists used k-means algorithm to cluster cows GPS data to spot different behaviors [21]. This is again an unsupervised learning classification. It is indeed usual to automatically extract interactions among animals in many different ways: unsupervised learning can be used to find interactions among cattle using a distance threshold and discern stronger ones by timing their lengths [22], scientists can exploit maximum entropy models to infer interactions of mice [23]; others [24] use various formulas as well, but tested its results with some synthesized labeled data, and with a little real labeled data, also showing the usefulness of synthesizing this type of data.

However, unsupervised learning methods have various limitations despite being easy to deploy: inferring a specific kind of interaction is extremely difficult, in fact the interaction is defined by our model, there is no way to estimate its recall and a recent work [25] discusses how generally wrong is the concept of interaction extracted from thresholds w.r.t. what biologists really look for. An alternative to that is supervised learning, as shown in another recent

work [26], where mouse social behaviors were extracted through a supervised learning approach involving data acquisition and adjustment from different sources and a labeling of around 150,000 video frames fed into a Random Decision Forest model.

The gain of supervised learning approaches is undeniable, thus in this study we are especially interested in knowing when we can exploit such a powerful method, instead of using the well known unsupervised approaches.

CHAPTER 3

BACKGROUND

This chapter is going to focus on giving a background knowledge of some key concepts which are going to be used in our proposed approach. Section 3.1 presents the concept of graphs. Section 3.2 explains the basic data subdivision used in building a model. Section 3.3 describes the Markov Assumption. Section 3.5 explains the concept of Support Vector Machine (SVM). Section 3.6 presents the model of Neural Network (NN) and the instances used in this work. Section 3.7 talks about Decision Tree (DT)s while focusing on its ensemble methods. Section 3.8 explains what is a test for statistical significance and why it is useful. Finally, Section 3.9 talks about Sensitivity Analysis, which is going to be the focus of our work.

3.1 Graph Representation

A graph is a mathematical representation of a set of objects usually referred to as vertices, nodes or points, where some pairs of them are connected by links, usually referred to as edges, arcs or lines. The links can either be directed, being referred as *arcs*, or undirected, usually being referred as *edges*. In our work we mostly name the objects *nodes* and the links *edges* when referring to undirected links, and arcs when referring to directed links.

To understand what is the difference between directed and undirected links, we first need to understand what means to have a link between two nodes. Being a mathematical representation of elements, a link represents the occurrence of a *relation* between two of them. The meaning

of such relations is ours to give, so for example suppose that we have two nodes called "Dante" and "Beatrice" and the relation we are interested in is "love". Dante loves Beatrice, so we would say with a mathematical relation Love(Dante,Beatrice). Here, the position of the nodes is relevant: Love(x,y) means: "x loves y", while Love(y,x) means: "y loves x". In this example, Beatrice doesn't love Dante, so, if we wanted to represent our relationships with a graph, we would have to use a *directed* graph, that is a graph which uses directed links and have an arc from Dante to Beatrice.

Suppose instead that we are interested in a relation such as friendship. We generally think that if an individual Foo is friend with Bar, then Bar is also friend with Foo, that is Friend(Foo,Bar) $\iff$ Friend(Bar,Foo). This is called a *symmetric relation*. If we wanted to represent such relations with a graph, we would use an *undirected* graph, that is a graph which uses undirected links.

Edges (as well as arcs) can have weights attached to them, meaning that two nodes have a relation which can be quantified with a number. For example, suppose we are interested in representing the distance between nodes. Every node would have an edge with a number attached to it, which would represent their distance. In a mathematical representation it might be something like Distance(x,y) = Distance(y,x) = k, or in a logical representation Distance(x,y,k) = Distance(y,x,k). A graph with weights attached to its edges is called *weighted* graph, if there are no weights it is called *unweighted* graph.

In this work we will use both weighted and unweighted graphs to represent our instances. To represent the interactions occurring between animals, we are going to use unweighted undirected

graphs, where an edge between two nodes would represent an interaction between them. By the definition of interaction we will use in this work, we know that it is a symmetrical relation, and we are not interested in the strength of such act, so it will be unweighted.

Sometimes, we will represent our input data with weighted graphs to show distances between nodes. Given the peculiar type of data we handle, it might not be the case that the graph is undirected. In that case, we would use directed weighted graphs to represent perceived distances between nodes.

3.2 Data Subdivision

To build a model able to accomplish the task we need it to do, it is usually necessary to have some data that it has to fit. This is referred as *training* data (set).

To fit some data for a model means to extract some latent information from it in order to understand some patterns which lead to a specific prediction [27].

Once a model is learned, it is customary to test it to see whether it is effective at predicting some new data. We do not want to test our model on the same data we trained it with, because it might be the case that the model *overfit* the data, and would perform very badly with new, never seen instances of the input data. Overfitting occurs when the model is excessively complex with respect to the real function that describes our output [28]. Being so complex, what it really does is fitting a random noise in the training data, and how it behaves for unseen instances between two seen ones is most likely going to be random.

To overcome this problem, when we want to evaluate our model properly, we use a different partition of our data which has been untouched by our training process, that is called *test* data (set).

This partitioning of our available data is usually enough to train and evaluate simple models. However, sometimes this is not enough to give unbiased evaluations of our model.

Suppose we have an extra parameter that we want to fine-tune. We can imagine it being the order of our function, or the type of input given to the model, or completely different models. If we trained it with our training set and tested with our test set and then selected the parameter that maximizes our evaluation function, we would be overfitting with respect to the test set. Basically, we would choose the model that best fits our test data, but it would have unknown results for new, never seen data.

When we face this problem, we split our data into three partitions: the training, the *cross validation* (or simply validation) and the test sets.

Following the exact same reasoning done with just train and test sets, we will not touch our test set for all the tuning of our model, and we will instead train with our training set and evaluate with our validation set. In the end, after we choose the best model, we will see its performance against the test set.

We are going to use this partitioning policy to develop our model. Further details will be given once the needed background will be presented.

3.3 Markov Assumption

Suppose we are interested in knowing what the weather will be like today. We might suspect that the weather of today depends on the weather we observed the previous days. For example we might find out that if it has been sunny for several days in a row, it is most likely going to be sunny today as well, with a small probability of being cloudy, and it never happens to rain. If we wanted to model this behavior we would talk about conditional probabilities, for example, suppose that w_t is a variable representing the weather at day t, we would be interested in knowing $P(w_t|w_{t-1}, w_{t-2}, w_{t-3}, , w_0)$ which is generally different than $P(w_t)$. Although this would be a very precise estimation of the probability of a given weather condition for today, it is most likely impossible to have enough data to make it usable in practice, as well as being a very burdensome task. Suppose we find out that in reality, the probability of a given weather today is dependent on only a fixed amount of previous days, for example 2. Then, we would be able to compute its probability in a much easier way: $P(w_t|w_{t-1}, w_{t-2}, w_{t-3}, ..., w_0) \approx P(w_t|w_{t-1}, w_{t-2})$. We call this a *Markov assumption* (in particular, a second-order Markov assumption).

More in general, we have a n^{th}-order Markov assumption if we believe that the probability of a given variable at time t depends only on the n previous times. Usually, this is not true, but it is a necessary assumption to make the problem tractable.

Now, suppose we know the relative positions of every animal we are observing with respect to any other animal at any time t. The best thing to do to find out if two animals are interacting would be to see all the previous history of the network. However, to do so, we would need an

enormous amount of data: suppose that we would need X instances to have a good estimation of a specific input configuration, then, every time we increase our Markov assumption by one, we would need to double the overall required input data for each input variable we consider at any give time. Also, this would be the lucky case where our variables had only two possible values, and we would have to triple, quadruple, etc. for a bigger range.

Thus, we need to make a reasonable Markov assumption. Suppose that the interaction between two animals would only be dependent on the actual time, then a zero-order Markov assumption would be fine. In general, we are interested in finding the best Markov assumption, that gives us good results w.r.t. the size of the data we have, without being too computationally expensive.

3.4 Cost Sensitive Learning

Generally, every classifier is built to maximize the accuracy of its predictions. Although it is usually what the user wants, sometimes this is not the case.

Suppose that we are predicting whether or not a patient has pancreas cancer. Surely we would like to accurately have only true positives and true negatives; however, when we are wrong, the two effects are dramatically different; if we have a false positive, we can just double check with maybe another method to be actually sure that he is ill. More in general, if he is in a hospital, he stays in the hospital and no harm is done. However, if we have a false negative, we would set the patient free and the next time he comes, his situation will be much worse than before.

In all the cases where a false negative is different than a false positive, cost sensitive learning should be done. Cost sensitive learning is also often used to diminish the harm of having unbalanced data, when sampling is not very practical or would worsen the results (such as when we don't have enough data).

3.4.1 Cost Matrix

If we want to represent the different costs of our classifications, it is customary to use a cost matrix, where the rows are the actual values and the columns are the inferred values. Sometimes in other papers the convention is the opposite, so we ask special care from the reader when looking at them.

TABLE I: OUR COST MATRIX: ROWS ARE ACTUAL LABELS, COLUMNS ARE INFERRED LABELS

	T	F
T	0	c_{fn}
F	c_{fp}	0

Usually both the true positive and true negative costs are 0, while very rarely true positives might have negative values to indicate profit. Given that the higher the cost the worse our model is at predicting, to evaluate a model while using cost sensitive learning, we will use a cost function which will simply multiply the number of predictions p_{ij} with the cost weight c_{ij} of that specific cost matrix and sum them all. Clearly, the lower the score, the better the

performance. To our purposes, we will have 0s on both true positives and true negatives, and have a higher weight for false negatives w.r.t. false positives. Table I shows our target cost matrix.

3.4.2 Classifier Adjustments

To make a classifier work with a cost matrix, adjustments must be made. We hereby discuss the most common methods used:

- Make the classifier provide probability estimates, compute the expected cost of each candidate and select the one which minimizes the cost.
- Sample the data during training phase in such a way to have uneven distributions of the classes, which will bias most classifiers to specific outcomes.
- Directly modify the inner workings of the classifier to make it use the cost matrix, which should maximize the fit of that classifier to that specific cost matrix.

The limitations of these approaches come from how specific classifiers work and the type of data we have available while training: most algorithms can't be modified to fit a cost matrix or cannot provide probability estimates. NNs can both provide probability estimates and modify their inner workings (the latter is preferred), but have a hard time sampling the input data when it returns multiple outputs. Given that what we usually want is to have a balanced distribution while training, a stronger weight in the cost matrix might be needed to impartially act as an equalizer.

3.5 Support Vector Machines

A Support Vector Machine (SVM) [29] is a model based on a supervised approach which can perform both classification and regression analysis. It needs as input a set of training data, each line labeled as one of the only two possible categories, to output a deterministic binary linear classifier. It can also perform a non-linear classification by performing the so called 'kernel trick'.

An SVM wants to find a hyperplane which can be used as a classifier between the two selected classes. Ideally, it tries to find a hyperplane that perfectly subdivides the two categories and selects among the many possible, the one furthest from all points. This is usually impossible to do, thus it uses a *soft-margin*, meaning that it tries to split its points at best, trying to minimize the cost of the misclassifications which are proportional to how far is the misclassified point from the right side of the hyperplane.

When a linear classifier is not enough, a transformation is used to represent our input data into a feature space that makes the dimension easier to be separated by a hyperplane. This operation however would be extremely time-consuming, thus a kernel trick is used: we don't need to transform our dataset, we just need to perform a dot product of an N dimensional space with a $M > N$ dimensional space. This can be efficiently done with a kernel function, thus the so called kernel trick.

There exist several extensions for SVMs, which are not needed for our purposes, such as multiclass SVMs and support vector clustering.

3.6 Neural Networks

Artificial Neural Networks, or NNs, are a family of many models inspired by the real biological neural networks.

While they may vary a lot one another, they all have in common their basic structure, which consists of interconnected artificial neurons that send messages each other. An artificial neuron is usually modeled taking example from the Hebbian Rule in biology: basically we receive a number of inputs with some weights attached, we have a threshold called bias and we fire an output. From that, we can have many various implementations.

We are interested in supervised learning with NNs. While the details of these models are many, the general approach is, during training, to try to compute an output while giving a sample input, compute its error w.r.t. the expected output for every output neuron, and then use backpropagation, which is a method to adjust more the weights of the network which are most responsible for the error by using the gradient of its error.

In our work, we will use a fully connected feedforward neural network, which is the oldest and most commonly used, built however using mostly state of the art methods: Rectified Linear Unit (ReLU) activation function with dropout neurons and a Bernoulli negative log-likelihood error function.

A ReLU is a very trivial activation function: the idea is simply to have $f(x) = \max(0, x)$. This has been shown [30] to be more biologically plausible than the more common functions such as logistic sigmoid and hyperbolic tangent. Usually, we prefer not to have 0 values, thus

the rectifier function is smoothed: $f(x) = \ln(1 + e^x)$, which only gives positive numbers, still resembling the rectifier function.

Dropout is considered one of the most successful recent discoveries of the decade. During training phase, we randomly 'drop out' some neurons, setting their output to 0, thus basically not making them fire. During prediction phase, every neuron is allowed to fire. This has again a biological inspiration, from genes: when a child is born, it receives half its genes from both parents, thus every gene has to be able to adapt with unknown other genes, and be useful by itself.

This method has several advantages: it prevents co-adaptations thus avoiding overfitting and it results as a very powerful ensemble method, as it can be seen, while training, to train up to 2^n different networks, where n is the number of neurons, and while testing to merge them. This sometimes implies the need of a higher amount of data required or simply a longer training phase, but it has been shown to own the state of the art performance for several deep learning classification tasks.

3.7 Decision Trees

DTs are a very common type of classifiers. They use a tree to decide how to label an input instance, where the leaves are the class labels and the nodes are some conditions of the input that branches the decision to be taken.

DTs can either predict multiple classes or a continuous value, in the latter case they are called Regression Trees.

This classifier is very popular, mostly because of its simplicity and its reasoning is understandable by humans and can be easily plotted.

A tree is built by greedily deciding what attribute to split at every step, selecting it by choosing a scoring measure. A very well known measure is the Information Gain, which tries to minimize the overall entropy of the system. Informally, we can say that Information Gain = Entropy(parent) - Weighted Sum of Entropy(Children). We do that for every possible child split and select the one with the maximum score. If a node already perfectly explains the data (it classifies all its data with just one label), it becomes a leaf. This process usually continues until or all nodes become leaves or there are no more features to split or the gain from splitting is less than a given threshold.
Usually, branches get pruned also after this process, to reduce overfitting.

3.7.1 Ensemble Methods

DTs by themselves perform well, but they don't own the state of the art performances. Whenever we are not interested in having a decision tree representation and we only want the best classifier, ensemble methods come into play.

Ensemble methods consist of using more than one classifier to infer a label. How the result is chosen among the individual classifications may vary one method another. Ensemble methods work well when the basic classifier is simple and very susceptible to noise. Decision trees are the most common models used because of these needs.

The most common ensemble methods are:

- Bagging [31]

- Boosting [32]
- Random Forest [33]

The main idea is more or less the same for the three of them: we want to have a big number of independent classifiers, that means that ideally if classifier A says 'Yes', we don't know anything about what classifier B is about to say. If every classifier has an accuracy greater than 50% and we take the majority vote, we end up increasing the accuracy ideally by as much as we want, depending on how many classifiers we use, because the more classifiers we use, the more likely it is that the majority of them classified correctly.

This directly applies to bagging and random forests, the difference is how they achieve the condition of independence of their classifiers (also, it is anyway impossible to have a perfect independence among them): bagging, or *bootstrap aggregating*, trains with up to the same number of data of the input data, however it samples them with replacement from it. This is shown to give $(1 - 1/e) \sim 63.2\%$ unique elements, while the rest are duplicates. Being decision trees very sensible to noise in the data, the resulting classifiers behave very differently. Of course, to have even less resemblance, a smaller number of samples can be taken.

Random forests do exactly as bagging, but they increase the classifiers independence by also ignoring a certain number of input features (usually they use the square root or half of such number), making them more prone to error, but still usually over 50%.

Usually random forests perform better than bagging, but, as we will see, this is *not* the case for our instances.

Boosting uses a slightly different approach: it keeps adding classifiers which specialize where the previous classifiers mispredicted the most. In the end, every classifier is weighted and besides predicting the label, it also gives a confidence on how certain it is. The final results are computed by weighting the classifiers by also considering this confidence and the most likely overall result is selected.

3.8 Statistical Significance

Sometimes, we try different models to see which one performs best w.r.t. our data. Now, suppose we have two models which performed similarly, i.e. they had almost the same accuracy, precision and recall. Is really one better than the other, or are they coming from the same distribution, thus either one of the two is the same for us?

To answer these questions, we do a test of statistical significance: how likely is the null hypothesis to be true for the two output distributions of our models? The null hypothesis is supposing that the two populations come from two different distributions (there is no correlation between them).

A test of statistical significance computes a p-value, which is the probability of obtaining results at least as extreme given that the null hypothesis is true, thus they are from two different distributions. A significance level α is introduced to understand when this p-value is excessively small: α is the probability to reject the null hypothesis in the case it is actually true.

Thus, suppose that our p-value is greater than our selected α, then we would be confident with probability $1 - \alpha$ that the null hypothesis is true. However, if the p-value is less than our significance level, it is very unlikely for the null hypothesis to be true, so we reject it.

Different distributions use a different test, for example for Gaussian distributions, t-test is the most commonly used.

3.8.1 Multiple Comparisons

When we have to do multiple comparisons, the standard statistical significance tests become unfit for the task.

Suppose in fact that we are testing a new drug and we want to know if it is helping to cure any of 100 disease symptoms. To do that, we perform a test of statistical significance with a significance level of 5%, thus we have 5% chance to incorrectly reject the null hypothesis if it is true.

However, supposing that the null hypothesis is true for all 100 symptoms, that is, it doesn't have any effect at all, we should expect to incorrectly reject 5 of them. Moreover, the probability to have at least an incorrect rejection is as much as 99.4%.

To solve this issue, we want to adjust our significance level to be more stringent. One very common example is the Bonferroni adjustment, where from Boole's inequality we know that if we have k tests and a target significance of α, adjusting it to α/k for each one of them we are sure for the total error to not exceed α.

3.9 Sensitivity Analysis

Whenever we ask the questions: "Why does my model not work? What are the weak points of my model? Are there useless inputs? How robust to noise is my model?" we want to perform some sensitivity analysis. It is the study of how the uncertainty in the output of a model can be properly split among uncertainties in its inputs.

There are different ways to do these analyses. One widely used is One-Factor-At-a-Time (OFAT): as the name suggests, we modify one factor at a time while not varying the remaining factors to see what effects it produces to the output.

Sensitivity analysis is infamously known to be computationally expensive, thus a full analysis might be unfeasible, especially when we have a big number of factors we are interested in. Still, we are going to do as many experiments as we can, to thoroughly test our models.

CHAPTER 4

PROPOSED APPROACH

Our task is given a configuration and some data regarding such configuration, labeled at every interval with the interactions occurring among any animal, find the model that best infers those interactions. Ideally, to infer some interactions at time t, we can look at our input data at time $t-1$, $t-2$ and so on. So, for every timestamp t to infer, we can use several input data of time $t' \leq t$ to aid our inferences. Figure 2 shows an example of input and output data.

Moreover, we consider *false negatives* to be more relevant than *false positives*: There are three reasons for that:

- The expected interaction network is very sparse. This implies that the trivial model that claims there are no interactions will have reasonably high accuracy, but terrible recall. We want to avoid this scenario, of course, which means to be more biased towards having less false negatives than false positives.
- False positives can be removed manually after the inference, if they are not too many, while false negatives cannot.
- Having a high number of false negatives seriously worsens the quality of the inferred network, and its structure might look like random [25].

Of course, saying "every animal always interacts with everybody else" would give us a recall of 1, but a terrible precision. On the other hand, precision by itself is still not sufficient, because

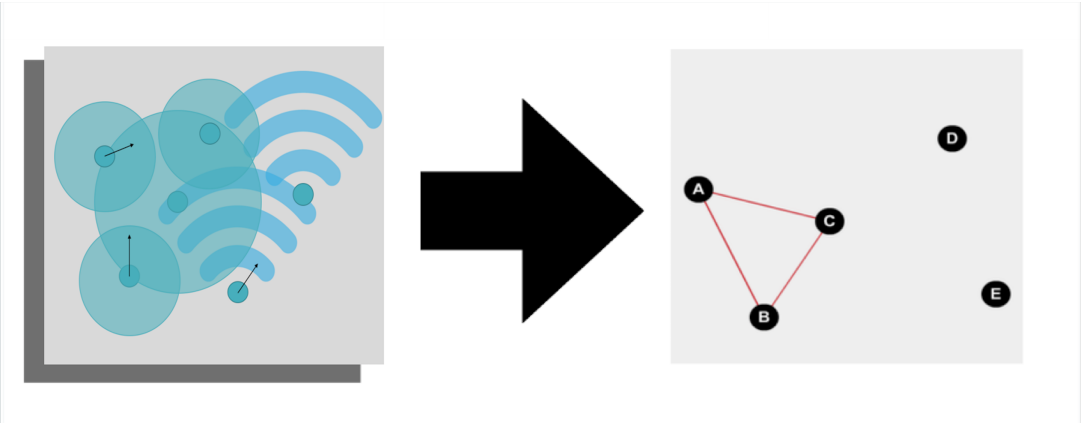

Figure 2: An example of inference: given several timestamps of sensor data, we want to infer an undirected graph of interactions.

we wouldn't be able to realize how many interactions we are missing. Thus, only using either recall or precision would not be enough to understand how good is our model.

This Chapter will further explain our proposed sensors, the model and the framework we built to respectively infer interaction networks in any configuration and understand when this model gives good results and when and why it doesn't. Section 4.1 will describe our new sensors. Section 4.2 explains the proposed model of this work. Section 4.3 will describe the proposed framework.

4.1 Proposed Sensors

We are interested in inferring the interactions occurring among animals, in order to build a graph of interactions for every given timestamp. If we want to infer interactions between two animals, knowing their absolute positions is not strictly necessary. Instead, we would be

interested in their distances. Moreover, we would like to know the distances among all the animals we are observing, which might spot interesting patterns.

Having accurate absolute distances would be ideal, however the state-of-art technology doesn't allow us to have an accuracy as good to be used by itself for such purposes. GPS data, with an accuracy of 3 meters, is unfit to be used for most animals, especially the smaller ones. Adjustments can be done and it has been reported an average accuracy of 30 centimeters [20] on a controlled environment. However, its reliability is very dependent on the signal and obstruction of the line of sight with some satellites is likely to occur. Also, in certain cases such as with small animals, even if we always had the guarantee of 30 centimeters accuracy it would still be too much. What we could use to have more accurate information are pairs of BLE sensors (one transmitter and one receiver) [34], the receiver would sense a signal and attach to it an RSSI value [35], which stands for the strength of the perceived signal. If we tried to convert our RSSI value into absolute distances, it would result to a relatively high error but it would still be more informative than using pairs of GPS sensors.

Moreover, suppose we received two signals from two different transmitters, then we would have two RSSI values and we could compare them. This makes us have arguably more accurate relative ranking distances among sensors: we would be able to say who is closest to me, who is second, and so on. Of course, we would still use thresholds of uncertainty so we might end up considering multiple sensors as equally "second". Ranking distances might be of key importance to spot interactions, as studies show how animals interact with a fixed number of closest neighbors [3].

Also, for both the inaccurate absolute distances and the somewhat accurate relative ranking distances, it may well be the case that a receiver doesn't receive any signal from a specific transmitter. This might either imply that the transmitter is too far away to be spotted, or that the signal was obstructed and wasn't able to reach its destination. Thus, we are in the unfortunate situation where "missing data" might be caused by two completely opposite reasons. This will get our data even noisier than just the inherent errors of precision of the sensors. Still, having redundant sensors, or multiple sensors in different positions should spot these errors and overcome them.

Also, our animals can be equipped with Magnetic Pick-Up (MPU) sensors to either extract accelerometer and/or gyroscope data. We will disregard gyroscope data for our instances and instead sometimes use the absolute value of the speed extracted from the accelerometer. We also have the per-axis speed, however it doesn't really make a lot of sense if we are not planning to use absolute positions but only distances.

Now, let's look at the input data we are going to use; we said we want to transform our RSSI values in absolute and ranking distances. However, both transformations suffer from the absence of received signal. To be able to input something also on these cases, we need to have discrete domains for both absolute and relative distances.

Ranking distances as we defined them are already discrete; we would have the closest ones, the second closests one, and so on. The only thing we would have to add is a ranking representing when we don't spot any signal. We can either say that all those sensors are considered the furthest (i.e. if the furthest spotted is in third position, we would add a fourth position for all

the others), or we would define a default position for all of them (i.e. suppose that we at most are able to discern 5 ranking positions, we would add a sixth position that would refer to all the not received signals). To remain consistent with the concept of ranking, we decided to exploit the first option.

Absolute distances are continuous instead. However, we wouldn't be able to build a not-biased model with continuous distances and many not received signals. What we can do is to split the input in different ranges such as "very close", "close", "medium", "far" and a special distance "very far" for when we don't sense anything.

As mentioned before, when the input should be sufficiently near to be sensed but it doesn't for any reason, it worsens our input data. That is one of the reasons why we would like to add information such as fixed receivers around our grid, to overcome such problems.

Finally, notice how these discretizations give us an ordered domain, that is we are using *ordinal* values, so our model can exploit the knowledge that "medium" is further than "close" and "very close" but closer than "far" and "very far". Likewise for rankings.

Once we have these sensors available, we can decide on using several different configurations; first of all, for all possible configurations we can either have or not have accelerometer data from each animal. We might decide to put a transmitter on every animal and put some receivers on some fixed positions over them (for example onto some beacons). We would then receive ranking and absolute distances of every animal from every fixed position.

Or, we could attach both receivers and transmitters to each animal, thus being able to have distances directly among them. Notice that, being relative, we would represent it with a

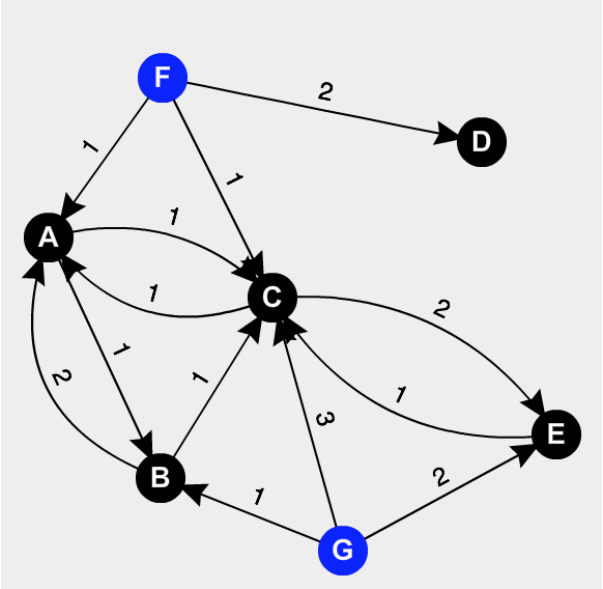

Figure 3: An example of input ranking distances data in a configuration with receivers on both animals and beacons. The black nodes are animals, while the blue nodes are fixed position beacons.

directed weighted graph, because it is not necessarily true that, given two individuals x and y, SensedDistance(x,y) = SensedDistance(y,x).

Finally, we could have both configurations altogether; transmitters and receivers on every animal, and receivers in some fixed positions.

Figure 3 shows an example of input ranking distances data. As we can see, this is a directed weighted graph where distances are discrete positive numbers, there can be multiple arrows of the same rank starting from the same node, but there are no gaps in the rankings. Also, receivers only sense distances and are not sensed. We can see that sometimes receivers do not sense a transmitter, for example B senses C, but C doesn't sense B. Also, ranks between two

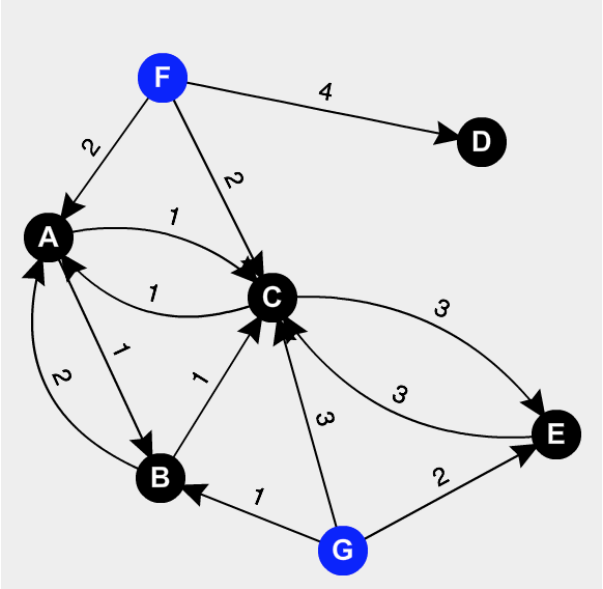

Figure 4: An example of input absolute discretized distance data in a configuration with receivers on both animals and beacons. The black nodes are animals, while the blue nodes are fixed position beacons.

nodes can be different either for noise in the data (pair AB) and for structural properties which have nothing to do with noise (pair CE, where E only senses C thus it is distant 1, but C senses A as well, which is closer, so it says E is distant 2).

Figure 4 shows a possible absolute distance input (already partially discretized) for the same timestamp. What wasn't sensed remains not sensed, however we can see that distances don't necessarily start from 1 (also, they could start from 0 if very close), there can be gaps and distances without noise are more consistent. In fact, even if pair AB due to noise gives a contradictory distance, pair CE this time spots the same distance from each other.

4.2 Proposed Model

Given some labeled data of a specific configuration of sensors, whose values are discrete ordinals and possibly floating point numbers (regarding speed in our examples, but not limited to them), we want to build a model which explains at best our data.

Our output will ultimately be an undirected unweighted graph of interactions. The first question we have to answer is: how do we want to predict the edges of our graph? Do we want to predict pairwise interactions one at a time, or predict them altogether? We propose a mixture of them; we are going to predict pairwise interactions by considering not only data regarding two specific nodes at a time, but all of the available data for all possible pairs.

4.2.1 Pairwise Classification

We want to individually predict every edge. This does not mean that we suppose that the presence of an edge is independent from all the other edges (in fact we suppose the opposite), what it means is that we are going to have one output for each edge, representing whether we consider an interaction occurring or not.

We feed our model with all the input data we have at any given time, and we individually predict every edge from such data. This makes every edge possibly dependent from the others, even if we were to predict such edges one at a time (notice that it is *not* always the case).

To understand why they are dependent, imagine what we would do if we considered every edge to be independent to the other edges. For now, consider a simple configuration, which can be easily generalized to all our cases, where every animal has both a transmitter and a receiver and we want to classify an edge between two of them. If we believed in edge independence,

we wouldn't care about any data regarding any pair of animals but the ones we are classifying now (why caring about the distances between animals 3 and 4, if we are analyzing 1 and 2? Also, why caring about the interaction between 2 and 3, when interested in 1 and 2, if the presence or not of an edge between 2 and 3 does not imply anything for 1 and 2?). Interestingly enough, the concept of "ranking distances" starts being less useful (why caring to know that 2 is second in distance to 1, if we don't care about other interactions?), however it should still be useful by itself, considering that animals are supposed to interact with a fixed number of closest neighbors, as previously noticed [3]. Likewise for speed, we would only care about their two values.

Generalizing, if we had fixed position receivers, we would be interested in only the signals received from the 2 animals we are considering. Knowing that edges can be dependent from other edges makes all our data useful for their classifications.

To actually classify, we consider several models and a baseline approach:

- Baseline: distance and time threshold
- NNs
- SVMs
- Ensemble methods for DTs

4.2.1.1 Baseline Classification

Our baseline approach assumes that both transmitters and receivers are on each animal and disregards the preprecessing of the data into ordinal categories; it wants to find, for each pair

of animals, the maximum distance and for how long that distance must be kept to consider it an interaction.

We only consider the distances between the two animals we care about, thus we can have at most two input values for each pair, and if at least one of them is sensed to be closer than the threshold, we consider them close. If they remain close for the minimum time threshold, we consider it an interaction. If no signal has been sensed, we consider it to be 'far away' and no interaction is spotted.

We want to find the couple of thresholds that minimizes our cost function. The maximum distance threshold is clearly the maximum range of the receivers, while the time threshold is a parameter to be chosen. However, the optimal threshold is hardly ever going to be greater than 3, thus we could select 3 for most instances, and only if 3 happened to be the optimal threshold for a specific instance, we might want to increase this number.

This baseline is simple enough to be able to find the best parameters (global optimum) by a bruteforce approach. Heuristics that find local optima in logarithmic time with respect to the range of the receivers exist, but are not needed in this case.

4.2.1.2 Neural Network Classification

Neural networks seem ideal for our target configurations: they can be built to have any number of numeric inputs as well as any number of binary outputs. Moreover, even if just slightly, mispredictions of one output modify most of the network which is shared by all outputs, thus actually being theoretically dependent from the other outcomes. We used a fully connected feedforward neural network with dropout ReLU neurons for all but the output neurons, and for

output neurons we used a standard sigmoid activation function where being close to 1 means interaction and being close to 0 means no interaction.

One problem with NNs is that their input data cannot be balanced. We can theoretically decrease the unbalance of 0s w.r.t. 1s by completely discarding train data without a single positive classification, but this, especially for networks with a lot of nodes, doesn't solve the problem.

Here, we decided to not discard training data (because it was not needed for our tested instances) and to balance the problem by using cost sensitive learning: we took the Bernoulli Negative Log-likelihood error function described this way where x is the inferred and y is the expected value:

$$error(x, y) = -y \ln x - (1 - y) \ln(1 - x) \tag{4.1}$$

noticing that

$$error(x, y) = \begin{cases} -\ln(1 - x), & \text{if } y = 0, \\ -\ln x, & \text{if y} = 1. \end{cases} \tag{4.2}$$

thus, suppose we have a cost matrix where mispredicting false negatives and false positives have a cost of fn and fp, respectively, where in our case $fn \geq fn$, we want to weight this error function to backpropagate differently those two types of error. First of all, we normalize the two weights so that we are sure to find convergence: $w_{fn} = fn/\max(fn, fp)$ and $w_{fp} = fp/\max(fn, fp)$ so that the weight for the bigger cost will remain 1 and the weight for the

smaller one will be reduced proportionally. Then, we multiply the two parts of our former cost function accordingly:

$$new\_error(x, y) = -w_{fn} y \ln x - w_{fp}(1-y)\ln(1-x) = \begin{cases} -w_{fp}\ln(1-x), & \text{if } y = 0, \\ -w_{fn}\ln x, & \text{if y} = 1. \end{cases} \tag{4.3}$$

This means that if we mispredicted and $y = 1$, then we must have said that $x = 0$ thus we had a false negative and we backpropagate multiplying its error with a w_{fn} factor, likewise for false positive errors, accordingly to our goals. In fact, even if the data is inherently unbalanced, by weighting less the false positives, we effectively help our model in being more balanced.

4.2.1.3 Balancing Data

The remaining classifiers have the way we handled data in common; we created one classifier for every possible output, effectively creating $n(n-1)/2$ binary classifiers which specialize in a single edge. This number, although being linear w.r.t. the size of the output, is easily handled for small to medium size networks of animals, which is always going to be the case for these kinds of inferences. This trick makes our input data easily balanceable: we can always use all the minority class input instances and the same number, randomly sampled, of majority class input instances.

Experimenting this turned out to make us *not need* to use cost sensitive learning, even if it was possible for all the following models.

While there is nothing more to add for our SVM models, we still have to discuss our ensemble methods.

4.2.1.4 Ensemble Methods for Decision Trees

We tried three different ensemble methods with DTs:

- Bagging with a sampling size of 50% of the data and 100 trees.
- Random Forest with 100 trees.
- AdaBoost boosting with a maximum number of 100 trees.

AdaBoost is one of the most used boosting ensembles on the market. While we can't ever suppose a priori which will perform best between AdaBoost and the two more similar Bagging and Random Forest, we don't even know which of the latter two will perform better than the other.

In fact, the main idea of Random Forest is to increase the diversity of the classifiers by selecting only a small number of input features. However, we already know that there are going to be a very, very small number of features which are going to be extremely important for classifications, while the others will impact our results much less. Thus, randomly removing a big number of features implies to very usually not have *any* of the most important features.

We expect this to often result in a worsening of the results, because it will decrease drastically the average accuracy of every classifier, even if increasing the diversity. Still, we don't expect Random Forests to perform badly, because the only condition they have to meet to give good results is to have an accuracy higher than 50% which with a binary classification is extremely easy to achieve, even with not so important features. Thus, it will be interesting to see what impacts best the results. Will it be a higher accuracy or a higher diversity?

4.2.1.5 Markov Assumption and Data Subdivision

Up to now, we only considered the input data of the specific time we want to label. This is a zero-order Markov assumption and it is not necessarily the case that it holds. It could be the case that the distances of our animals of one, two, etc. previous time periods might well alter the likelihood of an interaction. Thus, we would like to try different assumptions. To do that, we need several things:

- A way to compare two results.
- To split the data into train, validation and test sets.
- Understand when to stop increasing our order of Markov assumption.

In order to understand if a bigger order model would eventually overfit the data, we need to evaluate our results and compare them; if we find out that a bigger order Markov assumption behaves worse than a smaller order, we would select the lower order one. This implies that we need to test our models while training, thus we need to have a validation set to not overfit the test set. The test set will only be used at the end of every model tuning.

To evaluate our performances we will use a cost function directly extracted from our cost matrix: the lower the result is, the better our model is.

Finally, it might be the case that using a higher order assumption only slightly increases our results. This is a good hint for understanding that the proper assumption was the previous one, because it means that adding new information doesn't improve our model. But what if a higher order actually improved the results? Using a threshold of improvement is not the best

way to tackle the problem.But what if a higher order actually improved the results? Using a threshold of improvement is not the best way to tackle the problem.But what if a higher order actually improved the results? Using a threshold of improvement is not the best way to tackle the problem. But what if a higher order actually improved the results? Using a threshold of improvement is not the best way to tackle the problem. What we will do is make a test of statistical significance between those two models, using the multiple comparisons approach. If the two models are most likely going to not be statistically different, we would refuse the higher order assumption to select the lower one.

4.3 Proposed Framework

In order to evaluate our proposed models and to perform sensitivity analyses on them, we first need to be able to synthesize our data. Also, we need to do it in such a way that it is easily possible to fine-tune its parameters that impact the reliability of the input data.

4.3.1 Creating the Environment

Accurately synthesizing some real data is a very complex task whose results are very often far from perfect.

The background setup is straightforward; we create a virtual world, where we put inside a given number of animals with some transmitters and possibly receivers and eventually also put pillars in fixed positions with some receivers. We make our animals 'move' a fixed number of times while sampling either the values from our sensors and the occurring interactions every k steps with k not necessarily being equal to 1.

During this process we stumbled across three major design issues:

- Simulate the data retrieved from our sensors.
- Simulate the behavior of our animals
- Label the interactions of our animals

4.3.1.1 Simulating the Data

How do we design our sensors? Which parameters do they take? How are their outputs presented to us?

We want our sensors to give us data similar to RSSI values. Their absolute values are not relevant as long as their relative values are consistent, but we expect our sensors to fail sometimes and even when they don't fail we expect to have some noise in the values received.

We know that our receivers are unable to spot signals further than a given range, thus we could easily say that if the two sensors (one transmitter and one receiver) are distant more than that range, the receiver doesn't receive any signal. But what about when they are not as far away?

If we want to model the possibility of failure, we can just assume that every transmitter has a probability of failure to transmit a given signal and likewise for receivers if needed.

The strength of the signal sent can vary for several reasons, and that can be modeled by using a Gaussian noise which modifies the position we infer (instead of using RSSI values we directly estimate the distances between the two sensors, which requires one less computational step and gives us the same results).

However, when we do receive a signal, it is unlikely to be noiseless. We want to add some noise to it, also, the further it travels, the bigger the noise we can receive. We model this by

using a Gaussian noise with mean 0 and a parameter for the standard deviation which decreases linearly the closer the two sensors are. The good thing about it is that those parameters can be fine-tuned for our sensitivity analysis and fairly accurately describe the real world examples.

4.3.1.2 Simulating Animals Behavior and Labeling

The last two issues will be solved together, but before explaining how, it is important to understand why such things are difficult to deal with.

If we want to synthesize data properly, we want to have inputs as close as possible to reality, thus simulating animals behavior becomes crucial for our framework. Clearly, if animals didn't move at all, or moved at random all the time, their behavior would be useless for our purposes. Our goal is to be able to construct an animal which resembles sufficiently the behavior of a real animal, but not so much in details to have too many arbitrary parameters which could make us jump into false conclusions.

For labeling we have yet again a problem: we can't manually annotate interactions after creating some data. Well, in theory we could: we would first create our input data, then we would have to plot it somehow and to manually say at every timestamp which are the interacting pairs. After all this is what we will have to do with the real data, so what is the problem?

Fist of all, one of the main goals of our study is to estimate *how much* data we need to label in order to build a reasonable model to spot interactions, so we don't know this information here yet. Connected to this rises the second problem; we want to synthesize a variable amount of data, from a lot to very little, an enormous amount of time to perform sensitivity analysis.

Ideally, this type of analysis can be made automatically, thus we really can't manually label our data, and even if we could, it would probably require years.

So how do we automatically label our data? We could say that if two animals are closer than a certain threshold for a certain time threshold, then they are interacting. The problem with that is *it is too simple*. We want to have challenging interactions to be spotted, otherwise we could wrongly believe our models are good, while they are good only for trivial cases.

As anticipated, we solved these two problems at the same time. We performed a very simple instance of agent based modeling [36], by building a Finite-State Machine (FSM) artificial agent intelligence. Agent based modeling is a very wide and complex field, and this work does not intend on going in depth into that. Given that our needs are not having extremely accurate behaviors (which for most animals would be impossible to have, otherwise we wouldn't need to observe them in the first place), a simple FSM with random transitions will suffice.

We decided to model the behavior of every animal as a machine which lives in several different states, and that behaves differently accordingly to its actual one. It must have a movement function which can switch states as well as performing its state behavior.

This FSM must of course be handled case by case, and the only requirement it has, is to have a special *interacting* state. In this state, an agent is interacting with at least another one, and we know who it is interacting with. This makes our labeling of interactions trivial; at any timestamp, we just look at every agent, if it is interacting, we add an edge of interaction with every one it is interacting with. Even if our interacting behavior is somehow directed for now, we can still create an undirected edge (or directed if preferred).

The nice thing about that is we never miss an interaction, being as close as possible to a perfect labeling. If we don't believe it to be the case, we can add uncertainty to the automatic labeling. However, sensitivity analysis about that is far from useful, because it would be useless to say to our labelers 'Be sure to be accurate at least 87% of the time!' because we are supposing that they are as accurate as they can be and they wouldn't know to be mistaken most of the time.

Finally, this gives us the possibility to actually label a specific behavior not by using our input data, thus our models will need to reverse engineer the correlations between our input data and the labelling, which is, said in a less fashioned way, performing supervised learning.

CHAPTER 5

CASE STUDY

We propose a case study where we synthesize some data and do some sensitivity analysis on it, to show example of how our framework and model work and evaluate our methodology while possibly giving some insights on general properties of dyadic interactions.

As anticipated, we need to create a random agent FSM. We decided to make one very simple agent which lives in four different states, and that behaves differently accordingly to its actual one. We made it fairly simplistic to avoid making assumptions of any type. It has a body and moves not much more than its body length for every timestamp, but we didn't assume how it senses any other agent, thus it has no explicit sensory system. Every number you will see is a specific instance of our more general parametric model. Of course, their numbers will be pure (without a dimension such as distance in meters or time etc.), but for the readers' clarity we will interpret them as centimeters.

The four states actually implemented are the following:

- Normal
- Hungry
- Sleeping
- Interacting

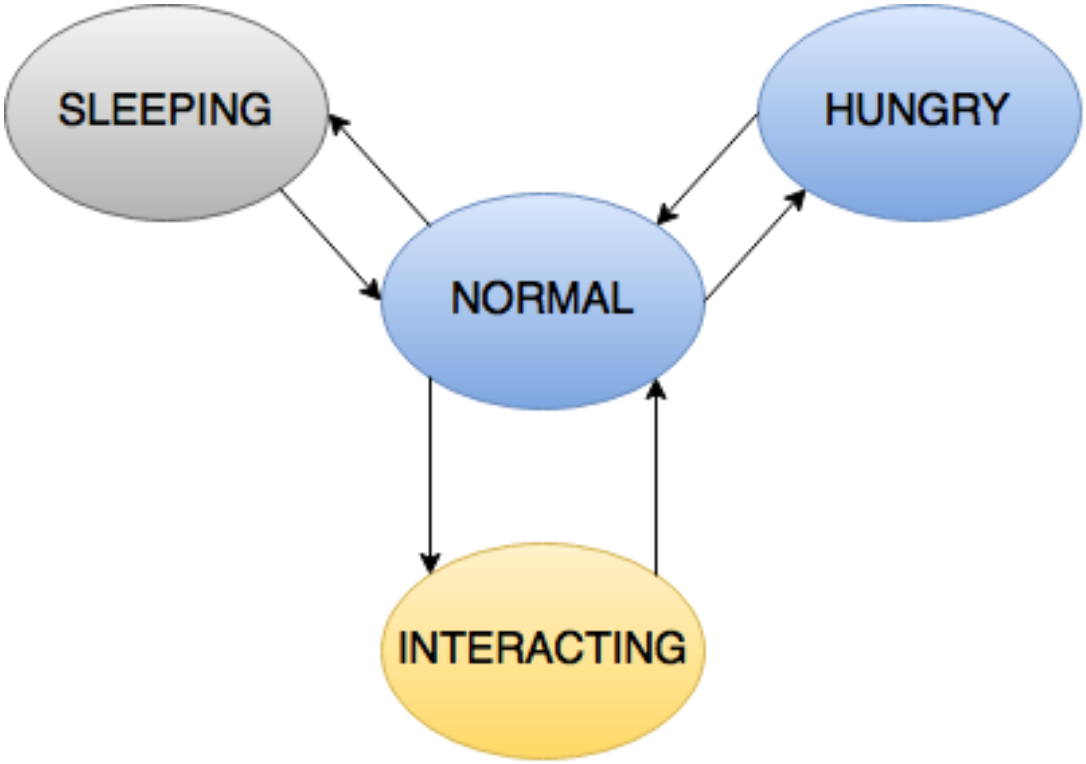

Figure 5: The actually implemented FSM of our agents. Transitions from states are random, although their chance may vary.

Figure 5 shows the FSM of our agents. The Normal state is the core of our automaton: it can either behave 'normally' or it can switch to any of the other three states. If it behaves normally, it can either stay still or move, whereto is irrelevant.

Randomly, the agent can switch from Normal to Hungry state. When it does, its priority becomes to feed. To implement this behavior we created in our virtual world a spot where our agents can eat food (and ideally drink), when an agent is hungry, it doesn't anymore behave at

random, but it instead moves towards the food spot of the fence and eats there to its heart's content (conveniently modeled by a number). After that, it goes back to Normal.

Or, starting from Normal state, the agent can decide to go to sleep. If it goes to sleep it doesn't move at all for a variable number of turns. After that, it wakes up and goes back to Normal.

Finally, it can try to start an interaction. If the agent is in Normal state and wants to interact with another agent, it can look around itself. If there is another agent close to its position, it can start interacting with them. This has a completely different behavior than the general case; the interacting agent starts moving towards and around the other agent (simulating either a poke or some active jumps around the other individual). Moreover, being playful increases the chance to interact with any other close agent. If it is interacting with more than one agent at the same time, it randomly decides who to torment. Also, it randomly decides whether to drop an interaction with an agent at any time and, if there are no more interactions active, it goes back to Normal.

In details, we implemented our agent behavior with these parameters:

- Normal state is the starting state of any agent.
- While in Normal state, it can transition to Hungry or Sleeping states with 5% chance each, and if sufficiently close (there is at least one other agent closer than 50 cm to our target agent) it has a 20% chance to transition to Interacting state.
- An agent in Sleeping state stays there for a random number of timestamps varying from 3 to 20.

- An agent in Hungry state, once reached a feeding spot, stays there for a random number of timestamps varying from 1 to 4.
- While in Interacting state, an agent can start interacting also with another agent very close to it (less than 30 cm) with a 30% chance. However, any occurring interaction can drop with a 40% chance at the beginning of every timestamp.

The interacting likelihood is a parameter we will modify later, to make interactions occur less and less often to make sensitivity analysis on our models.

Modeling an animal this way is reasonably similar to a real animal behavior (which still needs several parameters to be set to better resemble them, such as sleeping time, movement chance and so on), but not as specific to be close to one specific type of animal, and does not assume anything like the line of sight, hearing and smell of an animal, which is usually unknown.

Notice that this behavior construction is made purposely to have hard time inferring interactions from our model. It is most likely going to be harder than the real labels, so that we can give a lower bound for our parameters to surely have acceptable results even on simpler cases. Of course, for this to be the case, the right types of sensors must be given for the specific interaction we are interested in (suppose we are interested in a type of interaction between two animals that is performed by some arms' movements, and we don't have any accelerometer data in the arms of the animals. That interaction is impossible to be spotted with no false positives, regardless of the model).

However, this is only a draft and can be either made simpler or more realistic. Some examples for improvement are: to really model the starving factor of an animal as well as its fatigue which directly affect which state is most likely to be selected. To model the orientation of an animal and its field of vision (otherwise the orientation would be useless). To say that if you are sleeping and an interacting animal pokes you, or even every time a animal stumbles across you, you have a chance to wake up. Or, if some animal is interacting with you and you are not sleeping or hungry, you have a very high chance to either avoid it, or start interacting as well. As anticipated, the last improvements regarding interaction were not implemented to make the inference harder and more general, but if you are willing to have a more realistic interaction, it is implementable in our framework in theory. However, once the complexity of the agents becomes reasonably big, we suggest using specific tools for agent based modeling such as NetLogo [37]. Nevertheless, care is recommended if following that route; setting the wrong (very specific) parameters w.r.t. the real animal's might result in completely misleading results.

We will put our agents in a M×N (5×5 is what we arbitrarily decided) m^2 fence. Whenever we say we use fixed position beacons in this work, we will refer to a configuration where beacons are positioned in a very redundant way; they are positioned in a grid, where their distance from either a wall or another beacon is exactly as big as its range, thus we are confident that an agent would be sensed by at least two beacons in most places, if there weren't reliability issues. Figure 6 shows our target positions of beacons.

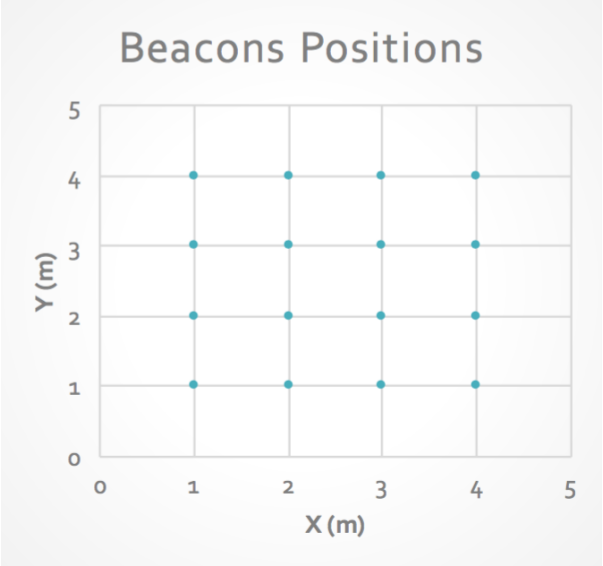

Figure 6: Positions of beacons in the synthesized world. The blue dots represent the beacons containing the receivers.

First, we have to estimate some parameter thresholds regarding error and reliability of our sensors. We don't suppose our sensors (both transmitter and receivers) will be 100% reliable. Thus, we will estimate that their reliability, that is the probability to send/receive a signal, will never exceed 95%. Also, in our synthesized world we could be 100% sure of the distances between the two sensors, thus we add positional errors being sent from the transmitters and a distance error being received by the receiver. They are both coming from a Gaussian noise, where we give as input parameter the standard deviation.

A reasonably starting value for these errors is, given that the maximum sensed distance is 100 cm, we give a transmitter standard deviation of 10, and receiver distance standard deviation

of 15. Thus we suppose our signals to be reasonably noisy to begin with because it wouldn't make much sense to suppose no noise in our default case.

For all our evaluations, we will always use a test size of 2000 time frames. This is large enough to give us relevant estimates of our results, but more importantly it is necessary to always have the same arbitrary size once decided, to be able to compare results with different parameters. Also, we will use a cost matrix where false positives are weighted 1, false negatives are weighted 3, and true positives and negatives have 0 weight whenever we don't specify otherwise. The cost matrix is a sometimes crucial parameter which has to be manually inserted. There is no objective way to define the best cost matrix, however a good rule of thumbs looks like to weight false negatives close to the inverse ratio of their proportion with false positives.

However, input size is the most important parameter to estimate as first; the lower the input size needed, the faster all our other analyses will be.

Also, as the generation of the data is a random process, as well as some inner calculations of our classifiers while being trained (think of how a Neural Network gets trained, or a Random Forest, for example), we want to repeat every analysis multiple times and average results. We decided 5 to be the number of repetitions, only for computational constraints (some of our analyses are too long to be repeated more than 5 times for every classifier). Thus, whenever we show evaluation measures, they are all averaged from 5 runs. Given the variability of our training phase, we believe that a greater number of tests for each model would be more insightful. However, as you will see shortly, the variance of our models does not seem like to be a primal concern.

Finally, special care must be taken with the parameters regarding our agents; they are very complex to be finely tuned. At first, we are supposing to have 6 (a partially arbitrary starting number, driven by the fact that a small number of agents should suffice in showing patterns and that real world labeling would start with a small number of animals first, however why exactly 6 was a preference of the author) agents which interact relatively often when they are close (20 % as stated before) in a time frame. This can and will later be significantly reduced to see how our models behave with much more unbalanced data. Also, this work will always put one receiver and transmitter on every single agent, although future analyses might be done to see what happens with different configurations.

To give an insight of what data without noise would give as a result, we performed two tests, one without (Table II) and one with beacons (Table III), where there is no noise whatsoever. We will also show its uncertainty, so we performed 10 runs each (twice what we will do after this) to be more precise on the analysis.

TABLE II: EVALUATIONS OF SEVERAL MODELS FOR A CONFIGURATION WITH NO BEACONS AND A TRAINING SET OF 10000 SAMPLES WITHOUT NOISE

Method	Cost	Accuracy	Precision	Recall
Baseline	12371 ± 643	0.677 ± 0.019	0.390 ± 0.014	0.800 ± 0.018
SVM	13857 ± 630	0.638 ± 0.022	0.356 ± 0.017	0.776 ± 0.016
NN	8793 ± 1184	0.782 ± 0.018	0.506 ± 0.008	0.834 ± 0.049
Bagging	$\mathbf{7753} \pm 1091$	$\mathbf{0.784} \pm 0.028$	$\mathbf{0.509} \pm 0.021$	$\mathbf{0.906} \pm 0.016$
Boosting	8169 ± 966	0.774 ± 0.025	0.496 ± 0.02	0.896 ± 0.015
Random Forest	8136 ± 868	0.775 ± 0.021	0.497 ± 0.018	0.896 ± 0.019

TABLE III: EVALUATIONS OF SEVERAL MODELS FOR A CONFIGURATION WITH BEACONS AND A TRAINING SET OF 10000 SAMPLES WITHOUT NOISE

Method	Cost	Accuracy	Precision	Recall
Baseline	12085 ± 610	0.687 ± 0.013	0.397 ± 0.009	0.796 ± 0.016
SVM	10759 ± 474	0.710 ± 0.014	0.422 ± 0.008	0.846 ± 0.010
NN	19720 ± 1509	0.731 ± 0.053	0.196 ± 0.144	0.121 ± 0.094
Bagging	7863 ± 859	0.782 ± 0.021	0.504 ± 0.018	0.900 ± 0.022
Boosting	8147 ± 704	0.776 ± 0.017	0.496 ± 0.017	0.892 ± 0.021
Random Forest	$\mathbf{7763} \pm 729$	$\mathbf{0.784} \pm 0.019$	$\mathbf{0.506} \pm 0.021$	$\mathbf{0.904} \pm 0.017$

We can see from Table II and Table III that bagging performs better than any other classifier with no beacons, and random forest performs better with beacons. This might be because when we significantly increase the number of input features, having a higher variability in our default decision trees is better than having a higher accuracy. However, we can also notice that our top performing classifiers do not gain much from having beacons when there is no noise, hinting that fixed positions do not help for spotting interactions. We can also notice that NN perform terribly with beacons, and we will discuss about that later, and that SVM improves significantly in both precision and recall. Finally, notice that variance of our models is not of crucial concern, especially for recall.

5.1 Estimating the Training Size

Estimating the size of the training data needed to have reasonable results is extremely important. We will have to estimate them for at least two configurations:

- Without fixed position beacons holding receivers.

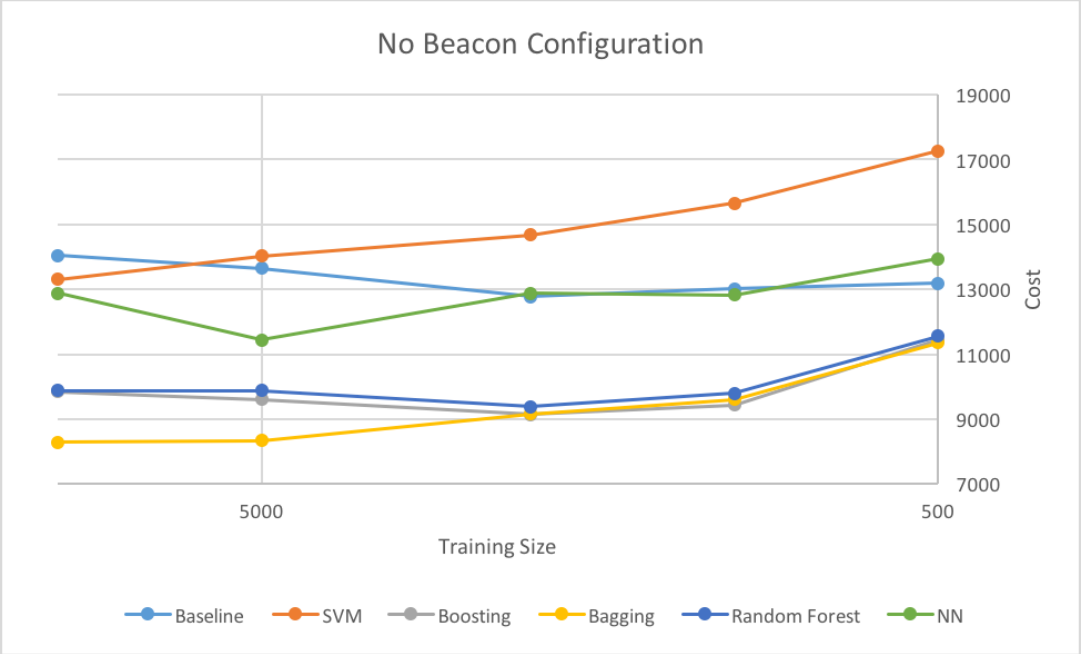

Figure 7: Cost evaluation of our selected models with different training size and with no beacons.

- With them.

The reason is simple; the size of the input features significantly increases when we add those beacons, thus it may be the case that a small size of data for the first configuration is not enough for the second.

We will start without beacons and see how our models behave with gradually less data. We start from 10000 time frames which we believe it to be reasonably high (it would mean that 10000 different time frames would have been manually labeled, which is unlikely), and go downwards in a non linear fashion.

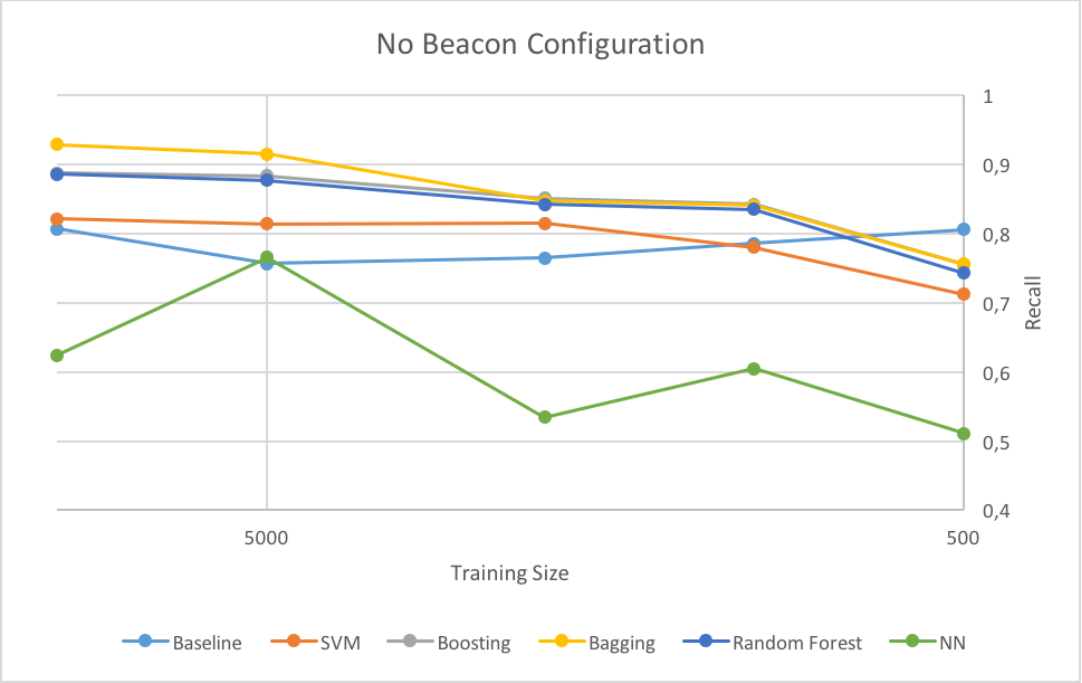

Figure 8: Recall evaluation of our selected models with different training size and with no beacons.

The first thing we can notice is that even if having a configuration with no noise improves the results, it doesn't for the top performing classifiers in a significant way.

Looking at Figure 7 we can see different things: although as we could expect the more data we have, the better we predict, there appears to not be significant difference between 10000 and 5000, and starting from 5000 going downwards the cost evaluations of most classifiers drop significantly.

In particular, the best classifier appears to be Bagging when we feed it with a lot of data (10000-5000) and there is no significant difference among the three ensemble methods with lower

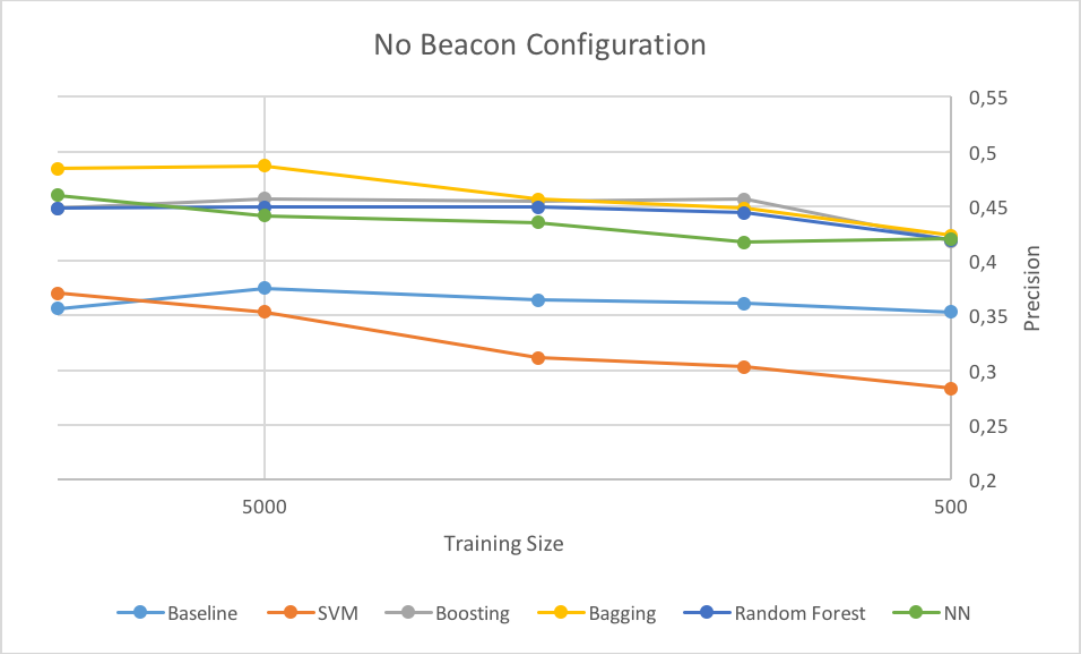

Figure 9: Precision evaluation of our selected models with different training size and with no beacons.

data. As expected, the Baseline performs consistently with a little variety in the results. Also, SVM has the worst performance consistently among the not baseline classifiers and doesn't seem to perform significantly better than the baseline even with a lot of data. Finally, NN performs better than the baseline with a lot of data, but struggles with less.

Figure 8 shows us the recall evaluations of our models. It appears that NN still struggles to have a high recall even with cost sensitive learning, while all other models lose only a small amount of recall, exception made for the 500 training size configuration.

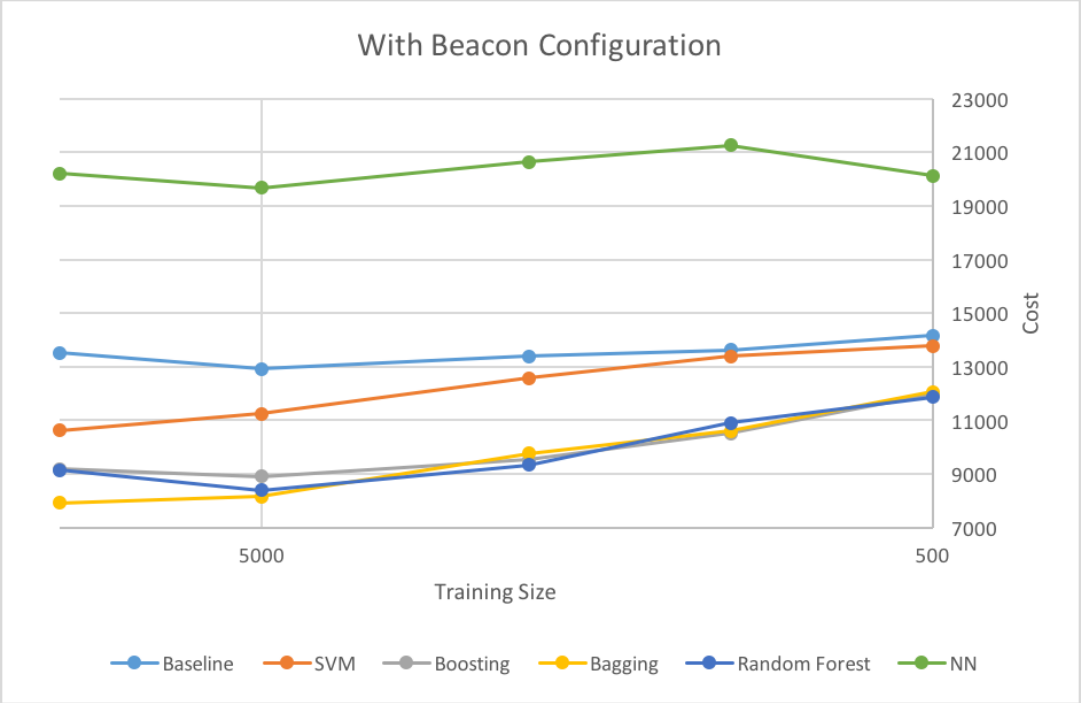

Figure 10: Cost evaluation of our selected models with different training size and with beacons.

Figure 9 shows the precision of our models. Interestingly NN is competitive with the ensembles methods here, and, exception made for Bagging and SVM, they keep a stable value by decreasing the training size.

Adding information from fixed position receivers, as we can see in Figure 10, shows a little improvement on bagging, a remarkable improvement on SVM, which appears to perform extremely better than the baseline with a lot of data, no improvement for both boosting and random forest as well as no improvement for bagging with little data, and a surprising worsening of the performance for NN, most likely caused by underfitting.

Summing things up:

- There does not appear to be an extreme advantage on using fixed position sensors for the top performing classifiers, with this little noise in the data, even if it helps a bit.
- NN tuning of layers and number of neurons appears to be extremely difficult and time consuming to be used in an automatic way, for so many different configurations.
- Although having more data *does* improves the results, we cannot always use a large amount of data for computational constraints as well as for comparisons with real world labeling, thus we will use 2000 samples from now on, which still gives reasonably good performance while maintaining small enough computation time.

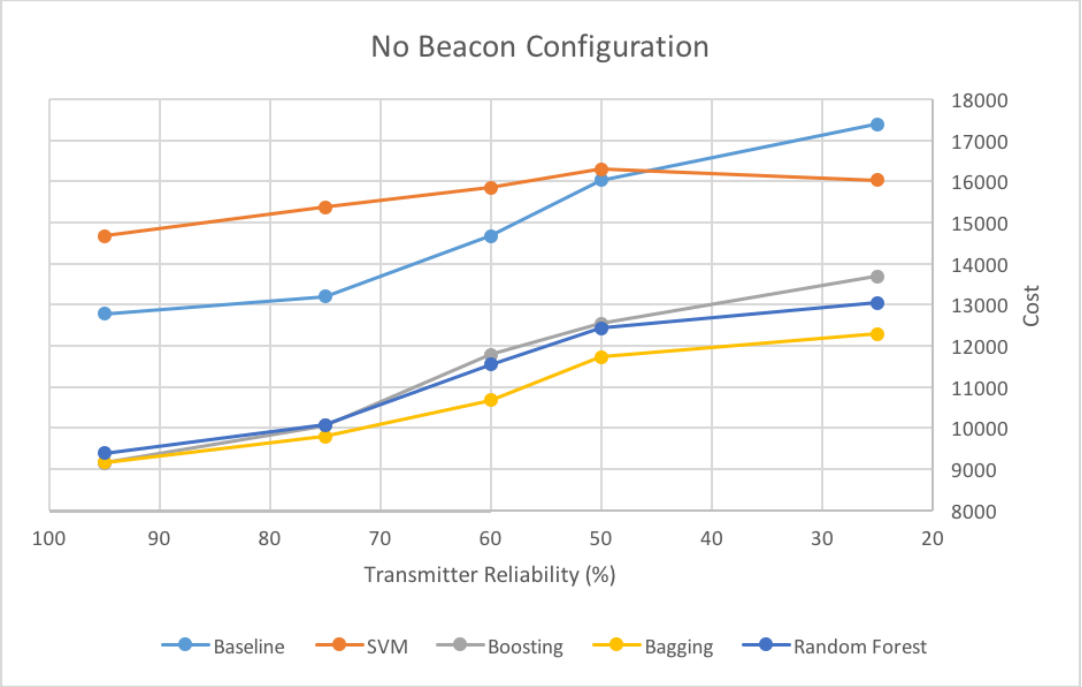

Figure 11: Cost evaluation of our selected models with different transmitter reliability and with no beacons.

5.2 Sensor Reliability

How reliable must sensors be to still have reasonable results? In this Section, we are interested in dropping the reliability of a transmitter to send a signal. If it doesn't, in that time frame no receiver receives it.

We will not use NNs anymore, as they appear to not be fit for the task, and we will decrease the reliability in a non linear fashion again. The first analysis is done with no beacons.

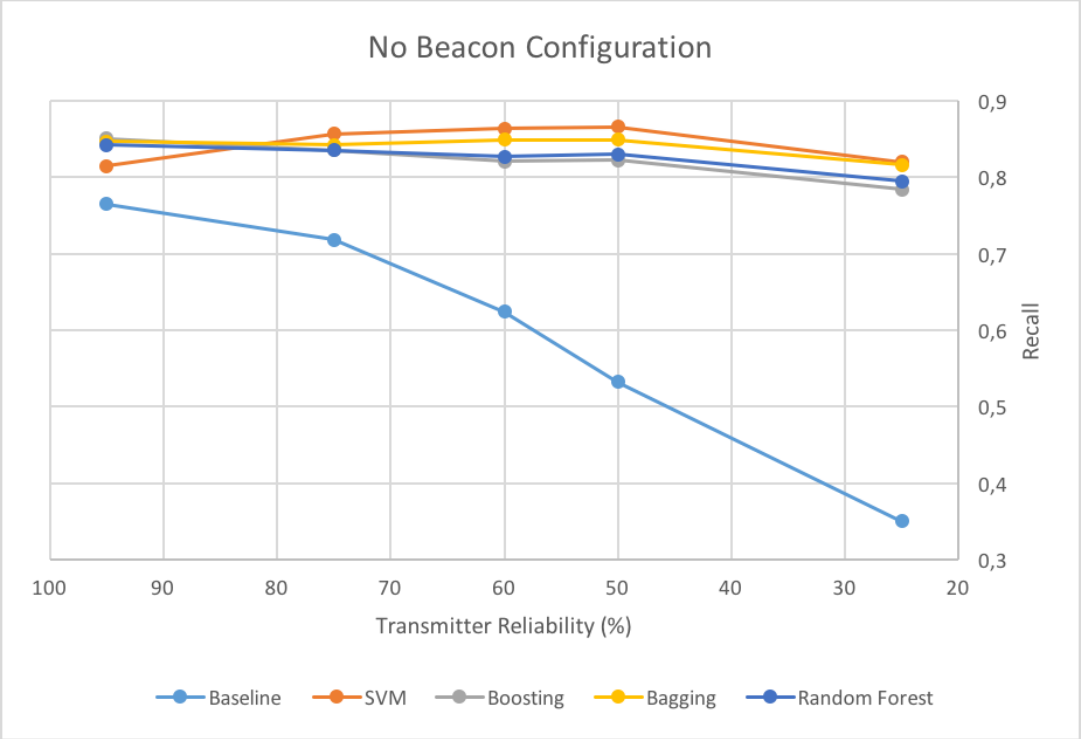

Figure 12: Recall evaluation of our selected models with different transmitter reliability and with no beacons.

As expected, the performances drop for all classifiers (Figure 11). In particular, the precision is harmed significantly by this noise, as shown in Figure 13, for our ensemble methods, while both SVM and the baseline keep their (low) performance stable.

Surprisingly, recall remains very high. In particular, the ensemble methods are able to make reasonable inferences even with extremely unreliable data, as seen in Figure 12, even if the precision falls behind 40%, maintaining a 80% recall.

Finally, our baseline's recall drops drastically, making it not usable at all.

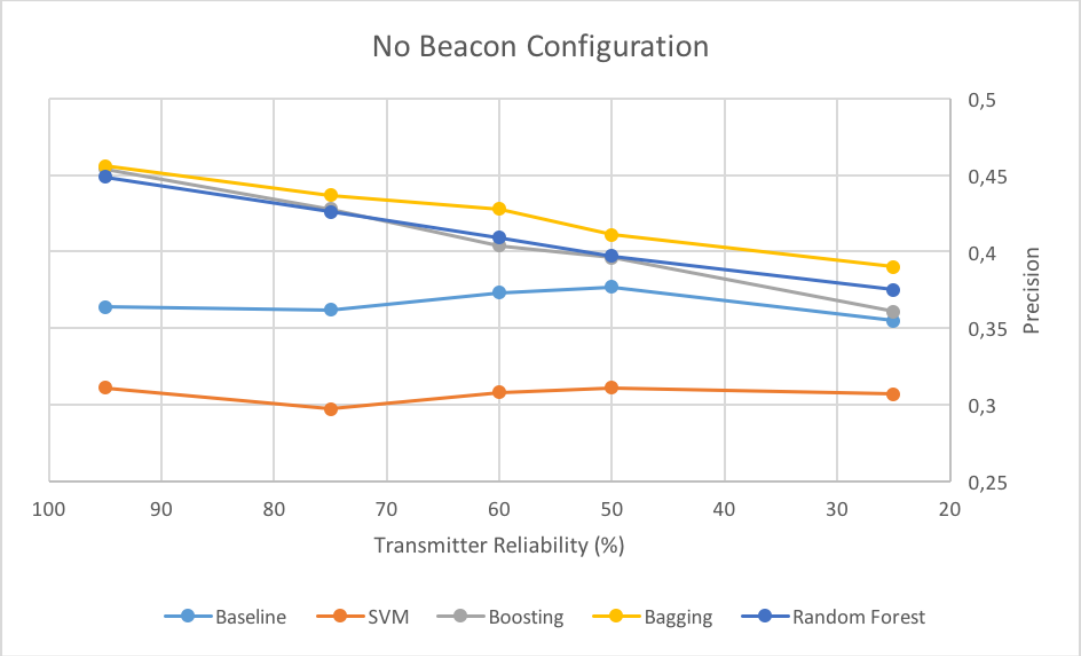

Figure 13: Precision evaluation of our selected models with different transmitter reliability and with no beacons.

Unexpectedly, adding fixed position receivers does not help even with unreliable transmitters for all models but SVM, which improves significantly (Figure 14).

However, we can conclude that these receivers are not needed in such a configuration, by looking at the top performing classifiers again.

5.3 Sensor Error

Transmitter and receiver errors are modeled in two different ways and have two different effects on our input data. We will first increase the noise on solely the transmitter signals. Second, we will analyze the effects of the noise in the receivers.

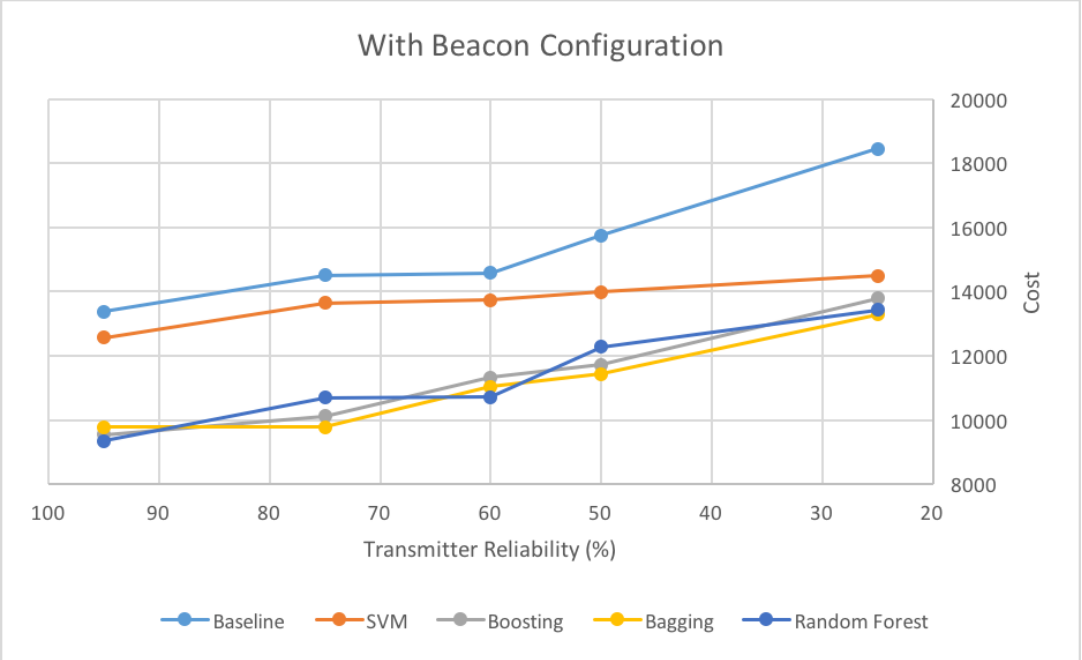

Figure 14: Cost evaluation of our selected models with different transmitter reliability and with beacons.

5.3.1 Transmitter Error

It appears that this kind of noise heavily impacts the performance of all our models in a configuration with no beacons (Figure 15). The baseline's performance in particular is terrible, while our models still keep respectable results. In particular, recall remains untouched (Figure 16), while precision drastically drops (Figure 17).

Apparently, adding beacons does not mitigate significantly the errors caused by this type of errors (Figure 22), which was to be expected because beacons have no transmitters.

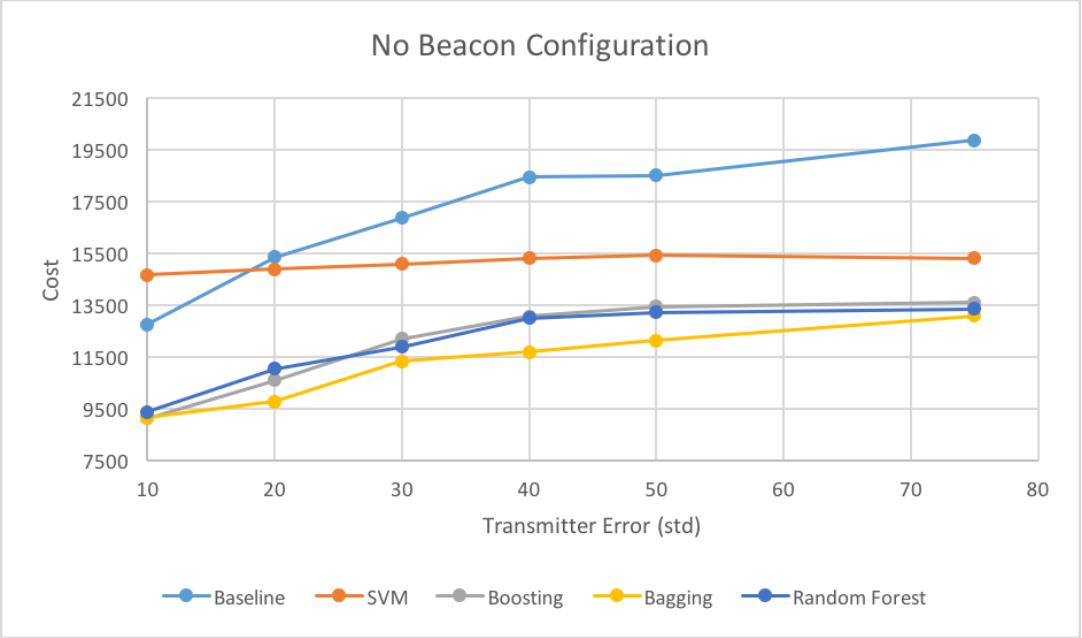

Figure 15: Cost evaluation of our selected models with different transmitter error and with no beacons.

5.3.2 Receiver Error

With respect to the transmitter noise, this kind of noise impacts significantly less the performances of our classifiers (Figure 19). Recall still performs well (Figure 20), while precision decreases with a 'lesser' slope (Figure 21).

Again, the gain of accuracy adding beacons does not justify their cost even to mitigate this kind of noise (Figure 22).

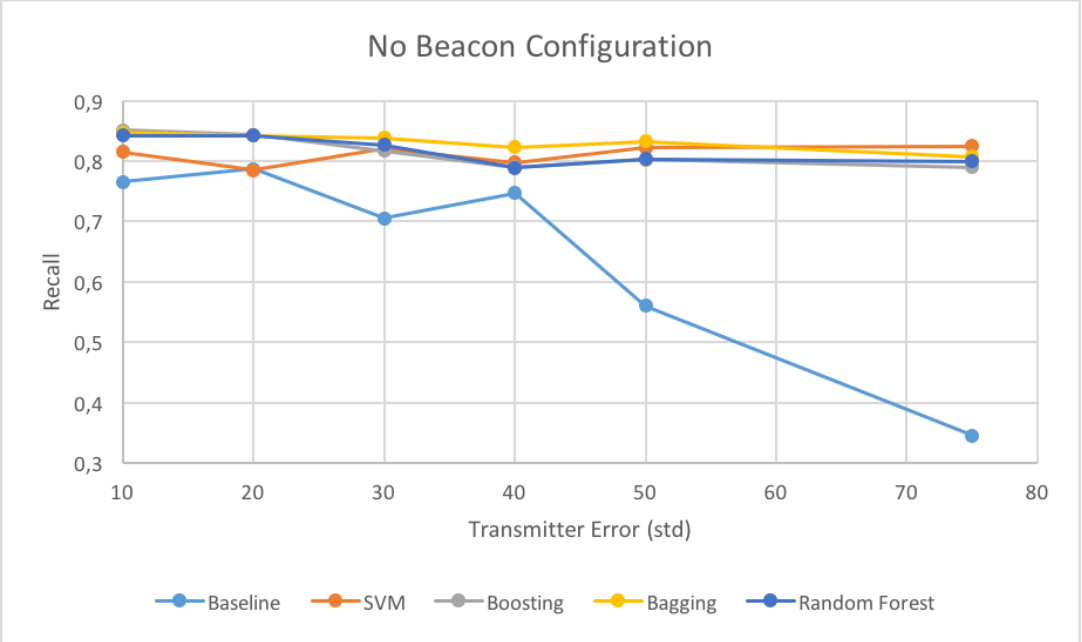

Figure 16: Recall evaluation of our selected models with different transmitter error and with no beacons.

5.4 Feature Relevance

We are interested in knowing what are the input features that impact the most on our models, to eventually trim down the input size.

We split our input features into:

- Absolute pairwise distances.
- Ranking pairwise distances.
- Speed of agents.

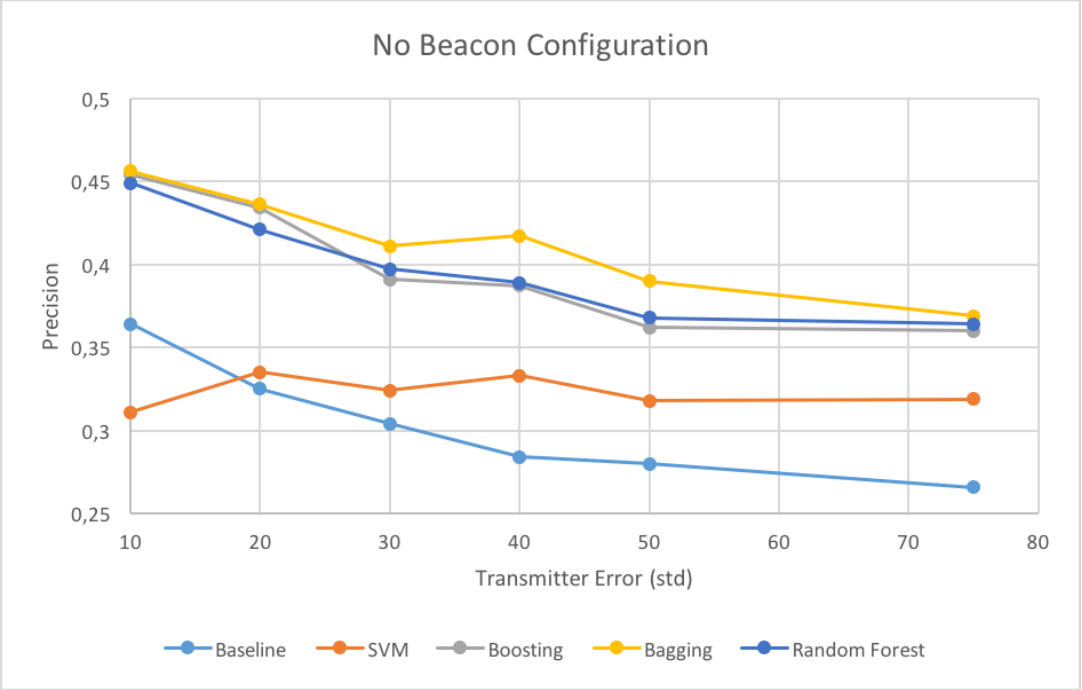

Figure 17: Precision evaluation of our selected models with different transmitter error and with no beacons.

Exception made for the speed features, which are useless by themselves, we tried every combination of features with our standard configuration of 2000 training size, standard noise and with and without beacons.

As we can see from Figure 23, it appears that our models without speed features are worse than the baseline. Also, the ranking distances by themselves give significantly worse results than the ones obtained with absolute distances alone but they are still very good at both precision and recall, and the combination of the two does not improve significantly with respect to just having absolute distances.

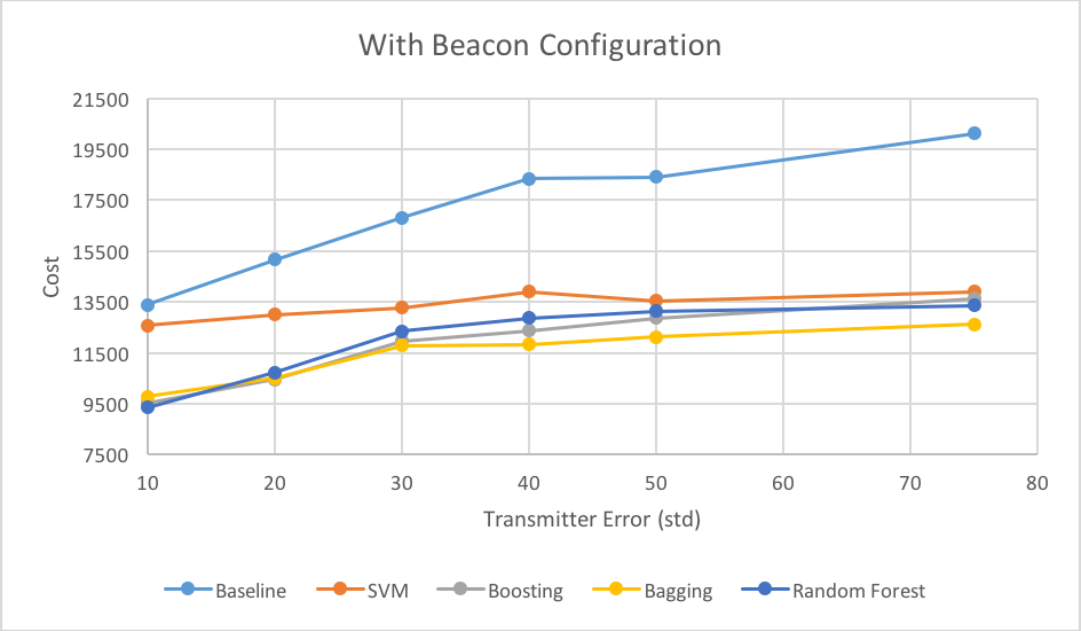

Figure 18: Cost evaluation of our selected models with different transmitter error and with beacons.

This result is interesting, and it is probably ought to the artificial labeling of our data. It is extremely reasonable to suppose that humans would label in a way much more similar to relative ranking distances than absolute ones, thus future work might want to see whether with different labels the situation might change.

Figure 24 and Figure 25 show recall and precision evaluations, respectively.

Figure 26 shows the cost evaluations with beacons. Exception made for the surprisingly good results given by boosting and bagging with the configuration of absolute distances and speed, outperforming the complete features configuration, everything said for the configurations with no beacons remains true.

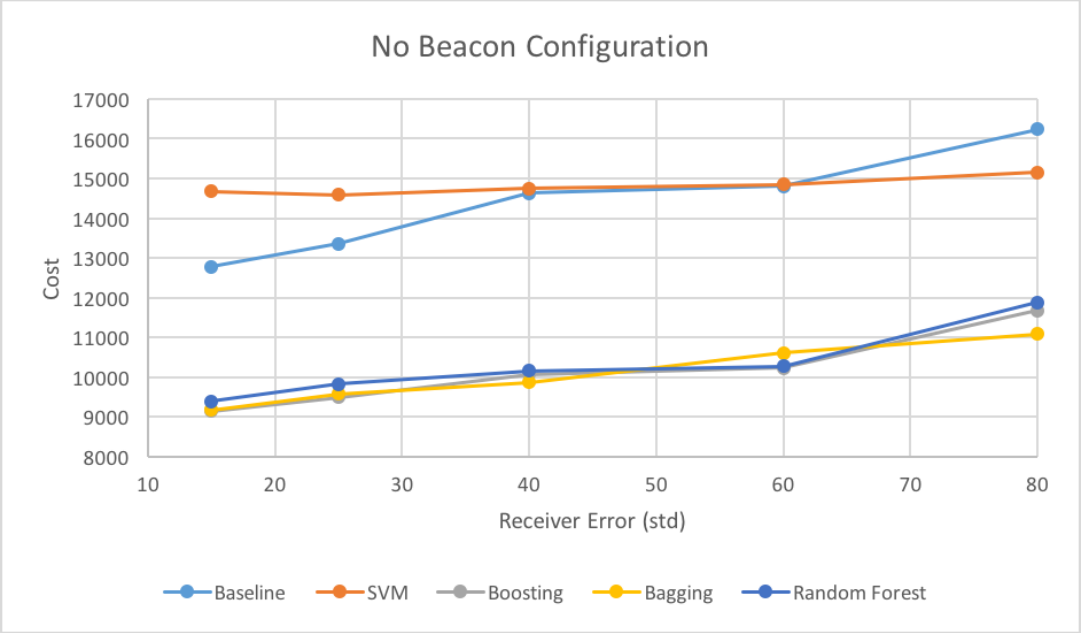

Figure 19: Cost evaluation of our selected models with different receiver error and with no beacons.

It appears that the best configuration is found with beacons and absolute distances with speed, using either boosting or bagging, this time with an improvement with respect to the configuration with no beacons. However, adding beacons generally requires more data and more computational training time.

5.5 Adding Agents

We are interested in knowing what happens when we increase the number of agents in our fence. Up to now, we have always had 6 agents in our 5x5 meters squared environment. Now, we will increment this number. From now on, we will only use absolute distances and speeds, as it was shown to be more than enough in these configurations. Our cost evaluation starts not

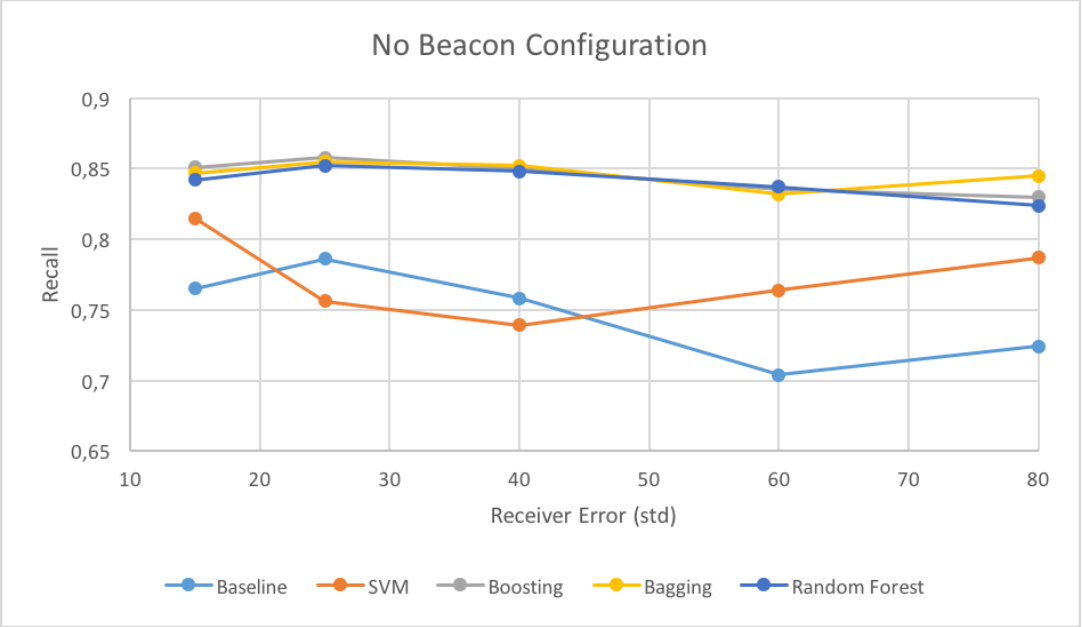

Figure 20: Recall evaluation of our selected models with different receiver error and with no beacons.

making sense here, because increasing the number of agents significantly changes the expected cost, thus we will only show recall and precision. Also, this task turned out to be extremely time consuming, thus we didn't perform this evaluation for the configuration with beacons, and stopped after it took more than one day to compute a complete cycle.

Figure 27 shows recall drops drastically for SVMs, a little for our ensemble methods and surprisingly it increases for our baseline. Figure 28 shows an increase of precision for all our classifiers, baseline included.

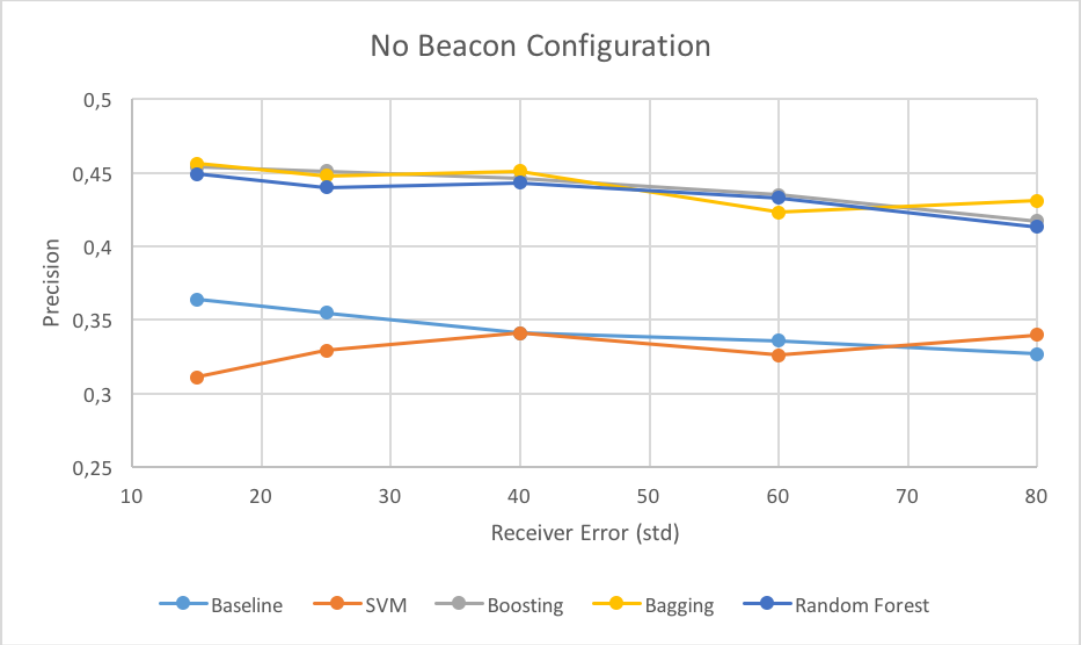

Figure 21: Precision evaluation of our selected models with different receiver error and with no beacons.

It appears that the more agents we add, the more likely we are to have an interaction, this explains the linear increase in performance for our baseline. Still, our ensemble methods perform consistently better than it.

We are curious to know what would happen in a larger environment, or with even more agents, or with a different agents behavior.

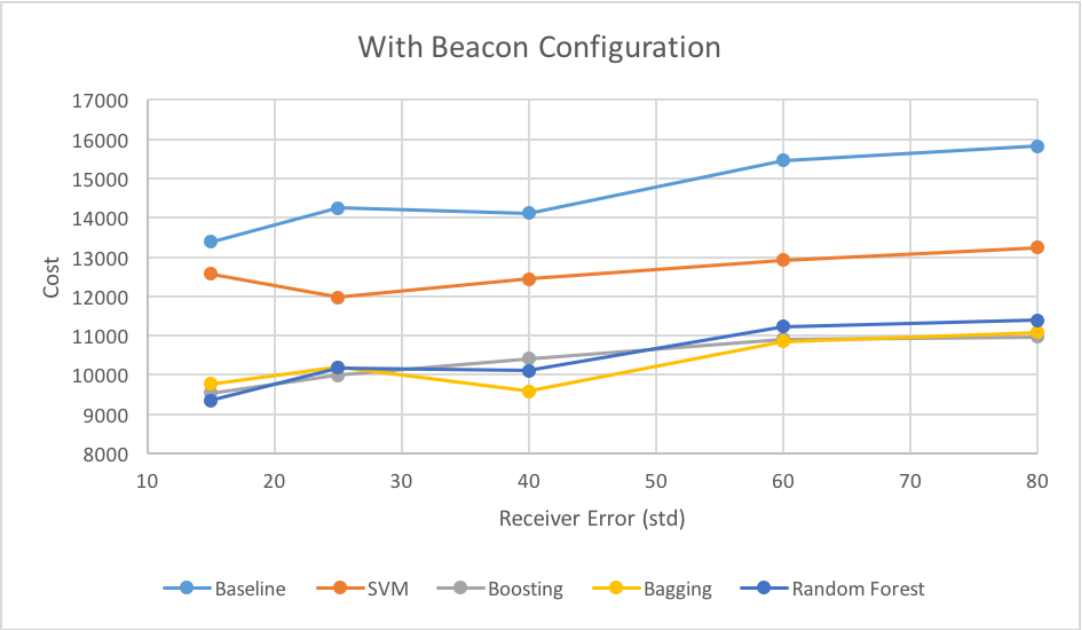

Figure 22: Cost evaluation of our selected models with different receiver error and with beacons.

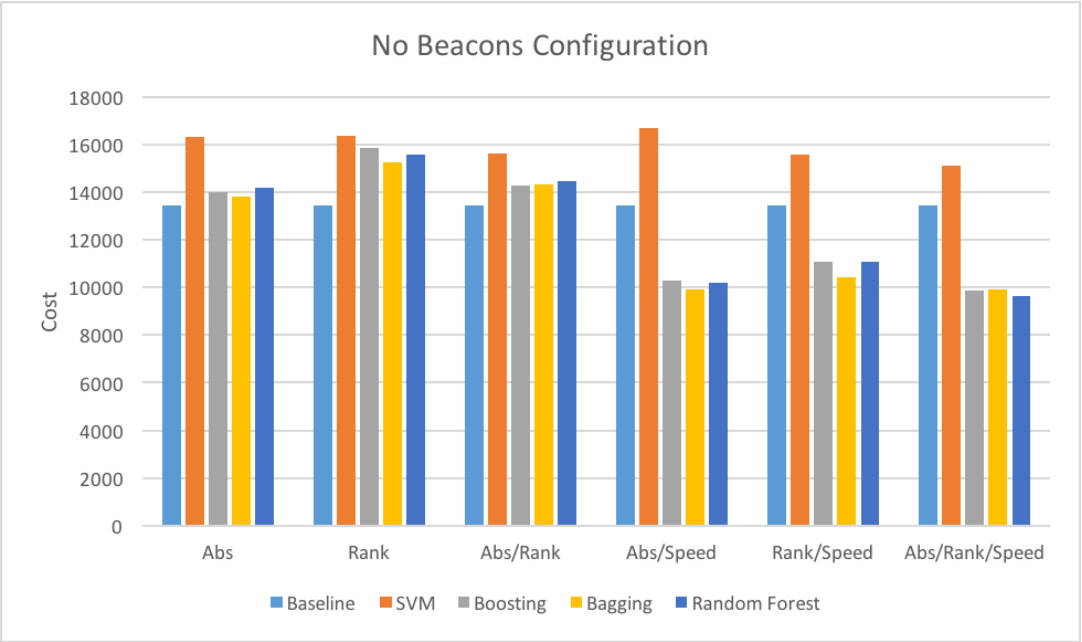

Figure 23: Cost evaluation of our selected models with different features and no beacons.

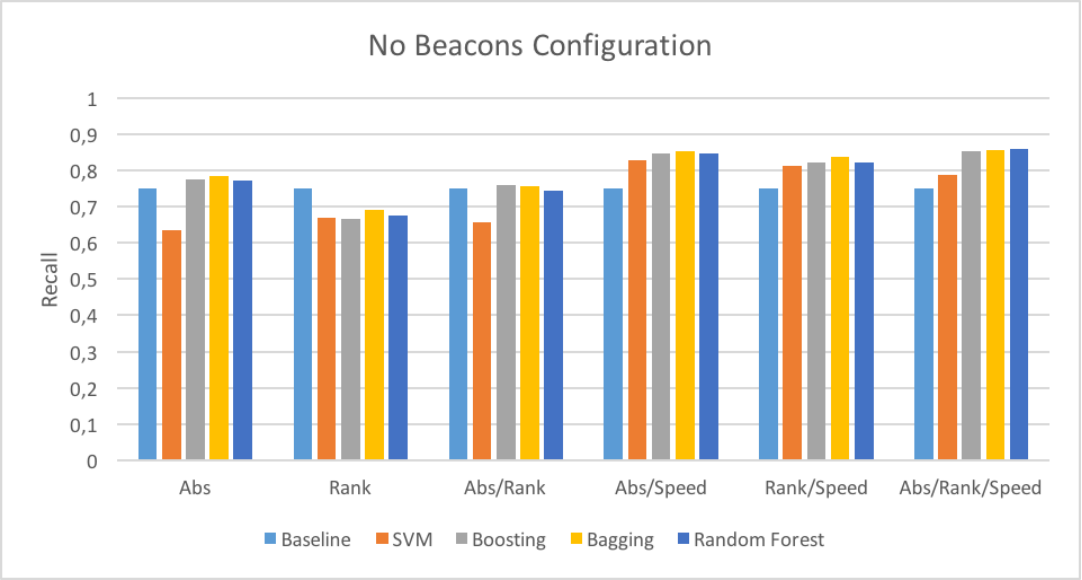

Figure 24: Recall evaluation of our selected models with different features and no beacons.

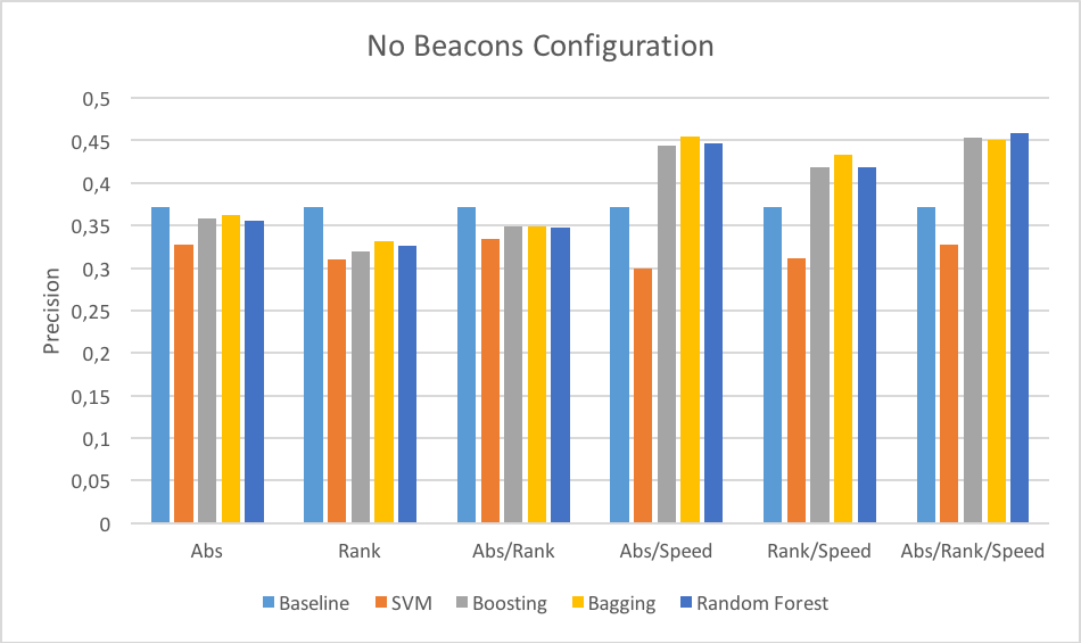

Figure 25: Precision evaluation of our selected models with different features and no beacons.

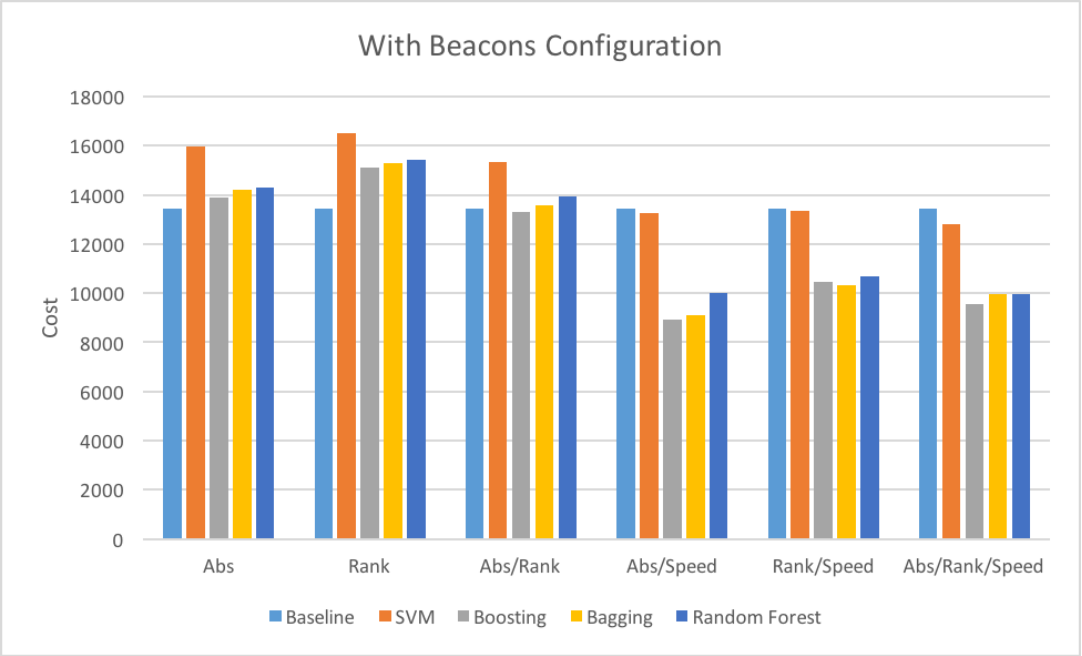

Figure 26: Cost evaluation of our selected models with different features and with beacons.

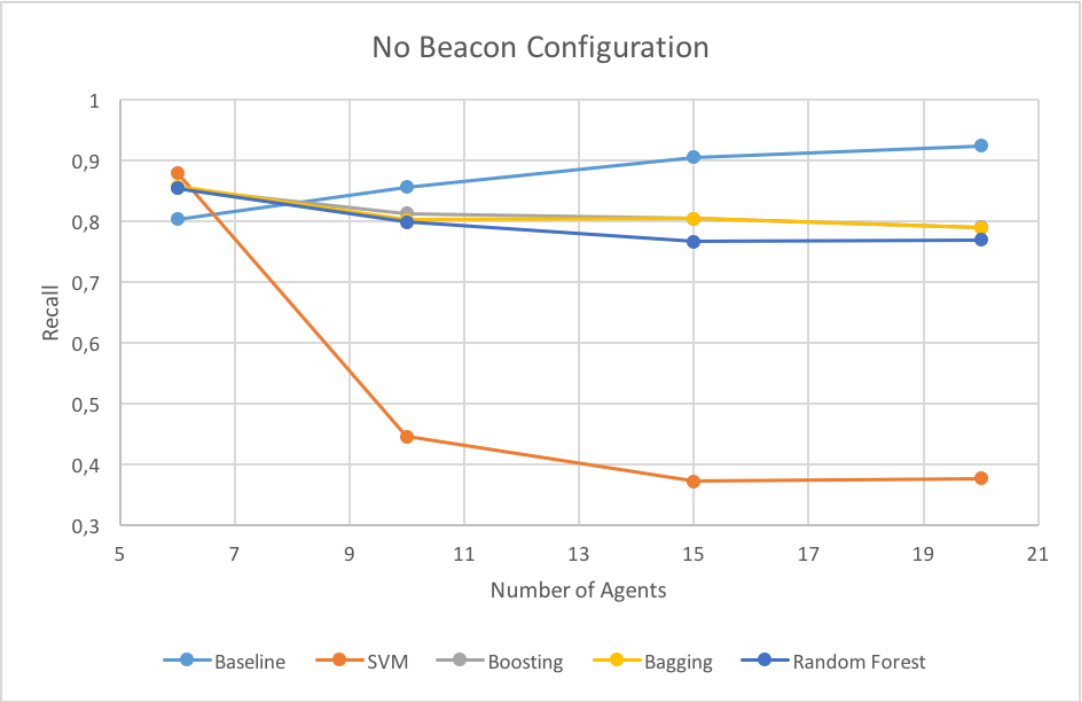

Figure 27: Recall evaluation of our selected models with different numbers of agents and with no beacons.

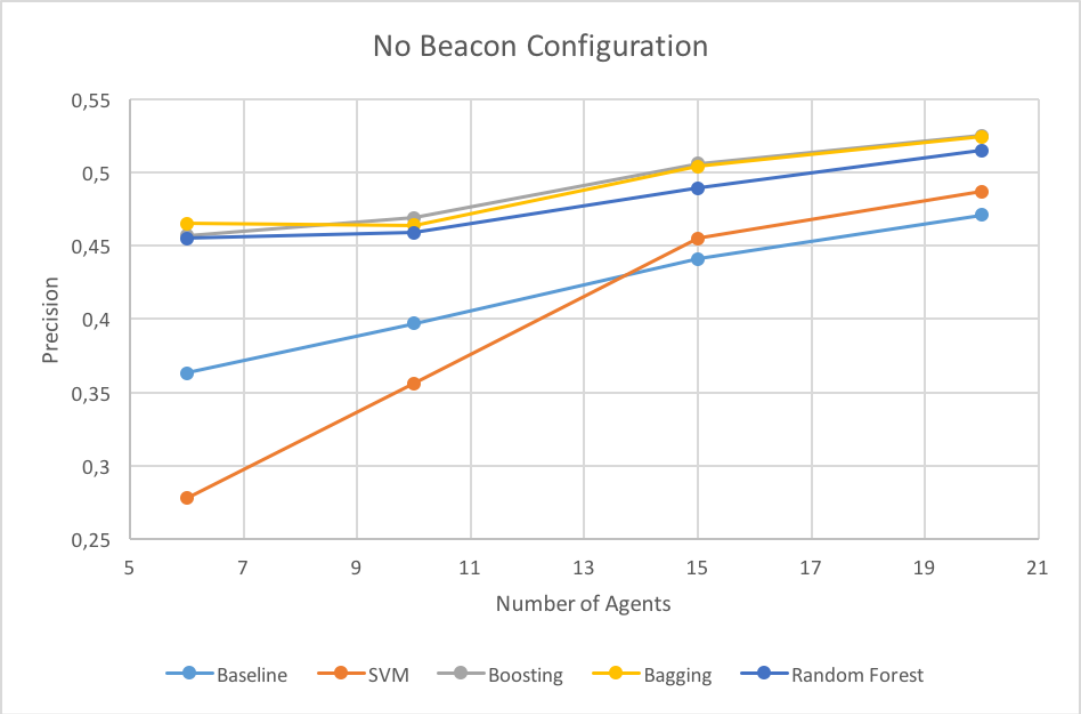

Figure 28: Precision evaluation of our selected models with different numbers of agents and with no beacons.

5.6 Decreasing Interactions

Probably the most critical parameter for our models to work is the likelihood of interactions between two close animals.

As a supervised learning approach which assumes differences among our animals, it generally needs a lot of data to work properly. When interactions become extremely rare, it starts to be a very big problem for our model to be trained properly.

We are going to decrease our interaction likelihood and try to understand when our proposed models become unfit for the task, and different approaches should be considered.

Up to now, we have always used a 20% chance of interaction between two close animals. We will start with 50% and decrease it significantly, with a training size of 2000 timestamps, a cost matrix which considers false negatives three times more important than false positives and only using absolute distances and speeds, as before.

Also, comparing results with our cost function doesn't make sense here again, because decreasing positives modifies the expected result of an evaluation. Thus, we will analyze recall and precision only.

Figure 29 shows how the baseline quickly drops down to not ever predicting a positive result, while our models appear to hold a very high recall up to 5% interactions, to drastically drop their performance after.

Figure 30, however, shows how precision drops all the time we decrease the likelihood of interactions and even more when we reach 1%.

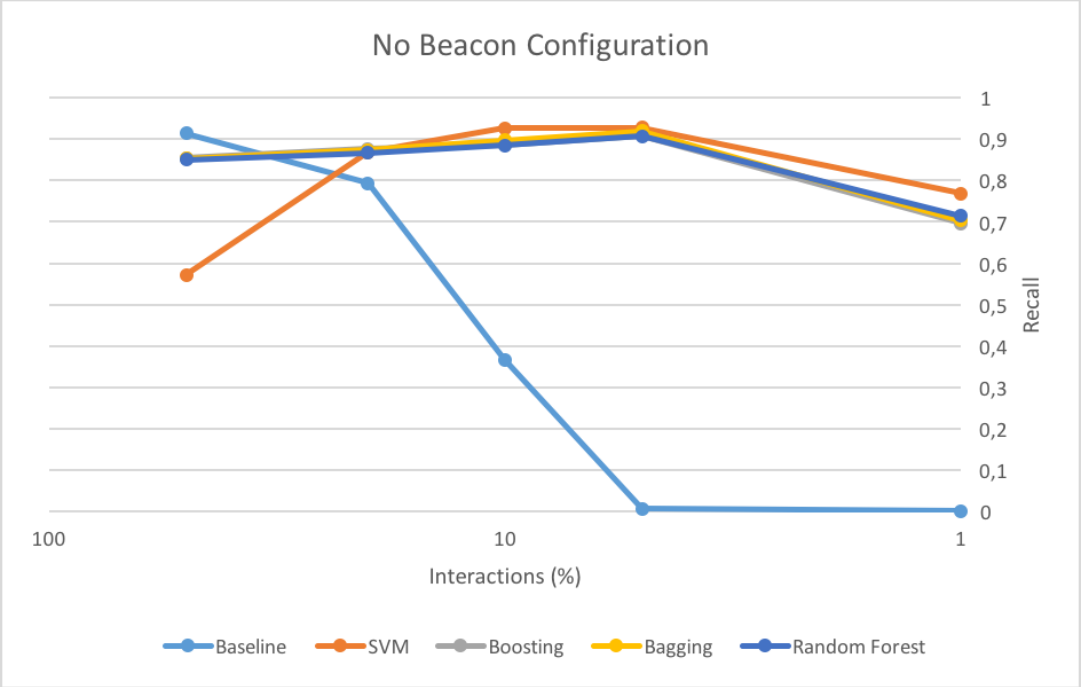

Figure 29: Recall evaluation of our selected models with different likelihood of interactions and with no beacons.

This was again predictable, because if the agents stay in the same small environment but interact less, its is extremely likely that just by chance more of their random movements look like an interaction, but it isn't. If we had a different type of interaction, in a bigger area, where being close to each other significantly increases the likelihood of an interaction, or with more specific accelerometer and orientation data, we expect our model to perform much better.

Still, we asked what would happen if we had more data. First of all, we modified the cost matrix to consider false negatives ten times more than false positives, then we significantly increased the training size to see if our models improved and by how much.

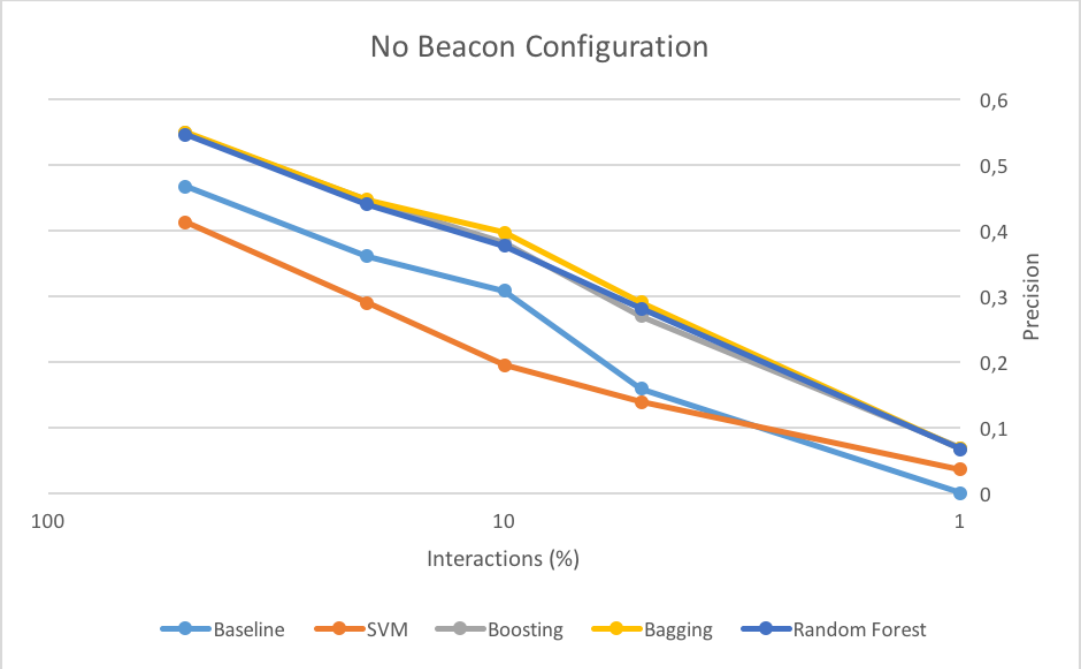

Figure 30: Precision evaluation of our selected models with different likelihood of interactions and with no beacons.

Figure 31 shows how recall increases until it stabilizes at 96% for all models but the baseline, which remains unable to predict a positive interaction. Figure 32 shows how precision grows *very slowly*, becoming excessively slow after 50000 samples.

This suggests us that for spotting this specific kinds of interactions, either this model might be too complex and we would need too much data, thus we should go with pairwise independent models, or it is too difficult to understand when an interaction does not occur without dropping the more important recall. Future work might want to question this idea by using simpler models.

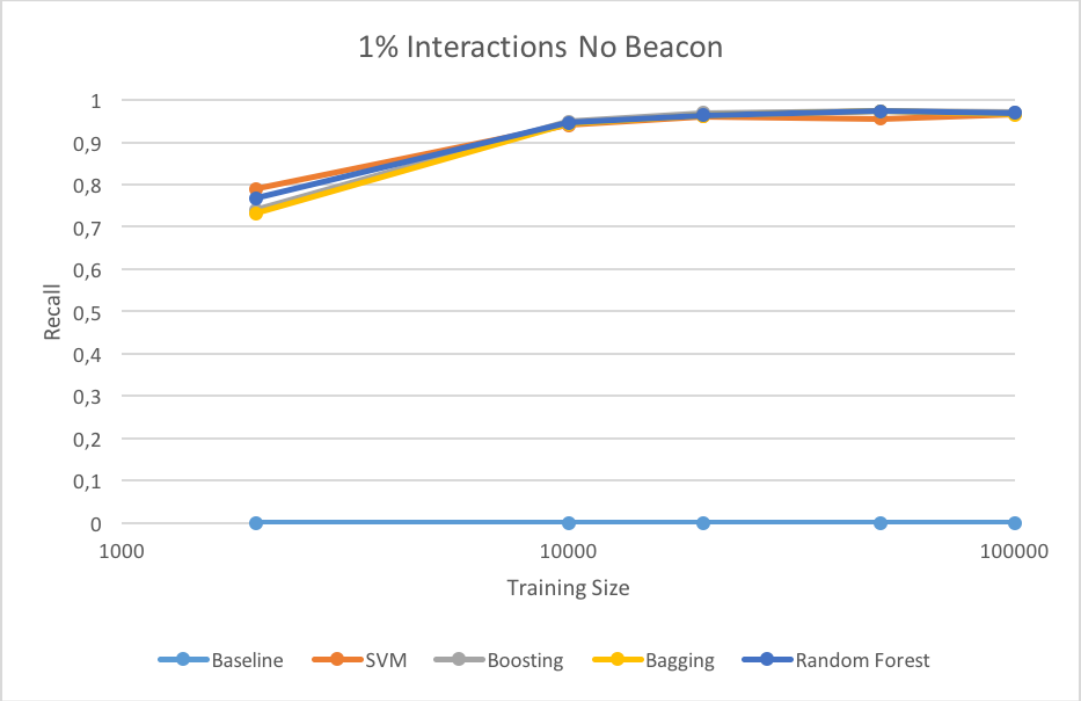

Figure 31: Recall evaluation of our selected models with different training size, 1% likelihood of interactions and no beacons.

Finally, we tried all these configurations with beacons too, but the results are conceptually identical, thus we avoided to insert them here.

5.7 Case Study Conclusions

We performed sensitivity analysis on our case study with FSM agents in a virtual fence with and without fixed position receivers.

The more training data we have, the better we classify. Reliability on transmitters is not critical for practical configurations, but their error is. Receivers error is sustainable in small

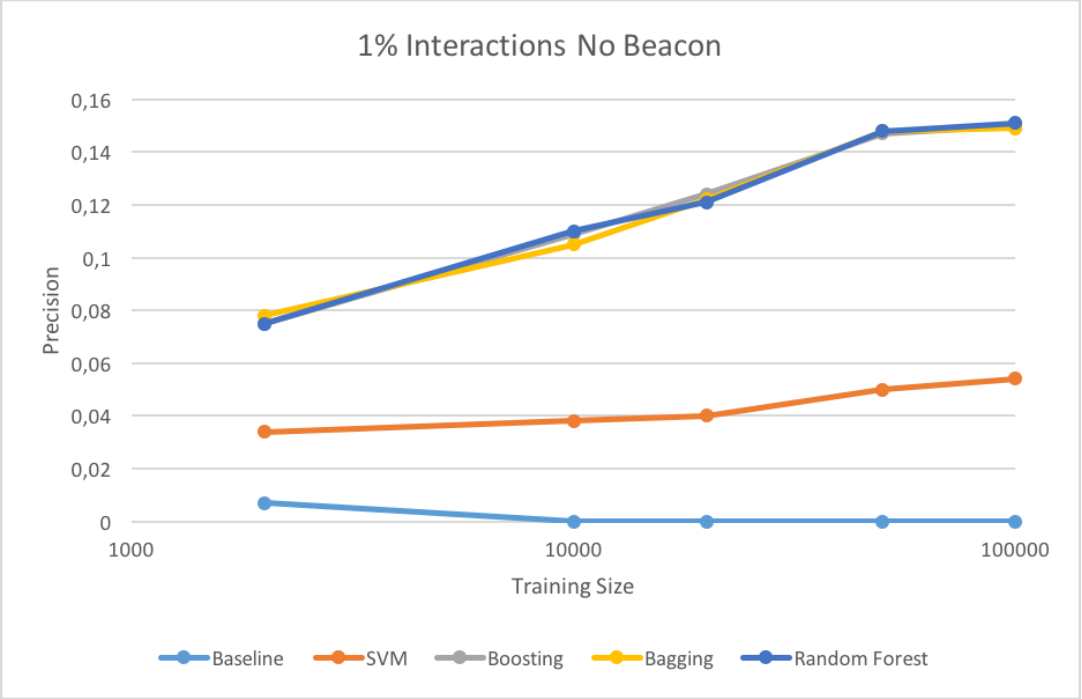

Figure 32: Precision evaluation of our selected models with different training size, 1% likelihood of interactions and no beacons.

amounts and in our configurations having fixed position beacons doesn't seem to be necessary, even with extremely noisy data.

With the types of interactions we created, ranking distances perform worse than absolute distances, but they are still very useful by themselves. However, we believe them being worse than absolute ones might be caused by our bias at creating interactions in that specific way, so it might be that more realistic agents or human labeling might turn this conclusion upside down.

Increasing the number of agents improved our performances, probably by making our predictions easier, but significantly increased the computational time needed to perform our tests.

Finally, decreasing the likelihood of agents interactions extremely impacted our results, dropping the precision by an exponential amount. This might be caused by either some inherent factors due to our synthesized model, or because of the extremely big amount of data required to make good predictions. We believe that once reached a 1% chance or less of interaction for two close animals, it is worthwhile starting to make some assumptions and try with some simpler methods which require much less data, to see if their performance outperforms our more needy model.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

We introduced a new way of using proximity data extracted from RSSI values to infer pairwise interactions of animals. We created different models for a supervised learning, assumptions free environment. We created a framework to perform extensive sensitivity analysis for calibrating any sensor system for inferring biological interactions which gives us the best model to use given any type of sensor data.

Our models work with arbitrary sensor data (GPS data can be used as well, of course it is still recommended to give distances and not positions as inputs) and can use context variables such as time of the day, season, specific places and so on.

Our framework for sensitivity analysis can be used to give guidelines for deploying sensors for a real configuration as well as suggesting the amount of training data required, and can give protocols for models and parameters selections.

Actual suggestions we were able to give to biologists are: we showed evidence supporting the thesis that ranking distances are useful and usable, that BLE sensors can be used in the field, that fixed position beacons are not necessary when we are interested in dyadic interactions, at least when we can have pairwise distances by applying them to animals, that accelerometer data is vital to spot specific types of interactions. We also discussed about when such models can be used and for what types of interactions. Finally, we gave some guidelines to how much

data we would need to have labeled for our models to work, with respect to the likelihood of interactions.

Future work can be done in many areas: we definitely need to try our models with human labeled data to see how it performs with real configurations.

We can add information of orientation of animals, to be able to give an idea if an animal is looking at another kin or not, which might significantly increase the accuracy of most human labeled and better synthesized models.

We can exploit Active Learning [33] to manually aid our model where it doesn't perform well, in our case, to try to increase the precision, trying to understand when it isn't able to spot a false positive and focus on labelling the types of interactions that are classified with the most errors, effectively decreasing the amount of data to label.

Comparing the performance of our model with respect to a simpler one which assumes independence of individuality and of interactions is extremely important, therefore we expect to build a new family of models and compare it with the ones we discussed here in a future extension of this work [38].

Finally, we might also be interested in knowing the absolute positions of our animals given our sensor, to see if it is possible to accurately triangulate their positions in a closed environment.

CITED LITERATURE

1. Darwin, C.: The Origin of Species. P. F. Collier & Son, 1909.

2. Watson, J. D. and Crick, F. H.: Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. Nature, 171:737–738, 1953.

3. Ballerini, M., Cabibbo, N., Candelier, R., Cavagna, A., Cisbani, E., Giardina, I., Lecomte, V., Orlandi, A., Parisi, G., Procaccini, A., Viale, M., and Zdravkovic, V.: Interaction ruling animal collective behavior depends on topological rather than metric distance: Evidence from a field study. Proceedings of the National Academy of Sciences, 105(4):1232–1237, 2008.

4. Altmann, J.: Observational study of behavior: Sampling methods. Behaviour, 49(3/4):227–267, 1974.

5. Silk, J. B., Alberts, S. C., and Altmann, J.: Social bonds of female baboons enhance infant survival. Science, 302(5648):1231–1234, 2003.

6. Blonder, B. and Dornhaus, A.: Time-ordered networks reveal limitations to information flow in ant colonies. PLoS ONE, 6(5):1–8, 05 2011.

7. Campbell, J., Mummert, L., and Sukthankar, R.: Video monitoring of honey bee colonies at the hive entrance. visual observation & analysis of animal & insect behavior, 2008.

8. Katz, Y., Tunstrm, K., Ioannou, C. C., Huepe, C., and Couzin, I. D.: Inferring the structure and dynamics of interactions in schooling fish. Proceedings of the National Academy of Sciences, 108(46):18720–18725, 2011.

9. Liu, J., Liu, J., Reich, J., Cheung, P., and Zhao, F.: Distributed group management for track initiation and maintenance in target localization applications. In Proceedings of the 2Nd International Conference on Information Processing in Sensor Networks, IPSN'03, pages 113–128, Berlin, Heidelberg, 2003. Springer-Verlag.

CITED LITERATURE (continued)

10. Shin, J., Guibas, L. J., and Zhao, F.: Information Processing in Sensor Networks: Second International Workshop, IPSN 2003, Palo Alto, CA, USA, April 22–23, 2003 Proceedings, chapter A Distributed Algorithm for Managing Multi-target Identities in Wireless Ad-hoc Sensor Networks, pages 223–238. Berlin, Heidelberg, Springer Berlin Heidelberg, 2003.

11. Zhao, F., Liu, J., Liu, J., Guibas, L., and Reich, J.: Collaborative signal and information processing: an information-directed approach. Proceedings of the IEEE, 91(8):1199–1209, Aug 2003.

12. Balch, T., Dellaert, F., Feldman, A., Guillory, A., Isbell, C. L., Khan, Z., Pratt, S. C., Stein, A. N., and Wilde, H.: How multirobot systems research will accelerate our understanding of social animal behavior. Proceedings of the IEEE, 94(7):1445–1463, July 2006.

13. Pereira, D. P., Dias, W. R. A., d. L. Braga, M., d. S. Barreto, R., Figueiredo, C. M. S., and Brilhante, V.: Model to integration of rfid into wireless sensor network for tracking and monitoring animals. In Computational Science and Engineering, 2008. CSE '08. 11th IEEE International Conference on, pages 125–131, July 2008.

14. Markham, A. C. and Wilkinson, A. J.: Novel Algorithms and Techniques In Telecommunications, Automation and Industrial Electronics, chapter EcoLocate: A Heterogeneous Wireless Network System for Wildlife Tracking, pages 293–298. Dordrecht, Springer Netherlands, 2008.

15. Juang, P., Oki, H., Wang, Y., Martonosi, M., Peh, L. S., and Rubenstein, D.: Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. SIGARCH Comput. Archit. News, 30(5):96–107, October 2002.

16. Webb, S. L., Gee, K. L., Strickland, B. K., Demarais, S., and DeYoung, R. W.: Measuring fine-scale white-tailed deer movements and environmental influences using gps collars. International Journal of Ecology, 2010, January 2010.

17. Hebblewhite, M. and Haydon, D. T.: Distinguishing technology from biology: a critical review of the use of gps telemetry data in ecology. Philosophical Transactions of the Royal Society of London B: Biological Sciences, 365(1550):2303–2312, 2010.

18. Ni, L. M., Liu, Y., Lau, Y. C., and Patil, A. P.: Landmarc: indoor location sensing using active rfid. In Pervasive Computing and Communications, 2003. (PerCom 2003).

CITED LITERATURE (continued)

Proceedings of the First IEEE International Conference on, pages 407–415, March 2003.

19. Priyantha, N. B., Chakraborty, A., and Balakrishnan, H.: The Cricket Location-Support System. In 6th ACM MOBICOM, Boston, MA, August 2000.

20. Strandburg-Peshkin, A., Farine, D. R., Couzin, I. D., and Crofoot, M. C.: Shared decision-making drives collective movement in wild baboons. Science, 348(6241):1358–1361, 2015.

21. Schwager, M., Anderson, D. M., Butler, Z., and Rus, D.: Robust classification of animal tracking data. Comput. Electron. Agric., 56(1):46–59, March 2007.

22. Handcock, R. N., Swain, D. L., Bishop-Hurley, G. J., Patison, K. P., Wark, T., Valencia, P., Corke, P., and ONeill, C. J.: Monitoring animal behaviour and environmental interactions using wireless sensor networks, gps collars and satellite remote sensing. Sensors, 9(5):3586, 2009.

23. Shemesh, Y., Sztainberg, Y., Forkosh, O., Shlapobersky, T., Chen, A., and Schneidman, E.: High-order social interactions in groups of mice. eLife, 2:e00759, sep 2013.

24. Long, J. A., Nelson, T. A., Webb, S. L., and Gee, K. L.: A critical examination of indices of dynamic interaction for wildlife telemetry studies. Journal of Animal Ecology, 83(5):1216–1233, 2014.

25. Farine, D. R. and Whitehead, H.: Constructing, conducting and interpreting animal social network analysis. Journal of Animal Ecology, 84(5):1144–1163, 2015.

26. Hong, W., Kennedy, A., Burgos-Artizzu, X. P., Zelikowsky, M., Navonne, S. G., Perona, P., and Anderson, D. J.: Automated measurement of mouse social behaviors using depth sensing, video tracking, and machine learning. Proceedings of the National Academy of Sciences, 112(38):E5351–E5360, 2015.

27. Glossary of terms. Mach. Learn., 30(2-3):271–274, February 1998.

28. Dietterich, T.: Overfitting and undercomputing in machine learning. ACM Comput. Surv., 27(3):326–327, September 1995.

29. Cortes, C. and Vapnik, V.: Support-vector networks. Machine Learning, 20(3):273–297, 1995.

CITED LITERATURE (continued)

30. Glorot, X., Bordes, A., and Bengio, Y.: Deep sparse rectifier neural networks. In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11), eds. G. J. Gordon and D. B. Dunson, volume 15, pages 315–323. Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011.

31. Breiman, L.: Bagging predictors. Machine Learning, 24(2):123–140, 1996.

32. Freund, Y. and Schapire, R. E.: A short introduction to boosting, 1999.

33. Ho, T. K.: Random decision forests. In Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1, ICDAR '95, pages 278–, Washington, DC, USA, 1995. IEEE Computer Society.

34. Gomez, C., Oller, J., and Paradells, J.: Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. Sensors, 12(9):11734, 2012.

35. Benkic, K., Malajner, M., Planinsic, P., and Cucej, Z.: Using rssi value for distance estimation in wireless sensor networks based on zigbee. In 2008 15th International Conference on Systems, Signals and Image Processing, pages 303–306, June 2008.

36. Macal, C. M. and North, M. J.: Tutorial on agent-based modeling and simulation. In Proceedings of the 37th Conference on Winter Simulation, WSC '05, pages 2–15. Winter Simulation Conference, 2005.

37. Damaceanu, R.-C.: Agent-based Computational Social Sciences Using NetLogo: Theory and Applications. Germany, LAP Lambert Academic Publishing, 2011.

38. Randazzo, E.: Inferenza di reti di interazioni diadiche da dati sensoriali. Tesi di Laurea Magistrale, Politecnico di Milano, 2016.

39. Settles, B.: Active learning literature survey. Technical report, 2010.

VITA

NAME: Ettore Randazzo

EDUCATION: Master of Science in Computer Science, University of Illinois at Chicago, May 2015, USA

Master Degree in Computer Science and Engineering, Jul 2016, Politecnico di Milano, Italy

Bachelor's Degree in Engineering Of Computing Systems, Jul 2014, Politecnico di Milano, Italy