HAL
open science

Human-in-the-Loop Feature Selection

Alvaro H C Correia, Freddy Lecue

To cite this version:

Alvaro H C Correia, Freddy Lecue. Human-in-the-Loop Feature Selection. AAAI 2019 Conference - 33th Association for the Advancement of Artificial Intelligence, Jan 2019, Honolulu, United States. hal-01934916

HAL Id: hal-01934916

https://inria.hal.science/hal-01934916v1

Submitted on 28 Nov 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Human-in-the-Loop Feature Selection

Alvaro H. C. Correia*
ENSTA ParisTech, Palaiseau, France
Utrecht University, The Netherlands
a.h.chaimcorreia@uu.nl

Freddy Lecue*
CortAIx Thales, Montreal, Canada
Inria, Sophia Antipolis, France
freddy.lecue@inria.fr

Abstract

Feature selection is a crucial step in the conception of Machine Learning models, which is often performed via data-driven approaches that overlook the possibility of tapping into the human decision-making of the model's designers and users. We present a *human-in-the-loop* framework that interacts with domain experts by collecting their feedback regarding the variables (of few samples) they evaluate as the most relevant for the task at hand. Such information can be modeled via Reinforcement Learning to derive a per-example feature selection method that tries to minimize the model's loss function by focusing on the most pertinent variables from a human perspective. We report results on a proof-of-concept image classification dataset and on a real-world risk classification task in which the model successfully incorporated feedback from experts to improve its accuracy.

Introduction

Selecting a subset of the available features for a given Machine Learning task, also known as *feature selection*, is a critical step that helps in the design of relevant, accurate and robust models (Guyon and Elisseeff 2003; Chandrashekar and Sahin 2014). Although ideally models should be capable of identifying the most predictive features during training, a large input space can reduce performance due to the "*curse of dimensionality*": the amount of data required to fit a reliable estimator increases exponentially with the number of variables. Therefore, a model based on a smaller number of features might produce better results while being faster and more cost-effective.

A way of tackling this problem is to select features according to experts' prior knowledge, but that is costly, time demanding and manual. Furthermore, often the users with the relevant understanding of the domain and task are not the ones designing the model. An alternative is to resort to methods that automatically rank and select features under some measure of relevance or importance. Even so, a limitation common to manual and automatic approaches alike is that they define a single feature subset to represent the entire dataset. When the amount of training data is small compared to the number of features, a single subset is unlikely to be able to describe all observations (Avdiyenko, Bertschinger, and Jost 2012).

*Work funded by Accenture Labs, Ireland for an internship.

Towards this challenge, we present a *per-example feature selection* method by including the *human-in-the-loop*. We propose a framework where experts are asked to identify the most relevant features for a few examples. Reinforcement Learning (RL) allows us to use that feedback to explicitly model the probability of selecting each feature and derive a policy that produces a new feature subset for each observation in the dataset. Each example is thus represented by a different set of features, which is the only input to a subsequent learning algorithm (e.g., a classifier or regressor). The feature selection policy is learned through policy gradient methods to minimize both (i) the loss function of the associated learning algorithm and (ii) the dissimilarity between the feature subsets chosen by the model and the users.

Such an approach can not only improve the performance of the model but also render its decision-making more interpretable to human users. As the final prediction relies only on the selected feature subset, which is individual to each example, one can interpret those variables as an explanation for the model's output. Moreover, by mimicking human annotation in selecting the most relevant features, the model can better reflect causal relationships in the experts' minds.

We frame our model as a stochastic computation graph (Schulman et al. 2015) and compare two different solvers: Score Function (SF) and Pathwise Derivative (PD) estimators. We validate our method in two scenarios: (i) a proof-of-concept image classification task and (ii) a real-world project risk classification task. In the latter, which is highly human-curated, our architecture improves the baseline accuracy by more than 50%. We also study the influence of different hyperparameters and factors of practical relevance, such as the required number of pieces of feedback and the presence of conflicting information from different experts.

The next section reviews related work. Then, we present (i) the feature selection problem, (ii) our human-in-the-loop approach, (iii) the stochastic computation graph framework with the SF and PD solvers, and finally (iv) the experiments in the two tasks described above.

Related Work

The literature is rich in feature selection methods, and we refer to (Guyon and Elisseeff 2003) and (Chandrashekar and Sahin 2014) for excellent surveys. More precisely, we are concerned with variable elimination techniques, where only

a subset of the available features is selected. These are traditionally divided in *filters* (Tyagi and Mishra 2013), *wrappers* (Kohavi and John 1997) and *embedded* methods (Guyon and Elisseeff 2003). *Filter methods* consist of a preprocessing step where the top-k features are selected by ranking them against some score function, such as the mutual information between input and target variables. Conversely, *wrapper methods* rank feature subsets following some performance measure such as the accuracy in the training data. That requires retraining the model for each candidate subset, which can be computationally costly and time-consuming. Finally, *embedded methods* try to solve that problem by performing feature selection while training the learning algorithm.

Our approach can be classified as an embedded method because we jointly train the feature selection and the learning algorithm via gradient descent. However, it distinguishes itself from conventional ones because it selects a different subset of the features for each observation. Such a per-example approach can better describe the data (Avdiyenko, Bertschinger, and Jost 2012) and render the output more interpretable, as the selected variables can be understood as the "causes" that drive the model's decision-making.

Per-example feature selection has also been studied in (Avdiyenko, Bertschinger, and Jost 2012). They proposed a filter method based on the mutual information between features and target variables, conditioned on the specific values of each example. The mutual information is then used as a score to rank the k most relevant features for each input. Their work contrasts with ours in two significant ways: (i) their method is deterministic and completely data-driven, whereas ours include human feedback to provide insights that cannot be automatically inferred from small or complex datasets; (ii) they build a subset sequentially, while our method produces a candidate subset in a single step, with no need for an arbitrary stop criterion.

Our model is also inspired by recent developments in *attention mechanisms* in deep learning (Mnih et al. 2014; Xu et al. 2015; Bengio, Léonard, and Courville 2013; Bahdanau, Cho, and Bengio 2015). By attributing weights to the different hidden states in a neural network, attention mechanisms are also selecting the most relevant features to minimize the model's cost function. Our contribution lies in the application of such attention mechanisms to human-in-the-loop architectures as a way to provide prior knowledge to the model on-the-fly.

Finally, our model also relates to probabilistic approaches to knowledge elicitation (Cano, Masegosa, and Moral 2011; House, Leman, and Han 2015; Daee et al. 2017). The work by Daee et al. is particularly relevant because they also query users about feature importance. However, they only model global feature relevance, whereas we propose a per-example feature selection method capable of eliciting the variables that drive the prediction for each observation.

Feature Selection

We assume a supervised learning scenario, where a parametric model is trained to minimize some cost function $\mathcal{C}(x, y, \theta)$, where x is the input, y is the target, and θ is the set of parameters defining the model. For clarity, we refer to the model's prediction as $\hat{y}$ to contrast with the true value y.

Framework

We propose a framework where the annotators or the users not only provide the ground truth y but also select the most relevant features for some of the examples in the dataset. That is common practice when experts emphasize some characteristics of the data to drive the machine learning modeling (Raghavan, Madani, and Jones 2006). To model that annotation and perform feature selection (variable elimination) on a per-example basis, we introduce a mask a that is applied over the input vector x to filter out irrelevant features in each observation. Therefore, if an example is given by $x \in \mathcal{R}^d$, where each dimension corresponds to a feature x_j, then a is an indicator variable in $\{0, 1\}^d$, such that:

$$\begin{cases} a_j = 1, \text{ if } x_j \text{ is used for example } x \\ a_j = 0, \text{ otherwise.} \end{cases} \tag{1}$$

We denote the input to the learning algorithm by x', which is now given by $x' = x \odot a$, where $\odot$ is the element-wise or Hadamard product. As the feature selection is performed on a per-example basis, a must be a function of the input x. However, if a is an indicator variable, we cannot apply gradient descent to minimize the loss $C(x', y, \theta)$ because it is no longer differentiable with respect to the function that defines a. For that reason, we make variable a stochastic and model the probability of a conditioned on x. Given that $a \in \{0, 1\}^d$, we can associate each component a_j to a Bernoulli distribution parametrized by $\hat{q}_j$ and obtain:

$$\pi(a|x) = \prod_{j=1}^{d} \hat{q}_j^{a_j} (1 - \hat{q}_j)^{(1-a_j)}. \tag{2}$$

Our objective is to concentrate the mass of the distribution $\pi(a|x)$ on values of a that minimize the loss function $\mathcal{C}(x', y, \theta)$. To achieve that, we will resort to RL and, in particular, policy gradient methods (Sutton and Barto 2011). Hence, we will refer to $\pi(a|x)$ as a policy, which should tell us the best a given the current input x. Keeping the RL jargon, the function that defines such a policy will be called an agent. The probability $\hat{q}_j$ of each feature being selected by this agent will be a function of x and approximated by a neural network with parameters ψ, where the last layer consists of a sigmoid function to squash the results into a probability space. The vector composed of all d parameters $\hat{q}_j$ is estimated in a single step by this neural network.

$$\hat{q}_\psi = (\hat{q}_1, \hat{q}_2, ..., \hat{q}_d) = h(x; \psi). \tag{3}$$

Note that we defined a as an indicator variable to produce a variable elimination method: the features are either in the selected subset of variables or not. However, our framework can be easily extended to any number of states to represent the relative importance of each feature, which could also be seen as mid-ground between the soft and hard attention models proposed in (Bengio et al. 2016). In that case, assuming $\mathcal{K}$ possible states, the policy $\pi(a|x)$ can be rewritten in the following more general form:

$$\pi(a|x) = \prod_{j=1}^{d} \prod_{k=1}^{\mathcal{K}} \hat{q}_{jk}^{[a_j=k]}, \quad (4)$$

where $\hat{q}_{jk}$ is the probability of assigning state k to feature j and $[a_j = k]$ evaluates to 1 if $a_j = k$, 0 otherwise.

Human-Like Feature Selection

The framework described above can also be used to model the feature selection performed by humans. We will assume without loss of generality that the annotators' feedback can be translated into a vector $q \in [0,1]^d$, where each element q_j reflects the importance attributed to feature j. In that case, q is directly comparable to the model's output $\hat{q}$, which can also be interpreted as the relevance of each feature. The intuition is that if q_j or $\hat{q}_j$ is close to one, feature j is determinant to predict $\hat{y}$ and is negligible otherwise.

Therefore, to train the agent to mimic the feature selection done by the users, first we need to define a similarity measure between q and $\hat{q}$. The distance between q and $\hat{q}$ is also a cost function, which we will refer to as $\mathcal{C}_f(x, q, \psi)$ in (5) to distinguish it from $\mathcal{C}(x', y, \theta)$. The Euclidean distance is a straightforward option of a distance measure, which results

$$\mathcal{C}_f(x, q, \psi) = \mathbb{E}_x \, ||q_i - \hat{q}_i||^2 = \mathbb{E}_x \, ||q_i - h(x; \psi)||^2. \quad (5)$$

The Mean Squared Error (MSE) between q and $\hat{q}$, as defined above, is not the only possibility, and other dissimilarity functions might be pertinent depending on the application and type of feedback. For instance, an alternative is the cosine distance

$$C_f(x, q, \psi) = 1 - \frac{q\,\hat{q}}{||q||_2 \, ||\hat{q}||_2} = 1 - \frac{q\,h(x;\psi)}{||q||_2 \, ||h(x;\psi)||_2}. \quad (6)$$

With the cosine distance, we penalize the agent if the two vectors do not have the same orientation in space but ignore differences in magnitude.

Once the similarity measure is defined, we just have to add $C_f(x, q, \psi)$ to the final cost function to which we will refer as $\mathcal{C}$ in (7) to simplify the notation.

$$\mathcal{C} = \mathcal{C}(x, y, q, \theta, \psi) = \mathcal{C}(x', y, \theta) + \lambda \mathcal{C}_f(x, q, \psi). \quad (7)$$

That can be interpreted as the sum of two reward signals that encourage the agent to achieve a good performance at the machine learning task while mimicking human feature selection. In (7) λ is a hyperparameter that balances the trade-off between these two signals. Such sum of errors for different tasks is common practice in multi-task learning (Caruana 1997) Note that the framework does not require human feature selection data to be provided for all data. For examples with no available feedback, $C_f(x, q, \psi)$ is set to zero.

Stochastic Computation Graphs

The mask a is stochastic and before making a prediction, we first need to sample a from the policy defined in (2). Hence, we can frame the whole model, including the feature selection and the learning algorithm, as a stochastic computation graph (Schulman et al. 2015). That allows for a straightforward comparison of possible solvers and clear representations of the model as depicted in Figures 1 and 2.

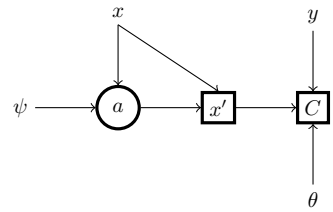


Figure 1: Stochastic graph for the score function estimator. As in (Schulman et al. 2015), square nodes are deterministic, whereas round ones are stochastic. Inputs and parameters are represented by their corresponding vector names.

The challenge in a stochastic computation graph is that the standard backpropagation algorithm is no longer sufficient because the cost $\mathcal{C}(x', y, \theta)$ is non-deterministic and non-differentiable with respect to the parameters ψ. In that case, we need some way of estimating $\nabla_\psi \, \mathbb{E}_a[\, C \,]$, the gradient of the expected loss function with respect to the policy parameters. To that end, we can resort to one of the following estimators to find the optimal policy $\pi(a|x)$: (i) the *Score Function* (SF) and (ii) the *Pathwise Derivative* (PD) estimators (Schulman et al. 2015).

Score Function Estimator

The Score Function estimator (SF), also known as the REINFORCE algorithm (Williams 1992), consists in rewriting $\nabla_\psi \, \mathbb{E}_a[\, \mathcal{C} \,]$ as $\mathbb{E}_a \left[\mathcal{C} \, \nabla_\psi log \left(\pi(a|x, \psi) \right) \right]$. Practically, it allows the complete model to be trained by gradient descent to minimize a surrogate cost function $\mathcal{C}'$ defined as follows:

$$\mathcal{C}' = \mathcal{C} \, log \left(\pi(a|x, \psi) \right) + \mathcal{C}. \quad (8)$$

In many cases, it is desirable to favor a subset with a smaller number of features, even if that does not decrease $\mathcal{C}'$. To encourage the desired sparsity in the mask a, we follow the approach used in (Bengio et al. 2016) and introduce a regularizer L_s, which penalizes the model if $\hat{q}_j$ is not sparse.

$$L_s = \mathbb{E} \left[(||\frac{1}{d} \sum_{j}^{d} \hat{q}_j) - \phi||_2 \right], \quad (9)$$

where $\phi \in [0, 1]$ is an hyperparameter that specifies the desired sparsity of activation. We also want $\hat{q}$ to have high variance across different examples to avoid always selecting the same subset of features. Hence, we add another regularizer to penalize low variance across examples in the same batch.

$$L_v = -\sum_{j}^{d} \frac{1}{m} \sum_{i}^{m} \left(\hat{q}_{ij} - \left(\frac{1}{m} \sum_{i}^{m} \hat{q}_{ij} \right) \right)^2, \quad (10)$$

where the index i refers to each example x and m is the batch size. The model is then trained to minimize $\mathcal{C}' + \lambda_s L_s + \lambda_v L_v$, where λ_s and λ_v are hyperparameters that balance the trade-off between the cost $\mathcal{C}'$ and the desired sparsity and variance in the feature subset.

Pathwise Derivative Estimator

The second alternative is to use the Pathwise Derivative estimator (PD), which is also known as the reparametrization trick and was made popular by variational autoencoders introduced in (Kingma and Welling 2014). We can sample from $\pi(a|x)$ by first sampling a latent variable z from a known fixed probability distribution $p(z)$ and transforming it using some function to recover a. In that case, the mask a is no longer sampled from a random variable because we introduce another stochastic node z to account for the randomness in the process. The corresponding graph for the pathwise derivative estimator is shown in Figure 2.

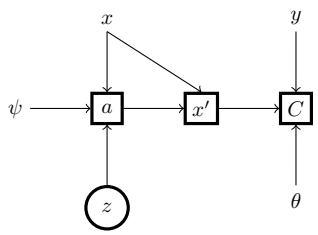


Figure 2: Stochastic graph for the pathwise derivative estimator. Here the mask a is no longer a stochastic node.

In variational autoencoders, z is usually distributed according to a Gaussian distribution (Kingma and Welling 2014; Kingma et al. 2016), but the reparametrization trick can be extended to categorical variables by sampling z from a Gumbel-softmax distribution (Jang, Gu, and Poole 2017; Maddison, Mnih, and Teh 2015). To use that distribution, we first need to rewrite a as a one-hot vector to match the output of a softmax. Assuming $\mathcal{K}$ different possible states, a_{jk} is the probability of assigning state k to feature j, which can be calculated as

$$a_{jk} = \frac{exp((log(\hat{q}_{jk}) + g_k)/\tau)}{\sum_{l=1}^{\mathcal{K}} exp((log(\hat{q}_{jl}) + g_l)/\tau)} \text{ for k in } 1, ..., \mathcal{K}, \tag{11}$$

where $g_0, g_1, ..., g_k$, are samples from Gumbel(0,1) (Gumbel 1954) and τ is a hyperparemeter that defines how close the distribution is from the argmax function. For $\tau > 0$, equation (11) is smooth and differentiable, but for $\tau = 0$ it is simply an argmax function. Hence, during training τ is maintained positive, but we gradually decrease its value as the model approaches convergence. During test time, we do not need the cost function to be fully differentiable and τ is set to zero to recover an argmax function.

Now the mask is a matrix $a \in \mathbb{R}^{d \times \mathcal{K}}$ and we need to reduce it to a vector of size d before multiplying it by the input x. One way to do so is to multiply a by a set of weights $w \in \mathbb{R}^{\mathcal{K}}$, so that $aw \in [0, 1]^d$. For instance, if we want to parametrize a Bernoulli distribution as in (2), a_j would have two states and w would be $[0, 1]^T$. Note that for $\tau > 0$, a is no longer an indicator variable, but is continuous in the interval $[0, 1]$. That is a relaxation that renders the graph wholly differentiable during training, but we regain the original definition of a in (1) during test time when $\tau = 0$.

The advantage of the pathwise derivative estimator is that it does not require a surrogate cost function and we can update the whole model via gradient descent by minimizing (7). We also observed that the PD estimator does not require the same type of regularization as the SF estimator, which reduces the number of hyperparameters that need fine-tuning. However, it does introduce another hyperparameter for the temperature τ, so that the entropy in the Gumbel-softmax can be regulated during training.

Experimental Results

We validate our feature selection method on: (i) a proof-of-concept image classification task for reproducibility and (ii) a real-world project risk classification task. All experiments were developed on top of the Tensorflow python API (GoogleResearch 2015) and run on a single GPU Nvidia GEFORCE GTX 1080 Ti. The code for the image classification task can be found at `github.com/AlCorreia/Human-in-the-loop-Feature-Selection`.

Set-up: The architecture of our feature selection model is similar for both use cases, which shows the generality of our approach. The classifier was a neural network with a feedforward architecture and a hidden layer of 256 neurons. The last activation was a softmax to yield the negative log-likelihood used as loss function for the classification task. The RL agent was also neural network-based with a single hidden layer of 128 neurons. In both cases, the activation function between the layers was a rectified linear unit (ReLU). For the project risk classification dataset, categorical features were embedded in a vector of size $\log_2 n$ where n is the total number of categories. These embeddings were learned via backpropagation with the rest of the model.

Runs: The model based on the PD estimator was trained for 100 epochs with batches of 876 projects (1000 images), but the feedback was only provided during the last 50 epochs. The SF estimator was trained the same way, but as it proved noisier, we also pretrained the network without any feature selection for 40 epochs. In each training phase, the whole model was updated via Adagrad (Duchi, Hazan, and Singer 2011) with initial learning rate of .5. The hyperparameter λ in (7) was set to 1, so that minimizing the cost function and reproducing the feedback were equally important objectives. For the SF estimator, the regularization parameters ϕ, λ_s and λ_v in (9) and (10) were set to .2, 1 and 1, respectively. For the PD estimator, different values for the temperature parameter were tested, but for the results in Table 4 and 5, it was set to 10 and decayed by 4% every 100 steps to slowly render the model more deterministic.

Image Classification Task

Context: We tested our model on an augmented MNIST dataset (LeCun et al. 1998) composed of cluttered images as in (Mnih et al. 2014). We randomly positioned each digit inside a larger frame and added four 8x8 sub patches of other digits to random locations of the image. For that task, the features x are not the raw pixels, but the output of a single convolutional layer (1x4x4x8), which discards the trivial solution of simply selecting non-zero pixels.

To prove that the agent can select relevant features, the attention vector was equally applied over all the filters of the convolution to make the mask directly related to locations in the image. Thus, one can interpret whether the agent's selection was pertinent by visually comparing it against the actual position of the digit.

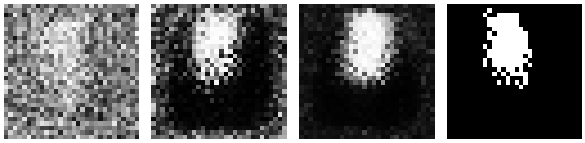
Feedback Computation: We simulated user feedback by fitting a 2D Gaussian with the position of the center of the digit for the mean. All training data was generated on-the-fly by repeating the above process. The test data was composed of a fixed set of 5000 images produced in the same way. Figure 3 shows an example of a cluttered image and its corresponding feedback.

(a) (b)

Figure 3: Cluttered MNIST image and simulated feedback.

Baseline: The baseline for the experiments is given by the neural network with the same architecture, but without any feature selection, i.e. all the features extracted by the convolutional layer are fed to the classifier. That model was able to classify correctly 92.23% of the examples in the test set after 100 epochs with batches of 1000 examples.

Selected Features: Figure 4 shows the evolution of the probabilities $\hat{q}$ corresponding to Figure 3, during the training of the model with the PD estimator. The model gradually learned to focus on a single region of the image, but $\hat{q}$ only became sharp after the feedback was introduced.



(a) Beginning (b) 25 epochs (c) 50 epochs (d) feedback

Figure 4: Evolution of the probabilities vector $\hat{q}$ during training of the model with PD estimator and MSE feedback.

Comparing Estimators

Table 1 shows the results obtained with both estimators when applying the proposed feature selection method on a cluttered MNIST dataset with a frame of 60x60.

Table 1: Estimators Impact on Accuracy (%).

Feedback / Estimator	SF	PD
Before Feedback	85.35	**85.70**
Cosine Feedback	**92.30**	88.40
MSE Feedback	91.16	89.61

Both estimators produced similar results during the first phase when the feature selection was based on the classification cost only. Despite similar performances, the PD estimator has the advantage of not requiring pretraining nor any regularization or variance reduction parameters.

In both cases the inclusion of the simulated feedback produced a considerable improvement in the accuracy of the model. However, the SF estimator proved more efficient in incorporating the human feature selection, outperforming the PD estimator by almost 5%. The difference can be attributed to the difficulty in fine-tuning the hyperparameter τ for the PD estimator based on the Gumbel-softmax. Even though the temperature was decayed during training, τ was probably still too high to allow for an efficient integration of the feedback. As indicated by the experiments presented in the next section, for τ around 1.0 the PD estimator produces similar results to those obtained with the SF here.

Impact of Temperature on Accuracy

The temperature τ is the only hyperparameter that needs to be fine-tuned when solving the graph with the PD estimator. We ran the model with the same architecture described above, but with five different values for the temperature. Differently from the previous experiments, in this case the value of τ was kept constant during the whole training.

Table 2: Temperature Impact on Accuracy (%).

Temperature Value	Before feedback	After feedback
0.25	84.77	88.70
0.50	85.48	89.76
1.0	**86.5**	**91.20**
3.0	84.10	88.90
10.0	84.77	88.82

When τ is close to zero, the Gumbel-softmax approaches an argmax over the probabilities $\hat{q}_j$, which means the agent only exploits and always picks the value for a that maximizes the reward under its current policy. For $\tau > 0$, a_j becomes stochastic, which allows the agent to explore and observe the loss for intermediate values of a. However, if τ is high, equation (11) is close to an uniform distribution over its $\mathcal{K}$ categories and a_{jk} is $\frac{1}{\mathcal{K}}$ for every k. Hence, assuming a fixed set of weights w, the final value for a_j would remain constant and independent of the policy parameters ψ, preventing the agent from updating its policy effectively. That is reflected in the results presented in Table 2. A temperature value of 1.0 produced the most accurate model, presumably striking a balance between exploration and exploitation.

From Bernoulli to Categorical Distribution

We studied the influence of the number of states $\mathcal{K}$ on the PD estimator. The results in Table 3 show that the accuracy of the model decreases with the number of states. One can infer two different reasons for that: (i) as the number of states increases, the search problem grows in complexity and (ii) given that the MNIST data is itself binary, the benefit of having multiple states does not compensate for the increase in complexity. However, in all cases the model benefited from

the feedback and achieved similar accuracy values to those presented Table 1. One can also conclude that the feedback is all the more valuable when the search space is large.

Table 3: Number of States on Accuracy (%).

States	Before Feedback	Cosine	MSE
2	**85.15**	88.40	89.50
3	83.82	**88.70**	**89.61**
10	80.83	88.69	88.98
20	74.24	86.62	85.40

Project Risk Classification Task

Business Context: We experimented our feature selection method on a Project Risk Classification (PRC) dataset. 4 years of project risk profiles of a leading service company, capturing $349,324$ projects across 97 features (19 categorical and 78 numerical) and classified among 5 categories (from high to no financial risk) are captured.

During the feedback collection phase, 114 business experts contributed by informing on: (i) (recommended) class, (ii) important features, (iii) textual comments (manually used to filter out some cases) cf. Figure 5. A total of $613,916$ pieces of feedback on $87,657$ active contracts (their classification and important features) have been collected over a 6-month pilot with an average of 7 pieces of feedback per project, and 45 per person/day. On average, the business experts provided feedback on 11 (standard deviation of 3.9) variables per example. Naturally, some feedback is conflicting among users: 21.6% (resp. 12.8%) of the feedback conflict at class (resp. feature importance) level.

Motivation: The PRC problem fits our framework as (i) baseline approaches reach a (low) maximum accuracy of 31.45% (random forest), (ii) project risk assessment is a highly human-curated task where experts can expose their knowledge by identifying relevant features, and (iii) completely data-driven approaches would require large amounts of data given the sparsity inherent of this context.

Feedback Computation: From now on, we will use the term feedback to refer to the feature-level annotation only, as we are mostly interested in feature selection. We limited the feedback to the binary impact of each feature because it facilitates the labeling task and better correlates with our variable elimination framework. Therefore, for each observation i, a piece of feedback translates to a binary vector $q_i \in \{0,1\}^d$, where $q_{i,j} = 1$ iff feature j is relevant for example i. Each example i received feedback from an average of 7 users. Consequently, for all experiments except those on Table 7, each i appeared multiple times in the dataset with different q_i's. All those observations were treated independently and randomly selected during training.

Validation: Accuracy is measured by comparing the predicted risk class with the real-world observations of risk for completed projects. Training, testing and validation sets correspond to $60/20/20\%$ of the projects, respectively, with each class equally distributed in the validation and test sets.

Baseline: There are two baselines for the experiments: (i) a random forest trained with the same features, which attained accuracy of 31.45% and (ii) a neural network with the same architecture described above (cf. Technical Context set-up), but without any feature selection. The latter classifies correctly 29.01% of the examples in the test set after 100 epochs with batches of 876 examples.

Selected Features: After training with the feedback, the model selected a relatively low number of features: 2 to 12 (mean: 4.6, std: 1.5). Such a low number of selected features is desirable as it favors the interpretability of the model.

Impact of Estimators on Accuracy

Table 4 reports the accuracy obtained with SF and PD estimators when applying our method on the PRC dataset.

Table 4: Accuracy (%) on PRC Test Set with Each Estimator.

Feedback / Estimator	SF	PD
Before Feedback	29.53	**29.99**
Cosine Feedback	**82.49**	77.51
MSE Feedback	80.11	78.44

The results show a very similar pattern to the one observed in the image classification dataset. Once more the SF was superior by 5%, but the PD was slightly more accurate before the introduction of feedback.

Impact of Temperature on Accuracy

We also varied the temperature τ for the PD estimator on the PRC dataset. The results presented in Table 5 support the same conclusions we reached on the image classification task. A τ of 1.0 lead to a higher accuracy both before and after the introduction of feedback, which suggests that would be a good default τ value for the PD estimator.

Table 5: Temperatures on Accuracy on PRC (PD & MSE).

Temperature Value	Before feedback	After feedback
0.25	28.78	77.82
0.50	30.02	78.77
1.0	**31.15**	**81.87**
3.0	27.98	78.33
10.0	28.36	78.07

Feedback Impact on Accuracy

Qualitative Feedback Impact (1): Table 6 presents an analysis of the optimal moment to start injecting the feedback into the model solved with the SF estimator. Even though pre-training the network without any feature selection is beneficial, feedback should be provided no later than 40 epochs into the training phase to reach optimal results.

Table 6: Feedback epoch on Accuracy on PRC (SF & MSE).

Epoch(#)	Accuracy(%)	Epoch(#)	Accuracy(%)
10	63.54	50	78.54
20	71.66	60	71.45
30	77.88	70	60.87
40	**80.11**	80	52.10

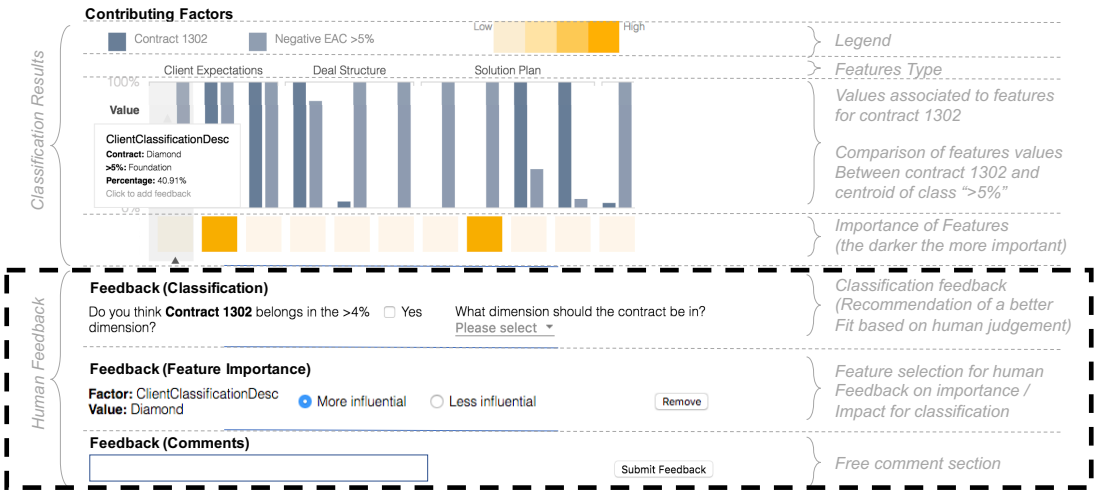


Figure 5: User interface for Project Risk Classification problem (human feedback collection inside dashed zone).

Qualitative Feedback Impact (2): We run the same tests with 4, 8, 16 and 32 states cf. Table 7 to evaluate the influence of the number of states $\mathcal{K}$ on the performance of the PD estimator. To match these different numbers of states, we combined the input from multiple users into a single score per example, $q_i \in \{0, \frac{1}{16}, ...\frac{15}{16}, 1\}$ (16 is the max. number of reviews per example). That score was adjusted proportionally to match the other versions with 2, 4, 8 and 32 states.

Table 7: Accuracy (%) on PRC Test Set with PD Estimator on Different Number of States.

States	Before Feedback	Cosine	MSE
2	24.01	72.99	73.11
4	**29.53**	77.00	77.51
8	28.71	**79.21**	**79.44**
16	26.39	78.01	78.33
32	26.14	76.17	76.24

The accuracy of the model increases with the number of states up to $\mathcal{K} = 8$ (outperforming results in Table 4) and then decreases. Although the search problem grows in complexity with the number of states, that seems to be compensated by the more fine-grained feedback. However, for $\mathcal{K} > 8$ the model starts to underperform, which indicates that at that point the number of pieces of feedback available is no longer enough to offset the larger search space. These results contrast with the ones shown in Table 3, as the more complex PRC dataset benefits more from granular feedback.

Quantitative Feedback Impact (1): As 12.8% of the feedback is conflicting among users, we evaluated the impact of such conflicts in Table 8. We varied this ratio by removing any conflicting feedback up to the point of not having any conflicts cf. case 0. We only removed pieces of feedback which had no common basis, $q_{i,j} \neq q'_{i,j} \forall j$. Interestingly, the best result was obtained when including 6% of the conflicting feedback. Although counterintuitive, such feedback might relax the constraint on some features, freeing the model to focus on minimizing the classification error.

Table 8: Conf. Feedback on Accuracy on PRC (SF & MSE).

Number (#)	Conf. Feedback (%)	Accuracy (%)
864,393	12.8	80.11
607,776	9	84.56
405,184	6	**86.67**
202,592	3	78.89
0	0	69.87

Quantitative Feedback Impact (2): Table 9 reports the impact of the quantity of feedback, considering similar ratio of conflicting feedback as for Tables 4 and 5. The accuracy plateaus with 80% of feedback. Thus, an average of 5.6 comments per project is required to obtain 77.54% of accuracy, which above the goal set up by the project stakeholders.

Table 9: Feedback Size on Accuracy on PRC (SF & MSE).

Feedback Size(#)	Feedback Ratio(%)	Accuracy(%)
0	0	29.53
3,376,538	50	61.65
5,402,460	80	77.54
6,077,768	90	78.11
6,753,076	100	**80.11**

Lessons Learnt

We learned a few important characteristics of the proposed human-in-the-loop architecture:

1. We observed the SF estimator does not respond well to dropout (Srivastava et al. 2014). Conversely, the PD estimator seems to benefit from this type of regularization (we used a dropout rate of 0.8 for all PD experiments);
2. The architecture used to model the agent is independent of the estimator. One can change between them during training to take advantage of both methods;
3. The influence of the dissimilarity distance (MSE/cosine) was marginal here but might vary with the application.

Conclusion and Future Work

We address the problem of human-in-the-loop per-example feature selection as a stochastic computation graph. It is a general approach that can be applied to a variety of machine learning tasks with little modifications, as demonstrated by the very distinct datasets we tackled in this paper. Direct applications could be in the context of transfer learning (Chen et al. 2018). With the image classification dataset, we visually proved the model can identify the most relevant features of each example, even though in that simple task, that did not reflect in a gain in accuracy. With the PRC dataset, we showed that our model successfully employed real human feedback to produce a significant improvement in accuracy, while also providing business-driven insights to the users. Most importantly, this new architecture enables a symbiotic interaction with stakeholders as the feature selection not only can enhance the model performance but also inform the most relevant properties of each example to the users. Thus, this type of interaction might prove useful in further developments of explainable artificial intelligence.

We identify two main lines of research for future work. The first is to extend the architecture to the full active learning scenario, where the model asks the users about specific examples. The second is to model the dependence between the features explicitly via a prior distribution over the probabilities $\hat{q}$ or a more complex RL policy. We framed the model as an RL problem precisely to support extensions of this sort.

References

Avdiyenko, L.; Bertschinger, N.; and Jost, J. 2012. Adaptive Sequential Feature Selection for Pattern Classification. In *IJCCI International Joint Conference of Computational Intelligence*, 474 – 482.

Bahdanau, D.; Cho, K.; and Bengio, Y. 2015. Neural Machine Translation By Jointly Learning To Align and Translate. *International Conference on Learning Representations 2015*.

Bengio, E.; Bacon, P.-l.; Pineau, J.; Precup, D.; Networks, K. N.; and Computing, C. 2016. Conditional computation in neural networks for faster models. In *International Conference on Learning Representations, Workshop Track, ICLR 2016*.

Bengio, Y.; Léonard, N.; and Courville, A. 2013. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *arXiv:1308.3432* 1–12.

Cano, A.; Masegosa, A. R.; and Moral, S. 2011. A method for integrating expert knowledge when learning bayesian networks from data. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 41(5):1382–1394.

Caruana, R. 1997. Multitask learning. *Machine Learning* 28(1):41–75.

Chandrashekar, G., and Sahin, F. 2014. A survey on feature selection methods. *Computers and Electrical Engineering* 40(1):16–28.

Chen, J.; Lécué, F.; Pan, J. Z.; Horrocks, I.; and Chen, H. 2018. Knowledge-based transfer learning explanation. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018, Tempe, Arizona, 30 October - 2 November 2018.*, 349–358.

Daee, P.; Peltola, T.; Soare, M.; and Kaski, S. 2017. Knowledge elicitation via sequential probabilistic inference for high-dimensional prediction. *Machine Learning* 106(9):1599–1620.

Duchi, J.; Hazan, E.; and Singer, Y. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research* 12:2121–2159.

GoogleResearch. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems.

Gumbel, E. J. 1954. *Statistical theory of extreme values and some practical applications: a series of lectures*. US Govt. Print. Office.

Guyon, I., and Elisseeff, A. 2003. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research* 3:1157–1182.

House, L.; Leman, S.; and Han, C. 2015. Bayesian visual analytics: Bava. *Statistical Analysis and Data Mining: The ASA Data Science Journal* 8(1):1–13.

Jang, E.; Gu, S.; and Poole, B. 2017. Categorical Reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations*, 1–13.

Kingma, D. P., and Welling, M. 2014. Auto-Encoding Variational Bayes. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*.

Kingma, D. P.; Salimans, T.; Jozefowicz, R.; Chen, X.; Sutskever, I.; and Welling, M. 2016. Improving Variational Inference with Inverse Autoregressive Flow. In *Conference on Neural Information Processing Systems*, number Nips.

Kohavi, R., and John, G. H. 1997. Wrappers for feature subset selection. *Artificial Intelligence* 1-2(97):273–324.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE* 11(86):2278–2324.

Maddison, C. J.; Mnih, A.; and Teh, Y. W. 2015. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *Advances in Neural Information Processing Systems*, 3528—-3536.

Mnih, V.; Heess, N.; Graves, A.; and Kavukcuoglu, K. 2014. Recurrent Models of Visual Attention. In *Advances in neural information processing systems*, 2204–2212.

Raghavan, H.; Madani, O.; and Jones, R. 2006. Active learning with feedback on features and instances. *The Journal of Machine Learning* 7:1655–1686.

Schulman, J.; Heess, N.; Weber, T.; and Abbeel, P. 2015. Gradient Estimation Using Stochastic Computation Graphs. In *Advances in Neural Information Processing Systems*, 3528—-3536.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15:1929–1958.

Sutton, R. S., and Barto, A. G. 2011. *Reinforcement Learning : An Introduction*. MIT press Cambridge.

Tyagi, V., and Mishra, A. 2013. A Survey on Different Feature Selection Methods for Microarray Data Analysis. *International Journal of Computer Applications* 67(16):975–8887.

Williams, R. J. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. In *Machine learning*, number 8, 229–256.

Xu, K.; Courville, A.; Zemel, R. S.; and Bengio, Y. 2015. Show , Attend and Tell : Neural Image Caption Generation with Visual Attention. In *International Conference on Machine Learning*, 2048–2057.