Active Learning Lecture Notes

Philip J. H. Jørgensen, Jesper D. Nielsen, Kristoffer H. Madsen

Contents

1 Introduction **2**
1.1 Active Machine Learning 2

2 Preliminaries **3**
2.1 Information Theory 3

3 Fundamentals **5**
3.1 Scenarios 5
3.1.1 Pool-based Sampling 5
3.1.2 Stream-based Selective Sampling 6
3.1.3 Membership Query Synthesis Scenario 6
3.2 Query Strategies 7
3.2.1 Query-By-Committee 7
3.2.2 Uncertainty Sampling 11
3.2.3 Expected Impact 12
3.2.4 Variance Reduction 16
3.2.5 Density Weighting 19

1 Introduction

You are probably, without knowing it, already familiar with the concept of active learning. The core idea is that a learning agent can ask meaningful questions about the world which it acts within in order to learn what it does not already know. In fact, you do it all the time, e.g., when you ask for the name of someone you have just met or execute some experiment to see its outcome. By using active learning, the learning process should, in turn, be fast by only focusing on investigating important elements for the task at hand. Essentially, an active learning system contains two entities: a learner able to ask questions based on what it sees and its current understanding of its world and a teacher, often referred to as an oracle, able to answer those questions. This interaction is illustrated in fig. 1.

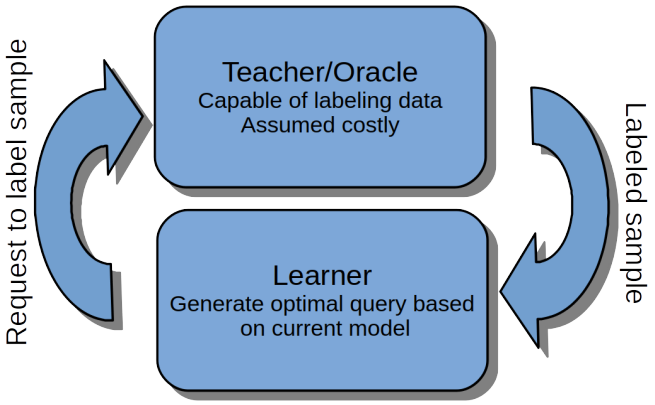

Figure 1: Schematic illustration of an active learning system.

1.1 Active Machine Learning

The more commonly taught setting of supervised learning where a learner is only trained on a "frozen" collection of data, i.e., no additional data will be collected, is known as *passive* learning, since the learner is incapable of influencing the exploration or sampling of the data space. Such data has been fully annotated with no regards to its usefulness to the learner. However, not all data is created equally in terms of how much new information each data point, after having observed and annotated it, provides to a learner. An extreme example of an uninformative data point is an exact duplicate of already seen data. If the data collection is a cheap, easy process, the overhead cost of annotating redundant data is small, but that is rarely the case for advanced systems such as, e.g., scientific experiments or images. Annotating the data might require expert knowledge or simply be very tedious to carry out, thus making it either expensive or highly time-consuming, or both. Instead, it is beneficial if we allow the learner to make decisions on what it should be taught, so the learning process becomes *active*, which is possible if the learner can quantify what it does not already know by computing how confident it is in its predictions on unlabeled data. Integrating the annotation of the data with the learning process is what active (machine) learning is all about.

Formally, for classification problems, we have a training set

$$D_t = \{(x_1, y_1), (x_2, y_2), \ldots, (x_t, y_t)\} \tag{1}$$

with x_i being the feature vector of the i'th observation with label y_i. A learning algorithm $\mathcal{L}$ has been fitted to D_t resulting in a score function $f_t : X \times Y \to \mathbb{R}$ which scores the possible label values for a given x. This score function is dependent on the currently available data indicated by its index t. Given some set of unlabeled data U, the active learning algorithm can ask the learner $\mathcal{L}$ which of the unlabeled observations in U it is the least confident about predicting a label for by computing the following expression

$$x^* = \underset{x \in U}{\operatorname{argmin}}\ c\left(\{f_t(x, y) \mid y \in Y\}\right) \tag{2}$$

where c quantifies the confidence of predicting the label of a given x using the scores given by f_t. The result x^* being the feature vector of U with the lowest confidence determined by the choice of confidence function c and the function score f_t. This is the feature vector which the active learner will query the oracle for a label, and sequentially add to the training set, resulting in the updated training set $D_{t+1} = \{D_t, (x^*, y_{t+1})\}$ and an updated score function f_{t+1} fitted to this set.

Passive supervised learning assumes that the collected data is sufficient to find a good score function given the static training set, whereas active learning will be suitable when data is difficult to collect and hence limited. The extra steps of evaluating the confidence of predicting the labels for new data and querying a teacher for labels are why active learning extends beyond the usual passive learning.

Active learning is highly motivated by the abundance or availability of unlabeled data. This could be an experiment where the conditions are easy to specify and set up but observing the outcome can take days or weeks. Another source could be the astronomical large pool of data available in form of text, images, and videos found online.

2 Preliminaries

In the following we introduce a few concepts from information theory which are important for understanding the metrics typically employed in query strategies. Here, just basic understanding of probability and statistics is assumed for now.

2.1 Information Theory

An important task of active learning is to evaluate how much information you gain from observing some event in order to decide what step to take for maximizing this gain, and to do so a measure of the information content of events are required. A natural measure of information gain is the *Shannon information content* of an event x from a random variable X, which is defined as

$$h(x) = \log \frac{1}{P(x)}. \tag{3}$$

Here, the base of the logarithm used will determine in what units the information is measured in, in case of base 2 the information will be measured in bits. $P(x)$ is the probability density function evaluated at the point $X = x$. Consequently, in the setting of a fair coin toss $P(X = \text{head}) = P(X = \text{tail}) = \frac{1}{2}$ the information gain for both possible outcomes would be one bit of information.

This measure only says something about the information of the event x and not all of X. However, when we want to evaluate the gain of the posterior predictive distribution of a binary classifier or even a multi-class classifier, we need a measure of the whole probability distribution of X, i.e., the probability of each label assignment given the input data given as $P(y \mid x)$. Using the Shannon information content in this way lead us to the measure most commonly known as the *entropy* of X, but today also often called the expected information content or even uncertainty of X. This is defined as

$$H(X) = \sum_{x \in \mathcal{A}_X} P(x) \log \frac{1}{P(x)} = -\sum_{x \in \mathcal{A}_X} P(x) \log P(x). \tag{4}$$

Hence, in the simple setting of the fair coin toss the two options are equally likely and hence the entropy or expected information content is still one bit. However, in the case of a biased coin toss where $P(X = \text{head}) = 0.2$ we would be quite surprised to receive heads and therefore the information gain for this outcome is larger than one bit. On the other hand the information content for tails is lower than one bit (as typically we just confirm our expectation of tails). Importantly, in this biased setting the average (weighted according to the probability density function) information content is $H(X) = -0.2 \log_2(0.2) - 0.8 \log_2(0.8) \approx 0.72$bits.

In the case of P(x) = 0 the convention is to say that $0 \times \log 1/0 = 0$ as $\lim_{X \to 0^+} X \log 1/X = 0$. The entropy is maximized when X is uniform with the intuition that the uncertainty is largest for such a distribution. If one event x contains all the probability mass with $P(x) = 1$ then we have no uncertainty and $H(X) = 0$.

Another important task when we have to evaluate the effect of observing some new label might be to compare the effect of labeling two different observations. This can be done by conditioning the learner fitted to the training set on observing some new data X_{T+1} leading to a new distribution dependent on X_{T+1}. For this we can use a distance measure of the two distributions such as the *Kullback-Leibler divergence*, also referred to as the relative entropy, defined as

$$D_{\mathrm{KL}}(P \mid\mid Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}. \tag{5}$$

If $P = Q$ we have $D_{\mathrm{KL}}(P \mid\mid Q) = 0$ and otherwise positive values as the equation satisfies Gibbs' inequality. The equation is not symmetric meaning that interchanging P and Q does not guarantee an identical result.

We will use these definitions of entropy and relative entropy throughout the following chapters and see examples of how they are useful for active learning in several applications.

3 Fundamentals

An active learning approach is defined by two elements; the learning context and the choice of query strategy. The learning context is determined by how data is made available to us. The query strategy is dependent on the learning algorithm and the properties of the data. Here we dive into these two aspects starting with the learning context.

3.1 Scenarios

Like any learning problem the specific context in which the learner is applied is fundamental for the algorithm. Each context or scenario for active learning defines a different way the learner can access and hence evaluate new unlabeled data in order to determine which to query. A summary of the three most common scenarios is listed below.

Pool-based sampling determines a subset of a given pool (set) of unlabeled data to query labels for.

Stream-based selective sampling determines on a basis of evaluating a single data point produced by a data stream whether to query for a label for it or not.

Membership query synthesis where the learner is allowed to query a label for any point belonging to the input space by generating "synthetic" points instead of sampling from an underlying data distribution.

3.1.1 Pool-based Sampling

Often it is easy to collect a large pool of unlabeled data, e.g. scraping the web for text, images or videos. The pool-based sampling scenario assumes that a small set of labeled data is obtained along with the large pool of unlabeled data. A learner is trained on the labeled data which in turn is evaluated on the unlabeled data in order to determine which subset of the unlabeled pool should be queried and added to the training set (i.e. the labeled data). After each query the learner is updated to incorporate the new data before making new queries on the remaining unlabeled pool of data. The pool is often considered to be static, i.e. no new data is added to the pool, but this is most often not strictly necessary. The process is shown in fig. 2 and summarized in algorithm 1.

Algorithm 1 Pool-based active learning

Given a labeled set, L, an unlabeled pool, U, and query strategy ϕ
repeat
  $\theta \leftarrow \text{train}(L)$ ▷ learn a model using L
  $x^* \leftarrow \arg\max_{x \in U} \phi(x)$ ▷ query the most informative instance
  $L \leftarrow L \cup \{x^*, \text{label}(x^*)\}$ ▷ move x^* from U to L and get label
  $U \leftarrow U \setminus x^*$
until some stopping criterion

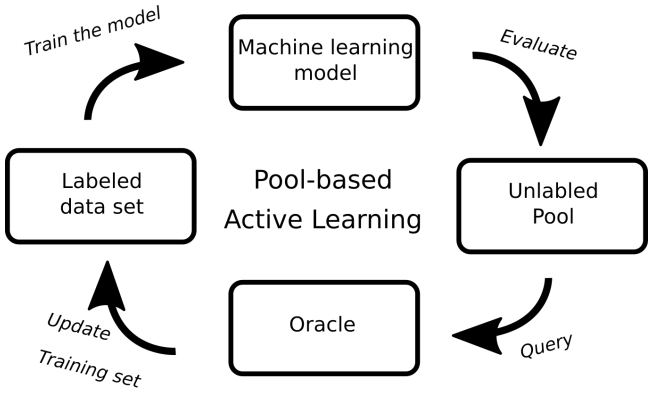

Figure 2: Diagram of pool-based sampling.

3.1.2 Stream-based Selective Sampling

When data is obtained sequentially through a data stream the learner no longer has a whole pool of unlabeled data to consider. Instead it has to determine whether or not each newly arrived data point should be labeled or not. If not, then the new data is simply discarded. If a query is made then the learner is again updated to incorporate the newly labeled data. This means that the decision of making a query in this scenario is based on the individual data points contrary to the pool-based sampling scenario considering the whole pool of unlabeled data for making the decision. The decision to make queries thus becomes harder since it can only be made by some threshold or relative to some statistic computed on the seen data. This is illustrated in fig. 3.

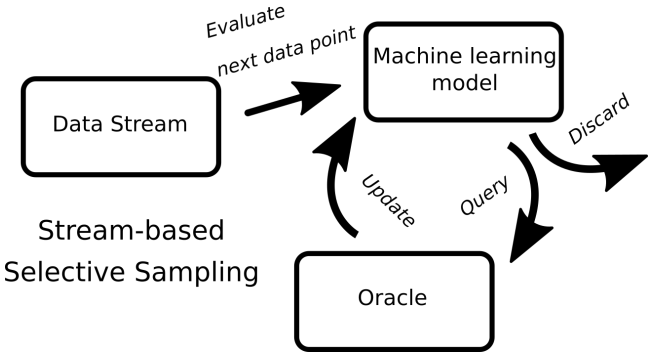

Figure 3: Diagram of stream-based selective sampling.

3.1.3 Membership Query Synthesis Scenario

In this scenario the learner does not evaluate unlabeled data sampled from an underlying distribution but can instead pick any data point being a part of its input space (i.e., "synthesize points"). This active learning scenario was one of the first to be investigated back in 1988 (Angluin, 1988). It is illustrated in fig. 4.

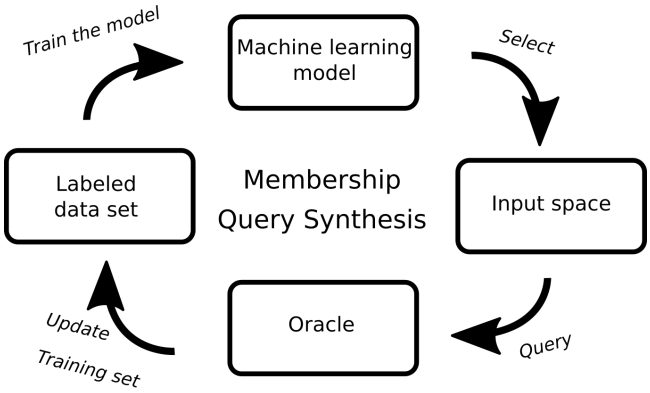

Figure 4: Diagram of the membership query synthesis scenario.

3.2 Query Strategies

A query strategy is the framework used to make decision on how to make queries for a given learner applied in a scenario. These query strategies can depend on the computational properties of the model and the properties of the data. Below is a list summarizing the most common strategies.

Query-By-Committee makes queries for the data points with the largest disagreement between a committee of models representing competing hypotheses.

Uncertainty sampling makes queries for the least certain data points as determined by the current model.

Expected Impact uses methods to estimate the expected model change or expected error reduction for each unlabeled data point in order to compute confidence.

Variance Reduction is a way to minimize the generalization error by minimizing the output variance of the learner.

Density-Weighted Methods suggests that the most informative unlabeled data points is not the most uncertain ones, which could be outliers, but those that are representative of the underlying distribution.

In this note, we denote query strategies by $\phi(x)$ or $\phi(x;\theta)$ depending on whether we need to be explicit about the model which is used or not. To be consistent with algorithm 1, we present these strategies in such a way that the objective is always to find the *maximizer*, i.e.,

$$x^* = \arg\max_{x \in U} \phi(x). \tag{6}$$

3.2.1 Query-By-Committee

The first query strategy we will be taking a closer look at is the *query-by-committee* (QBC) algorithm. This algorithm is one of the best theoretically

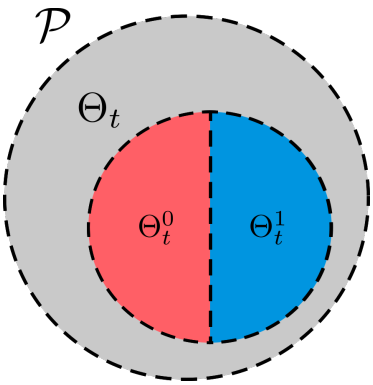

Figure 5: Illustration of the different spaces of parameters essential to the query by committee algorithm.

founded active learning algorithms. It is based on the idea that we can determine queries by how much labeling a given data point would reduce the *version space*. The version space is the set of all the hypotheses (or parameter values) given a class of learners that are consistent, i.e. agreeing, with the current training set. Since it is often unfeasible to integrate over the whole version space, it is approximated using a committee. Such a committee consists of a number of learners from the given model class each fitted to the training set, which can be used to determine the next query that would reduce the version space the most.

Consider the following example. I am thinking about a number in a certain interval and you have to guess which one (or come close enough). You can ask about numbers and I will tell you whether the one I am thinking of is higher or lower. In this case, the version space is the set of hypotheses (i.e., numbers) which are consistent with the information you have. Denote by θ a specific hypothesis (or threshold) separating high and low values and denote the version space after t questions as Θ_t, the latter of which is a subspace of $\mathcal{P}$, the set of all possible hypotheses. For this 1-dimensional problem we do not need a committee to evaluate the version space, but it should be clear to see that the center of the interval should be queried to maximize the reduction of Θ_t. Obtaining the label for the center value results in either the lower or upper half of the interval of the version space being removed from the version space, while the opposite half becomes the reduced version space. These spaces are illustrated in fig. 5.

Version Space Originally, the notion of version spaces was introduced by Mitchell (1982) where he cast the problem of supervised learning as a search problem. He shows that the problem of learning a generalization can be solved by searching through the set of hypotheses, i.e. parameter values for a class of models, that are consistent with the training data. A hypothesis is consistent with the training data only if all of the training data is correctly classified by the learner, since he makes the assumption that all the labels are correct. This assumption is still present in most active learning research today.

Denote the target model that we want to learn $f_*(x)$, which maps the input x to a label y, and a training set $D_t = \{(x_i, y_i)\}$ for $i \in 1, \ldots, t$ with $y_k \in \{0, 1\}$

and $x_k \in \mathbb{R}^d$. Then if we specify a class of parametric learning models $f_\theta(x)$ with parameters denoted θ, the version space is defined as

$$\Theta_t = \{\theta : f_\theta(x) = f_*(x), \forall x \in D_t\}. \tag{7}$$

This is the set of parameters that correctly labels the training set D_t according to the underlying model f_*. An important assumption we will make is that f_θ is able to fully realize the true model f_*. This means that there exists some parameters θ^* such that $f_{\theta^*}(x) = f_*(x)$ for all x. For this to hold we need to know what an appropriate model class is for the given problem which is not necessarily the case but for now we assume to know this.

Having defined the version space, we need a way of reducing it meaningfully. We begin to understand how to do so by setting a uniform prior $p(\theta)$ on the probability of the parameters. Then its posterior distribution will be uniform on the version space and otherwise 0, which is written as

$$p(\theta|D_t) = \begin{cases} \mathcal{V}_t^{-1} & \theta \in \Theta^t \\ 0, & \text{otherwise,} \end{cases} \tag{8}$$

where $\mathcal{V}_t$ is the volume of Θ_t. The entropy of the posterior in eq. (8) reflects our uncertainty about the parameters given the training set. Since the posterior is uniform, we can express its entropy as

$$\begin{aligned} H_t(\theta) &= \sum_{\theta \in \Theta^t} p(\theta|D_t) \log \frac{1}{p(\theta|D_t)} && (9) \\ &= \sum_{\theta \in \Theta^t} \mathcal{V}_t^{-1} \log \mathcal{V}_t && (10) \\ &= \log \mathcal{V}_t \sum_{\theta \in \Theta^t} \mathcal{V}_t^{-1} && (11) \\ &= \log \mathcal{V}_t. && (12) \end{aligned}$$

Using this, we can define the information gained by adding a new point, (x_{t+1}, y_{t+1}), as the change in entropy, i.e.,

$$\begin{aligned} I_{t+1}^\theta &= H_t(\theta) - H_{t+1}(\theta) && (13) \\ &= \log \frac{\mathcal{V}_t}{\mathcal{V}_{t+1}}. && (14) \end{aligned}$$

This is the information gain when the label y_{t+1} is known, however, we want to compute the information gain given a new data point x_{t+1} before we obtain this label. Equation (13) depends on the full sequence D_{t+1} and it contains the unknown label y_{t+1}, but instead we want to compute the information gain for the conditional distribution $p(y_{t+1}|D_t, x_{t+1})$. This is the probability of the unknown label y_{t+1} given the labeled data D_t and new data point x_{t+1} and, since the posterior is uniform, this is simply the fraction of the version space consistent with a given label, i.e.,

$$p(y = i|D, x) = \frac{\mathcal{V}^i}{\mathcal{V}}. \tag{15}$$

The information gain can therefore be expressed as

$$I^y_{t+1} = -\sum_{y_{t+1}\in\{0,1\}} p(y_{t+1}|D_t, x_{t+1}) \log p(y_{t+1}|D_t, x_{t+1}) \tag{16}$$

$$= -\frac{\mathcal{V}^0_t}{\mathcal{V}_t} \log \frac{\mathcal{V}^0_t}{\mathcal{V}_t} - \frac{\mathcal{V}^1_t}{\mathcal{V}_t} \log \frac{\mathcal{V}^1_t}{\mathcal{V}_t} \tag{17}$$

where $\mathcal{V}^i_t$ is the volume of the part of the version space defined by

$$\Theta^i_t = \{\theta \in \Theta_t : f_\theta(x_{t+1}) = i\}, \text{ for } i \in \{0, 1\}. \tag{18}$$

These two subspaces contains the parameters in the version space leading to the predictions of the label of the new data x_{t+1} by the given model to be 0 or 1 respectively.

Now, if new a data point x_{t+1} is generated by a query algorithm, which we for this moment specify as a conditional probability $p(x_{t+1}|D_t)$, we can compute the expected information gain with respect to the new data points specified by the query algorithm. This expectation is

$$\mathbb{E}_{x_{t+1}}[I^y_{t+1}] = \mathbb{E}_{x_{t+1}}\left[-\frac{\mathcal{V}^1_t}{\mathcal{V}_t} \log \frac{\mathcal{V}^1_t}{\mathcal{V}_t} - \frac{\mathcal{V}^0_t}{\mathcal{V}_t} \log \frac{\mathcal{V}^0_t}{\mathcal{V}_t}\right] \tag{19}$$

The new data point x_{t+1} which maximizes eq. (19) divides the version space exactly in half so that $\mathcal{V}^1_t = \mathcal{V}^0_t$ and hence y_{t+1} follows an uniform distribution which has maximal entropy. This is equivalent to a bisection of the version space. The information gain for obtaining a new label for such a data point is equal to $I^y = 1$ bit.

A Committee of Learners Ideally, we would like to compute the full version space for the training data and identify some input, x_{t+1}, that somehow divides it optimally. This, however, is not feasible for most nontrivial problems with huge version spaces and input spaces and we have to resort to approximations. We can approximate the version space by fitting a certain number of models (collectively denoted as a committee) to the training data and identify the data point which optimally divides the version space. The sought data point is the one about whose label the committee members disagree the most and which data point is selected depends on how disagreement is measured. One way to measure disagreement is vote entropy which is defined as

$$\phi_{\text{ve}}(x) = -\sum_i \frac{V(y_i)}{C} \log \frac{V(y_i)}{C} \tag{20}$$

where C is the size of the committee and $V(y_i)$ is the number of votes for the ith class. $V(y_i)$ is estimated from the committee members' predictions such that each model casts a vote for its most probable prediction (Settles, 2009). Alternatively, KL divergence may be used

$$\phi_{\text{kl}}(x) = \frac{1}{C}\sum_{c=1}^{C} D\left(P_c \| P_C\right) \tag{21}$$

where

$$D\left(P_c \| P_C\right) = \sum_i P_c(y_i|x) \log \frac{P_c(y_i|x)}{P_C(y_i|x)} \tag{22}$$

$$P_C(y_i|x) = \frac{1}{C} \sum_{c=1}^{C} P_c(y_i|x). \tag{23}$$

Here $P_c(y_i|x)$ is the predictive distribution of member c and $P_C(y_i|x)$ is the average predictive distribution across the committee. The advantage of vote entropy is that it can be used even with purely discriminative models or poorly calibrated uncertainties since the votes are used to generate a predictive distribution over classes. On the other hand, KL divergence uses the full predictive distribution of each model thus requiring an estimate of the uncertainty of each class. One could also use eq. (23) directly and then apply one of the uncertainty measures described below.

3.2.2 Uncertainty Sampling

Whereas QBC estimates the most uncertain point as the point about which a number of models (the committee members) disagree the most, uncertainty sampling strategies define the most informative unlabeled data point based on the uncertainty of its predicted labels as computed by a *single* model. The most informative unlabeled data point is defined as the one which the model is most uncertain on how to label. Depending on the problem at hand, this is defined in different ways. The uncertainties of the labels are readily available if the model is probabilistic, however, non-probabilistic models can still be used if it is somehow possible to quantify their uncertainty.

Binary classification is the most simple problem for which to specify the most uncertain data point. If a model returns the posterior probability of a given data point being classified as a positive example, then the most uncertain data point is the one with the posterior probability being closest to 0.5. It becomes more involved when encountering a multi-class classification problem since the specification of uncertainty has to take several probabilities into account. A simple approach would be to discard all but the largest probability, that is find

$$y_{(1)} = \arg\max_{y \in Y} p_\theta(y|x) \tag{24}$$

such that $y_{(1)}$ is the most probable label and estimate the uncertainty of x as

$$\phi_{\text{lc}}(x) = 1 - p_\theta(y_{(1)}|x). \tag{25}$$

Thus, we are looking for the *least confident* data point: the one with the lowest probability for its most likely label (denoted $y_{(1)}$). Intuitively, this is the probability for the model to mislabel x and can be viewed as the expected value of the 0-1 loss function, i.e., $\mathbb{E}\left[L(\hat{y}, y)\right] = \sum_{y \in Y} p(y|x) I_{y \neq \hat{y}}$, where I is the indicator function.

By specifying the most uncertain data point as the least confident, we do not use any information given by the distribution over the remaining labels. Say, if the probability of the second most likely label belonging to some x_p contains most of the remainder of the probability mass, the learner might be

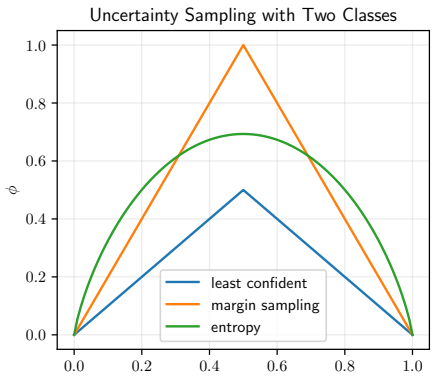

Figure 6: Uncertainty sampling strategies in a setting with two classes. All metrics have the same maximum.

more uncertain about its label than a x_q with a closer to uniform distribution for the remaining labels. To overcome this one could consider to use *margin sampling*, which is expressed as

$$\phi_{\mathrm{m}}(x) = 1 - \big(p_\theta(y_{(1)}|x) - p_\theta(y_{(2)}|x)\big) \tag{26}$$

where the notation $y_{(1)}$ and $y_{(2)}$ is borrowed from order statistics to denote the most and second most likely label of x, respectively. Here the data point with the smallest difference between the two most likely labels will be chosen for the next query. Margin sampling takes the probability of the second most likely label into account but still ignores all the remaining labels, so if the problem consists of a large set of labels Y much of the information about the label distribution is still not included in the query decision. For a small number of labels, margin sampling might perform well enough, but we can do better.

The final strategy for uncertainty sampling that we will be considering is the most general. It computes the entropy (given by eq. (4)) of the full label distribution given some x and uses it as the uncertainty measure for the query

$$\phi_{\mathrm{e}}(x) = -\sum_{y \in Y} p_\theta(y|x) \log p_\theta(y|x). \tag{27}$$

As described earlier, entropy quantifies the uncertainty of a random variable so it seems a sensible choice to use as an uncertainty sampling strategy. Also, since the full distribution of the predictions is considered, this strategy is straightforward to generalize to more structured data such as sequences or trees.

For binary classification problems, all of these uncertainty sampling strategies reduce to the strategy of choosing the data point with predictive probabilities closest to 0.5 and is illustrated in fig. 6. With three or more classes, this is not the case anymore and the strategies will not be equivalent as shown in fig. 7.

3.2.3 Expected Impact

Uncertainty sampling assumes that the most informative labels to learn are those of the most uncertain data points without considering the improvement

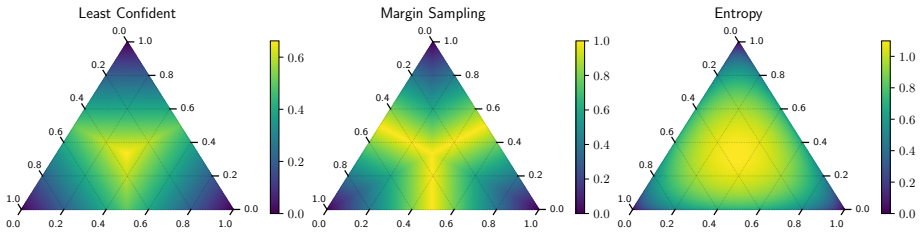

Figure 7: Uncertainty sampling strategies in a setting with three classes. In the corners of the simplex, the high probability of a particular class is high whereas it is low on the opposing edge. Adapted from Settles (2009).

in model performance. Consequently uncertainty sampling depends on a good model and that uncertain data points will be informative for improving the model. Instead of assuming that information follows uncertainty, we can compute an expected improvement of some objective for the model based on learning a new label. We will look at a number of ways to do this. In Bayesian optimization we already visited the concept of expected improvement when discussing acquisition functions. In that setting, we estimated the expectation value of the improvement in the objective function for each candidate point. In active learning the concept is similar, only here the definition of the objective function is less clear and we need to rely on other ways to estimate improvement.

Expected Model Change The first group of these strategies defines improvement as the amount of change in the model after learning a new label, i.e., *expected model change* strategies. Many learning algorithms are trained by optimizing an objective function using its gradients with respect to the parameters of the model. The magnitude of those gradients measures the change in the parameters of the model, and can be used to compute the *expected gradient length*

$$\phi_{\text{egl}}(x) = \sum_{y \in Y} P_\theta(y|x) \left\| \nabla \ell_\theta(D_t \cup \{x, y\}) \right\| \tag{28}$$

$$\approx \sum_{y \in Y} P_\theta(y|x) \left\| \nabla \ell_\theta(\{x, y\}) \right\| \tag{29}$$

where $\nabla \ell_\theta(\cdot)$ is the gradient of the objective function ℓ computed with respect to the model parameters θ and $\|\cdot\|$ is the Euclidean length of the given vector. Note that because the model was already optimized for the existing data points, D_t, the gradient of the loss should be zero (or at least close) for those given that the model has converged. Therefore, it is sufficient to evaluate the gradient for the newly considered data point. This query strategy selects the x with the largest expected gradient length with respect to the posterior predictive distribution of the labels.

Please note that scaling of features may affect the performance of this query strategy. For example, the informativeness of a given point may be overestimated simply because one of its feature values is (unusually) large or if the parameter value associated with a particular feature is large (Settles, 2009).

Expected Error Reduction Another way to define improvement is by how likely a reduction in the generalization error of our model is. For pool-based sampling one can assume that the unlabeled data, U, represents a test distribution and hence use it to estimate the expected future error (or risk) of the model. That is, even though we do not know the true labels of the data in U, we can still estimate the error we would expect based on the predictions of our current model. The degree to which we are able to estimate the expected reduction in error of course relies on our current model being reasonable accurate. Expected error reduction estimates take the form

$$\phi_{\text{eer}}(x) = -\sum_{y \in Y} P_\theta(y|x) \sum_{x' \in U'} \phi\left(x'; \theta'\right) \tag{30}$$

where $L' = L \cup \{x, y\}$, $U' = U \setminus \{x, y\}$, θ' is the model trained on L' (as opposed to θ which is trained on L), P_θ means probabilities from the model θ, and $\phi(x; \theta')$ indicates that ϕ is evaluated using the probabilities from θ'. ϕ denotes one of the uncertainty estimates from section 3.2.2. Consequently, using ϕ_{lc} corresponds to the expected 0-1 loss whereas using ϕ_{e} we get the expected log loss. Our objective is to identify the data point which we believe will result in the largest error *reduction*, however, to be consistent with the other sampling functions in this note, we have included a minus sign in eq. (30) such that the expression is to be maximized.

As an example, if we use entropy then the full expression becomes (i.e., plugging eq. (27) into eq. (30))

$$\phi_{\text{eer}}(x) = -\sum_{y \in Y} P_\theta(y|x) \sum_{x' \in U'} \phi_{\text{e}}(x'; \theta') \tag{31}$$

$$= \sum_{y \in Y} P_\theta(y|x) \sum_{x' \in U'} \sum_{y' \in Y} P_{\theta'}(y'|x') \log P_{\theta'}(y'|x'). \tag{32}$$

In words, we update the model by adding a new (hypothetical) data point, (x, y), estimate some uncertainty metric (e.g., ϕ_{e}) over the remaining pool (excluding the point we used to update the model), and weigh it by how likely we believe getting (x, y) is. We do this for all possible pairs (x, y) and choose the value of x which we expect will optimize the distribution of class probabilities wrt. the uncertainty metric we used. Equation (30) is summarized in algorithm 2.

Note that computing the loss in this way is likely going to be very computationally expensive as we have to fit a new model for each $x \in U$ using each label.

Algorithm 2 Expected Error Reduction

Given a labeled set, L, an unlabeled pool, U, the set of all possible labels, Y, a model, θ, trained on L, and $e^* = -\infty$
for $x \in U$ **do**
  $e_x \leftarrow 0$
  for $y \in Y$ **do**
    $L' \leftarrow L \cup \{x, y\}$
    $U' \leftarrow U \setminus \{x, y\}$
    $\theta' \leftarrow \text{train}(L')$
    $e_{xy} \leftarrow \sum_{x' \in U'} \phi(x'; \theta')$ ▷ loss over U' when adding $\{x, y\}$
    $e_x \leftarrow e_x - p_\theta(y|x) e_{xy}$ ▷ expected (negative) loss of adding x
  end for
  if $e_x > e^*$ **then**
    $x^* \leftarrow x$
    $e^* \leftarrow e_x$
  end if
end for
return x^*

3.2.4 Variance Reduction

Another approach to quantifying the information gain is based on the bias-variance decomposition of the squared loss. An example could be a linear regression problem with additive noise written as

$$y = f(x) + \epsilon, \tag{33}$$

where an estimate of f denoted $\hat{f}$ is determined by minimizing the expected squared loss. It is well known that the expected squared loss can be expanded into three parts

$$\mathbb{E}_T[(\hat{y} - y)^2|x] = \underbrace{\mathbb{E}[(y - \mathbb{E}[y|x])^2]}_{\text{noise}} + \underbrace{(\mathbb{E}_D[\hat{y}] - \mathbb{E}[y|x])^2}_{\text{bias}} + \underbrace{\mathbb{E}_D[(\hat{y} - \mathbb{E}_D[\hat{y}])^2]}_{\text{variance}} \tag{34}$$

where $\mathbb{E}_D[\cdot]$ denotes an expectation over data sets, $\mathbb{E}[\cdot]$ is an expectation wrt. the conditional density $p(y|x)$, and $\mathbb{E}_T[\cdot]$ is an expectation over both (Cohn et al., 1996). $\mathbb{E}[y|x]$ refers to the "true" value of the dependent variable whereas y is an actual observation (which includes noise) and $\hat{y}$ is an estimate of the output variable based on a particular data set, D.

The noise is the variance of the output given the input coming from limitations in the measurements. This term is independent of the model and training data, so it can not be minimized. The bias is the squared difference between the expected output of the model across training sets and the true model and is determined by how well the chosen model class of f approximates $\hat{f}$. The variance describes how much our model estimate changes from data set to data set and is determined by the complexity of the model since a very flexible model potentially leads to very different solutions for different (randomly sampled) training sets.

Instead of attempting to minimize the full generalization error, i.e. both the bias and variance, which often lead to expensive non-closed form solutions, we can focus on the variance term. One obvious way to reduce variance is simply to train the model using a larger number of observations since the expectation is taken over data sets. However, if this is not possible, we will have to choose our samples more wisely instead, i.e., choose samples which carry the most information about the parameters. As an example, consider fitting a line to a set of points as illustrated in fig. 8. Variance reduction techniques basically aim to sample where the error bars on the estimate is largest. In many cases, and for linear models in particular, such points will be those beyond the points we have already gathered[1]. However, simply sampling more and more "extreme" points may not be an ideal strategy in practice as we may not be interested in those regions of space (MacKay, 1992). Figure 9 shows how the likelihood function changes depending on which point is added to the data set. Since we are only fitting a single parameter in this example, it is easy to evaluate the variance, however, with two or more parameters, the variance takes the form of a covariance matrix and in this case it is less clear what exactly to optimize (Settles, 2009).

Cohn (1994) and Cohn et al. (1996) show how the variance term can be reduced for models such as neural networks, Gaussian mixture models, and

[1] In this sense, a linear model may not be ideal in terms of demonstrating the usefulness of the variance reduction principles, however, we feel it is still useful for illustrating the concepts.

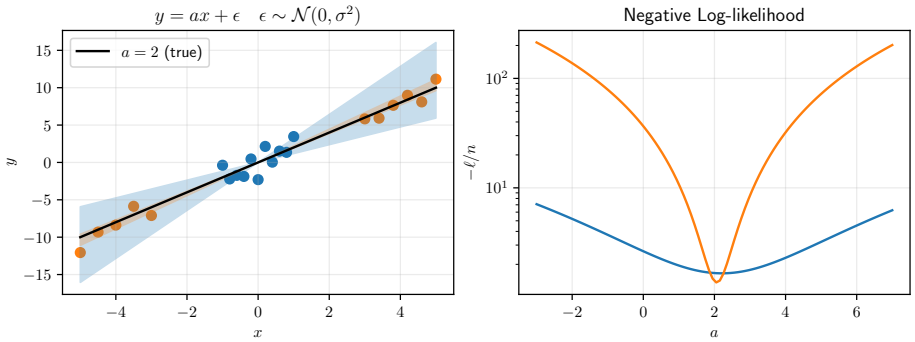

Figure 8: Correspondence between variance in input/output space and parameter/likelihood space. We fit the model $f(x) = ax$ and assume normally distributed residuals with equal variances (ordinary least squares) so the negative log-likelihood becomes $-\ell = \frac{n}{2}\log 2\pi + \frac{n}{2}\sigma^2 + \frac{1}{2\sigma^2}(y - ax)^2$. Here we used $\sigma = 1$. Left. Uncertainty of model estimated using bootstrapping of either the blue or the orange data points. It is clear that uncertainty is much higher when the spacing between the points is small as choosing more "extreme points" constrains the model parameters more. Shaded areas denote ± 2 standard deviations. Results are based on 100 simulations. Right. Average negative log-likelihood, $-\ell/n$, as a function of the model parameter. We see how the orange points generates a more peaked $-\ell$ since fewer models are compatible with the data. Notice that since we are fitting the model $f(x) = ax$ (slope only) we have an implicit constraint in $f(0) = 0$ which is why there is no uncertainty about this point.

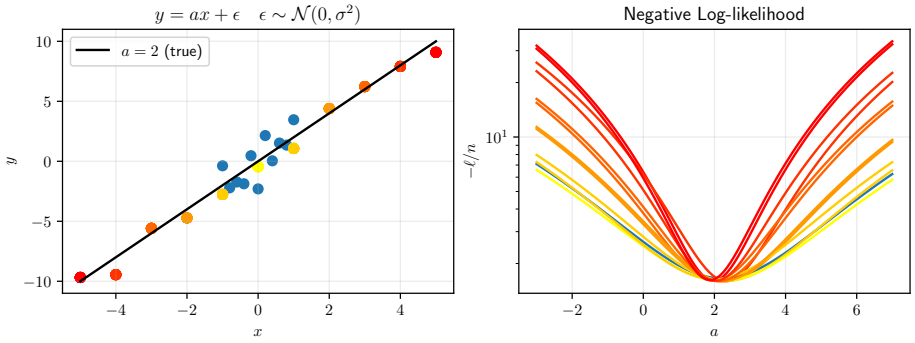

Figure 9: Effect on log-likelihood of adding different points. Left. The blue points are those in our current data set whereas the yellow/red points are candidate points for sampling. Right. Average negative log-likelihood, $-\ell/n$, of the model fitted on the blue points (blue) and the model fitted by added one of the yellow/red points (in corresponding color). Coloring of candidate points is based on the absolute value of x as the effect of adding $\pm x$ is the same in this situation. We see that sampling the red points make the likelihood function more peaked around the estimate of a, i.e., the variance of the estimate is reduced more compared to sampling the yellow points.

locally-weighted linear regression. They use the following quadratic approximation (MacKay, 1992),

$$\mathrm{var}_t(\tilde{x}) \approx \left[\frac{\partial \hat{f}(\tilde{x})}{\partial \theta}\right]^{\mathrm{T}} \left[\frac{\partial^2}{\partial \theta^2} \sum_{x,y \in D_t} (y - \hat{f}(x))^2\right]^{-1} \left[\frac{\partial \hat{f}(\tilde{x})}{\partial \theta}\right] \approx \nabla \tilde{x}^{\mathrm{T}} A_t^{-1} \nabla \tilde{x}, \tag{35}$$

which is an estimate of the output variance for the input instance $\tilde{x}$. The first and third term is the gradient of the output of the estimated model $\hat{f}$ at $\tilde{x}$ with respect to the parameters θ, i.e. $\nabla x = \partial \hat{f}(x)/\partial \theta$. The middle term is the covariance of the parameters θ and can be computed as

$$A_t^{-1} = \left[\frac{\partial^2}{\partial \theta^2} \sum_{x,y \in D_t} (y - \hat{f}(x))^2\right]^{-1} \approx (\sum_{x \in D_t} \nabla x^{\mathrm{T}} \nabla x)^{-1}. \tag{36}$$

In case of a new data point x_{t+1} this covariance is simply updated by addition of the outer product of its gradients, like so

$$A_{t+1}^{-1} \approx (A_t + \nabla x_{t+1}^{\mathrm{T}} \nabla x_{t+1})^{-1}, \tag{37}$$

which furthermore can be expanded to

$$A_{t+1}^{-1} = A_t^{-1} - A_t^{-1} \nabla x_{t+1} (1 + \nabla x_{t+1}^{\mathrm{T}} A_t^{-1} \nabla x_{t+1}) \nabla x_{t+1}^{\mathrm{T}} A_t^{-1} \tag{38}$$

Using the above equations, the derivation of the change in variance of the model output at input $\tilde{x}$ after seeing the new data point x_{t+1} becomes

$$\begin{aligned}
\Delta \,\mathrm{var}_{x_{t+1}}(\tilde{x}) &= \mathrm{var_t}(\tilde{x}) - \mathrm{var}_{t+1}(\tilde{x}) && (39)\\
&= \nabla \tilde{x}^{\mathrm{T}} A_t^{-1} \nabla \tilde{x} - \nabla \tilde{x} A_{t+1}^{-1} \nabla \tilde{x} && (40)\\
&= \nabla \tilde{x}^{\mathrm{T}} (A_t^{-1} - A_{t+1}^{-1}) \nabla \tilde{x} && (41)\\
&= \nabla \tilde{x}^{\mathrm{T}} (A_t^{-1} - A_t^{-1} + A_t^{-1} \nabla x_{t+1} (1 + \nabla x_{t+1}^{\mathrm{T}} A_t^{-1} \nabla x_{t+1}) \nabla x_{t+1}^{\mathrm{T}} A_t^{-1}) \nabla \tilde{x} && (42)\\
&= \nabla \tilde{x}^{\mathrm{T}} A_t^{-1} \nabla x_{t+1} (1 + \nabla x_{t+1}^{\mathrm{T}} A_t^{-1} \nabla x_{t+1} \nabla x_{t+1}^{\mathrm{T}} A_t^{-1}) \nabla \tilde{x} && (43)\\
&= (\nabla \tilde{x}^{\mathrm{T}} A_t^{-1} \nabla x_{t+1})^2 (1 + \nabla x_{t+1}^{\mathrm{T}} A_t^{-1} \nabla x_{t+1}) && (44)\\
&= \mathrm{cov}(\tilde{x}, x_{t+1})^2 (1 + \mathrm{var}(x_{t+1})). && (45)
\end{aligned}$$

Now, we can use this expression to determine the most informative new data point x^* by how much it reduces the variance. Cohn (1994) suggests to use optimization by computing the gradients $\partial \Delta \,\mathrm{var}_{x_t+1}(\tilde{x})/\partial x_{t+1}$ to identify x^* effectively using a membership query synthesis scheme to sample new data. However, for simplicity we will consider a pool-based approach where we find x^* as

$$x^* = \underset{x \in U}{\mathrm{argmax}} \frac{1}{|R|} \sum_{\tilde{x} \in R} \Delta \,\mathrm{var}_x(\tilde{x}), \tag{46}$$

where R is a reference set of data points in the input space for which we evaluate the change in variance, and $|R|$ is the cardinality (number of elements) of the

set R. These reference points can either be a fixed set, or randomly sampled for each iteration. A naive set could be the unlabeled set without the point up for consideration, i.e., $U \setminus x$, but this depends on a representative pool for good results.

The full details of the derivation of these approximations are extensive and can be found in MacKay (1992) and Cohn (1994). The approach described above originates from optimal experimental design, which is very closely tied with active learning, more about this can also be found in the references.

3.2.5 Density Weighting

Uncertainty sampling selects the sample whose label the model is most uncertain about, QBC selects the sample whose label the committee disagrees about the most, and expected impact selects the sample which is believed to change the model maximally. Ideally, these strategies should select samples which are somehow close to the decision boundary and we implicitly assume that such points will also be informative to our model. However, consider for a moment a point which, although close to the decision boundary, is far from most of the other data points. This point will be difficult to classify and will probably also affect the model substantially if added, however, we rarely encounter points in this region at all; it is an outlier. Adding this to our training set is unlikely improve our model for the majority the data which we expect to see and our model only gets better at classifying these "strange" instances.

One way to mitigate this is to use density weighting (Settles, 2009). The idea rests on the assumption that that in order for a point to be informative, it is not enough that we are uncertain about its label, we also need to consider whether it is representative of the underlying data distribution as it might otherwise be an outlier. That is, we want to optimize accuracy in the regions where it really matters. A general way to achieve density weighting is

$$\phi_{\mathrm{dw}}(x) = \phi(x) \left(\frac{1}{|U'|} \sum_{x' \in U'} s(x, x') \right)^{\beta} \tag{47}$$

where $\phi(x)$ measures how informative x is according to some criterion (e.g., uncertainty sampling, QBC, expected impact), U' is the pool from which we can query (i.e., excluding x), $|\cdot|$ denotes cardinality, and s is a function which measures the similarity between x and x'. Thus, density weighted methods takes an initial estimate of informativeness (first term) and weighs it by an estimate of the how representative a sample is (second term). The latter is taken as the average similarity of a sample to all other samples in U'. β is a free parameter which controls the relative importance of the density weighting.

As to the similarity function, s, we may choose any function which somehow estimates the similarity between samples. A simple way to do this is to compute the cosine of the angle between data points (i.e., the normalized dot product)

$$s_{\cos \angle}(x, x') = \frac{x^{\top} x'}{\|x\| \, \|x'\|} \tag{48}$$

where $\|\cdot\|$ denotes the 2-norm. Another example is the radial basis function kernel

$$s_{\mathrm{RBF}}(x, x') = \exp\left(-\gamma \left\| x - x' \right\|^2 \right) \tag{49}$$

where a common choice of γ is one divided by the number of features.

One potential issue with this scheme is that the number of similarities which needs to be calculated grows quadratically with the size of the pool, however, if we are able to precompute the density factor, e.g., when the pool is static, then evaluating eq. (47) is essentially the same as evaluating the base criterion in terms of computational complexity.

References

Angluin, D. (1988). Queries and concept learning. *Machine Learning*, *2*(4), 319–342. https://doi.org/10.1023/A:1022821128753

Cohn, D. A. (1994). Neural network exploration using optimal experiment design (J. D. Cowan, G. Tesauro, & J. Alspector, Eds.). In J. D. Cowan, G. Tesauro, & J. Alspector (Eds.), *Advances in neural information processing systems 6*. Morgan-Kaufmann.

Cohn, D. A., Ghahramani, Z., & Jordan, M. I. (1996). Active learning with statistical models. *J. Artif. Intell. Res.*, *4*, 129–145.

MacKay, D. J. C. (1992). Information-Based objective functions for active data selection. *Neural Comput.*, *4*(4), 590–604.

Mitchell, T. M. (1982). Generalization as search. *Artif. Intell.*, *18*(2), 203–226.

Settles, B. (2009). *Active learning literature survey* (Computer Sciences Technical Report No. 1648). University of Wisconsin–Madison.