Active Learning Lecture Series

Nicholas Denis

August 21, 2019

Contents

1 Lecture 1: Motivations, Background, Definitions, Theoretial Algorithms

1.1 Motivations

Example We begin with a simple motivating example: learning a thresholding function on the unit interval $[0, 1]$. That is, we want to learn an approximately optimal thresholding function $h(x) = 0$ if $x < t^*$ and $h(x) = 1$ if $x \geq t^*$ for some t^* we wish to learn. Suppose we want an ϵ-approximation to h^*, then under uniform sampling, it would take of the order $\frac{1}{\epsilon}$ samples, while using binary search it requires $\log \frac{1}{\epsilon}$ samples, resulting in an exponential efficiency over passive learning. Hence, by playing an active role in how we acquire data to be labeled, a learning algorithm may be able to achieve the same classification accuracy with less data.

1.2 Background

- We are interested in supervised learning
- We have access to an oracle (e.g. humans providing labels)
- We usally begin with a dataset $\mathcal{D} = \mathcal{L} \cup \mathcal{U}$, where $\mathcal{L} := \{x_i, y_i\}_{i=1}^{\ell}$, and $\mathcal{U} := \{x_j\}_{j=\ell+1}^{N}$, where $\ell = |\mathcal{L}|$, N is the cardinality of the entire data set, ane hence there are $N - \ell$ unlabaled x instances.
- *Sample/Label Complexity:* We want to build an approximately optimal classifier with as few labels as possible.
- Acquiring labels is costly, and hence we want to *intelligently* decide which $x \in \mathcal{U}$ to select to query the oracle
- Active Learning vs Passing Learning
- Typically we have some base supervised learning algorithm, and we have an acquisition function (AL strategy), which may or may not use the same supervised learning algorithm.

Organization of the Talk Series:

- Lecture 1:
  - Begin with theoretical motivations and algorithms (which may be intractable).
  - Definitions
  - Heuristics
  - Version space methods
  - Practical algorithm that approximates theory

- Lecture 2:
  - Continue with a good performing algorithm, SVM_{active}.
  - Modern Bayesian approaches to predictive uncertainty based methods (high performing)
- Lecture 3:
  - Combined AL + SSL
  - Modern Deep Generative Model Approaches
  - How to structure an AL experiment

1.3 Definitions

Definition: hypothesis A hyopthesis, h, is map

$$h : \mathcal{X} \to \mathcal{Y}$$

where $\mathcal{X}$ is data space and $\mathcal{Y}$ is the label space (e.g. classification or regression). When we perform supervised learning we use our *inductive bias* to select a *hypothesis space*. We do this by selecting a feature representation of the data, the processing steps we take, and the class of models/algorithms to search over, such as linear regression, decision trees or deep nets.

Definition: hypothesis space (class), $\mathcal{H}_\Theta$ $\mathcal{H}_\Theta = \{h_\theta | h_\theta : \mathcal{X} \to \mathcal{Y}, \theta \in \Theta\}$.

Using a fixed neural network architecture as an example, Θ is the space of all possible weights that the network can take, and θ is a particular set of weights.

We assume the typical supervised learning setup where there exists an underlying probability measure $\mathbb{P}$ (or measurable process) that generates the data on $\mathcal{X}$. Moreover, we assume the realizable setting, where $\exists h^*$, an actual function that generates the *true* labels $y \in \mathcal{Y}$. Given $\mathbb{P}$ and h^*, this induces a measure on $\mathcal{Y}$, and thus on $\mathcal{X} \times \mathcal{Y}$.

Definition: error The error of a given hypothesis, err(h), is given as:

$$\text{err(h)} = \mathbb{P}[h(x) \neq y] = \mathbb{P}[h(x) \neq h^*(x)]$$

The *version space* is a subset of the hypothesis space that contains all the hypotheses that are consistent with the available data.

Definition: version space For a given hypothesis space $\mathcal{H}$, and labeled dataset $\{x_i, y_i\}_{i \in \mathcal{L}}$, the version space, $\mathcal{V} \subset \mathcal{H}$ is defined as

$$\mathcal{V} := \{h \in \mathcal{H} | h(x_i) = y_i, \forall i \in \mathcal{L}\}$$

Definition: disagreement pseudo-metric Given the machinery we have, we can construct a pseudo-metric on $\mathcal{H}$. $\forall\ h, h' \in \mathcal{H}$, define

$$d(h, h') = \mathbb{P}[h(x) \neq h'(x)]$$

A useful concept for the theoretical work in version space based AL is that of the *disagreement region* of a version space.

Definition: disagreement region . Let $\mathcal{V}$ be a version space. The disagreement region of $\mathcal{V}$ is defined as

$$DIS(\mathcal{V}) = \{x \in \mathcal{X} | \exists h, h' \in \mathcal{V}, \text{s.t.} h(x) \neq h'(x)\}$$

The disagreement region is important for version space approaches to AL. The disagreement space is the set of data points where (some) consistent models differ in their predictions. In all version space approaches we will want to select x's from the disagreement region, since getting their labels will allow us to prune away some hypothesis. On a high level, if you are considering a set of possible models (hypothesis/explanations) and they all predict the same a given input, then learning what the true label of that input is not informative; however if half of these models disagree on a given input x, then learning its true label will help discard half the models from further consideration. This intuition is also what is used for disagreement based methods. We note that if we restrict the disagreement region to our dataset (e.g. $x \in \mathcal{D}$), then we use a similarly defined term called *the region of uncertainty*.

Next we introduce a purely theoretical concept, which is somewhat analogous to VC dimension and learnability, in the sense that if this quantity is finite, then active learning has a provable speedup over supervised learning.

Definition: disagreement coefficient . Denote $B(h, r)$ as the ball of radius r around hypothesis $h \in \mathcal{H}$. The disagreement coefficient, θ, is

$$\theta = \sup_{r>0} \frac{\mathbb{P}[DIS(B(h^*, r)]}{r}$$

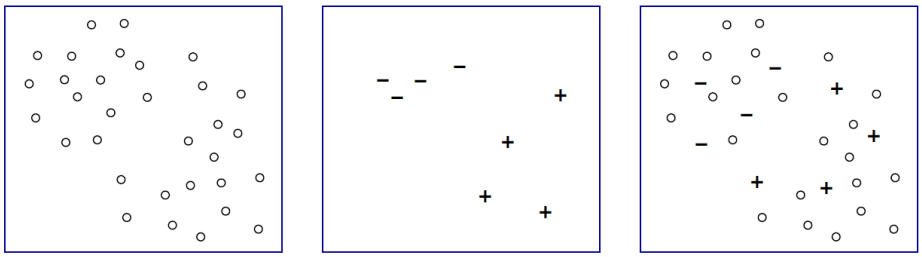


Figure 1: Each circle represents an unlabeled point, while + and − denote points of known label. *Left:* raw and cheap – a large reservoir of unlabeled data. *Middle:* supervised learning picks a few points to label and ignores the rest. *Right:* Semisupervised and active learning get more use out of the unlabeled pool, by using them to constrain the choice of classifier, or by choosing informative points to label.

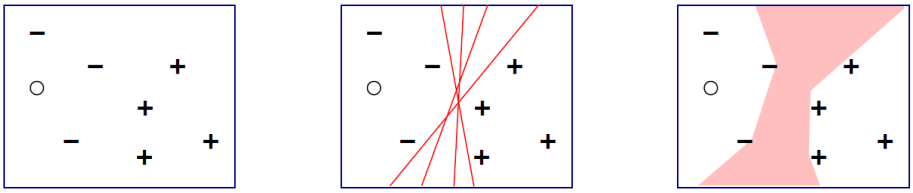


Figure 7: *Left:* The first seven points in the data stream were labeled. How about this next point? *Middle:* Some of the hypotheses in the current version space. *Right:* The region of disagreement.

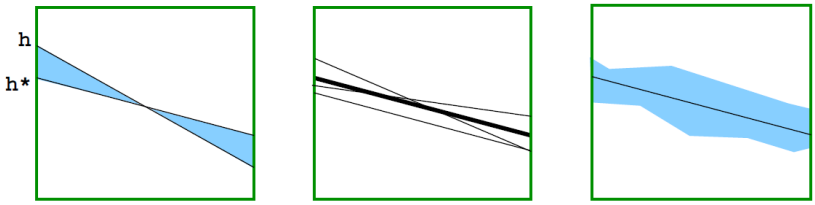


Figure 8: *Left:* Suppose the data lie in the plane, and that hypothesis class consists of linear separators. The distance between any two hypotheses h^* and h is the probability mass (under $\mathbb{P}$) of the region on which they disagree. *Middle:* The thick line is h^*. The thinner lines are examples of hypotheses in $B(h^*, r)$. *Right:* $\mathrm{DIS}(B(h^*, r))$ might look something like this.

Figure 1: Separating Hyperplanes and Version Space.

1.4 General Approaches

- Selecting sampling (query one by one; streaming data)
- Pool based sampling (query in batches)
- Query synthesis (able to query any $x \in \mathcal{X}$, not just from dataset; generative models)

1.5 Heuristic Algorithms

Here we describe some very simple heuristic algorithms. These algorithms have no theoretical gaurantees, however are (mostly) intuitive, but most importantly are easy and cheap to implement. This is a big factor, as typically the more well-principled AL algorithms can be quite expensive to implement. It is worth noting that though there are settings where these approaches can be arbitrarily bad (theoretically), empirically they often perform better than random sampling.

- **Predictive** Entropy
  - At each round of querying, using some base algorithm model h, run all $x \in \mathcal{U}$ to get a "probability score", then compute entropy: $-\sum_y p_\theta(y|x)\log_2 p_\theta(y|x)$.
  - Intuition: high entropy $\approx$ uncertainty, and we should sample in regions of $\mathcal{X}$ that the model is uncertain, under the assumption that as we get more data from this region, the model will become more certain. Note: when the number of classes is quite large, often the entropy computation is dominated by the irrelevant classes. It is easy to cook up examples where predictions that give the top two class labels a "probability" score of 0.5 and 0.4, (seemingly uncertain), and the rest of the mass across, say, 498 classes, and another example with predictions of 0.7, 0.1, 0.08, 0.09, and the reamaining mass across 496 classes, will have higher entropy. A hack (trick) used by some experimenters is to compute the entropy for the top n classes (e.g. $n = 10$).
- **Predictive** Margin
  - Compute $\forall x \in \mathcal{U}$: $p_\theta(y_1^*|x) - p_\theta(y_2^*|x)$, where y_1^* is the highest scored class and y_2^* is the 2nd highest scored class. Margin based methods select those $x's$ with the smallest margin.
  - Intuition: small margin means "uncertain" between two classes, so sample here, in hopes that the model becomes more certain in these regions.
- **Predictive** Variation Ratio

- Find the maximum class prediction: $1 - p_\theta(y_1^*|x)$. Query the x that has the largest value.
- Intuition: if the model is not "certain" about *any* class, then sampling from this region of $\mathcal{X}$...

NOTE: The methods mentioned above use notions of "uncertainty", however without a Bayesian or truly stochastic base algorithm, this interpretation is wrong. Moreover, we must consider different notions of uncertainty: epistemic and aleatoric. Epistemic uncertainty is uncertainty that **can** decrease by acquiring more data. Aleatoric uncertainty is uncertainty that cannot be reduced by acquiring more data, and is intrinsic (perhaps in the method of measurement). There is a growing field of work ML on modeling and representing these forms of uncertainty, but that is another story. The reason why this is important is because if there is a region of the data space $\mathcal{X}$ that is intrinsically noisy (e.g. labels are basically random), then an AL strategy using any of the above mechanisms would want to always sample from this region, as they are "uncertain", however no information can ever be gained from these data points, and so this approach can be arbitrarily bad.

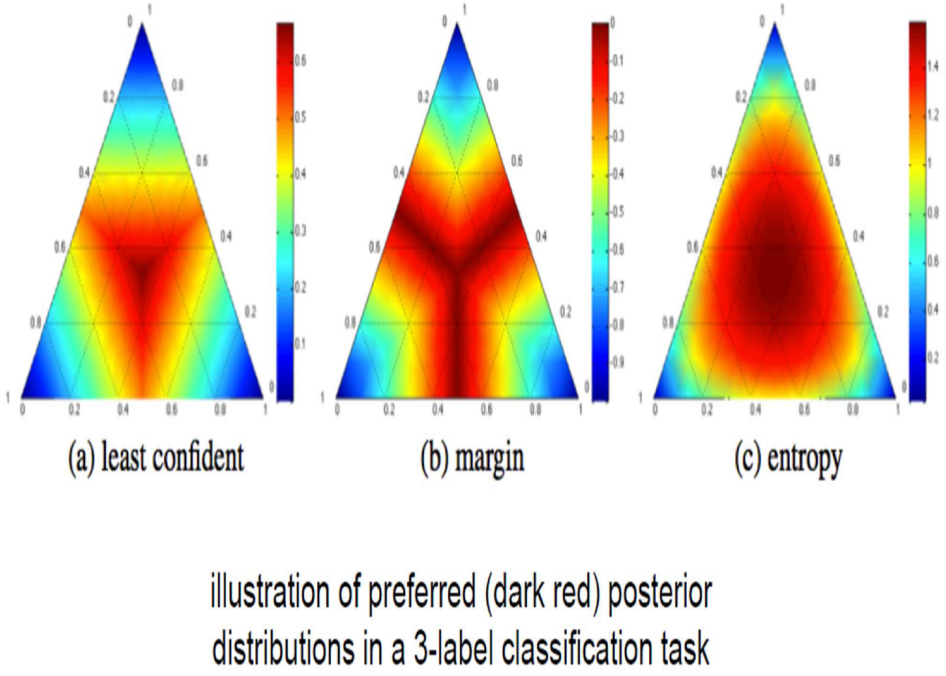


Figure 2: Sampling densities with 3 classes using "uncertainty" sampling heuristics.

1.6 Theoretical Algorithms

1.6.1 CAL (Cohn, Atlas, Ladner)

We begin with a theoretical algorithm, CAL. It is theoretical in the sense that it explicitly maintains the current version space, given the data seen thus far, and we are able to determine if any given x is in the disagreement region. The algorithm receives a sequence of unlabeled data instances x_t and given the current version space, the algorithm checks if x_t is in the disagreement region. If it is, the label is queried from an oracle and the data is added to the labeled dataset and the version space is updated. Otherwise, the data point is discarded. This process is repeated until the version space is sufficiently small where the remaining hypothesis are all basically the same.

Algorithm 1 CAL

procedure CAL($\mathcal{H}$)
  Initalize: $Z_0 := \emptyset$, $\mathcal{V}_0 := \mathcal{H}$ ▷ Labeled data and version space
  for $t = 1, 2, 3, ..., n$ **do**
    Obtain unlabeled data point x_t
    if $\exists h, h' \in \mathcal{V}_{t-1}$ s.t. $h(x_t) \neq h'(x_t)$ **then** ▷ Is x in $DIS(V_{t-1})$?
      $y_t \leftarrow \text{oracle}(x_t)$ ▷ Query oracle ▷ Update dataset
      $Z_t \leftarrow Z_{t-1} \cup \{(x_t, y_t)\}$
    else
      $\tilde{y_t} \leftarrow h(x_t)\ \forall h \in \mathcal{V}_{t-1}$ ▷ We know $h^*(x_t)$
      $Z_t \leftarrow Z_{t-1} \cup \{(x_t, \tilde{y_t})\}$
    end if
    $\mathcal{V}_t \leftarrow \{h \in \mathcal{H} | h(x_i) = y_i, \forall (x_i, y_i) \in Z_i$ ▷ Update version space
  end for
  Return any $h \in \mathcal{V}_n$
end procedure

Theorem 1. *Let $\mathcal{H}$ have finite VC dimension, d. For the realizable setting with disagreement coefficient θ, then the label complexity of CAL, $\mathcal{L}_{CAL}$, $\leq \tilde{\mathcal{O}}\Big(\theta d log\frac{1}{\epsilon}\Big)$.*

1.6.2 Simulating CAL

A method to simulate CAL without having to explicitly maintaing the current version space can be seen below. However even this algorithm assumes we can get perfect accuracy on the training set at each time step

Algorithm 2 Simulate CAL

procedure SIMULATE CAL($\mathcal{H}$)
  Initalize: $Z_0 := \emptyset$ ▷ Labeled data seen thus far
  for $t = 1, 2, 3, ...$ **do**
    Obtain unlabeled data point x_t
    if learn($S \cup (x_t, 1)$) and learn($S \cup (x_t, -1)$) both return an answer **then** ▷ black box SL
      $y_t \leftarrow$ oracle(x_t) ▷ Query oracle ▷ Update dataset
    else
      $y_t \leftarrow$ whichever was successful
    end if
    $Z_t \leftarrow Z_{t-1} \cup \{(x_t, y_t)\}$
  end for
end procedure

1.6.3 Approximating CAL: DHM algorithm

DHM (approximating CAL) is maintaining the same theoretical motivations as the original CAL algorithm. While not required to explicitly maintain the current version space, the algorithm does rely on a black-box "LEARN" function, which given a labelled dataset of known labels, T, and inferred labels, S, LEARN returns a hypothesis consistent with all the data in S (e.g. zero training error), and minimizes the error in T. Though this is perfectly feasible (see Samy Bengio's work: understanding deep learning requires rethinking generalization) using deep neural nets, each call to LEARN may be computationally expensive, as it requires training a model from scratch. Note, the base learning algorithm used in LEARN may differ from the one you may use once you have generated your dataset. Remember - AL is a mechanism to efficiently label datasets from oracles, and we can be orthogonal from our favoriate SL algorithm.

In this algorithm, S is the set of data points with inferred labels, and T is the set of data points labeled by the oracle. The algorithm proceeds iteratively, receiving a new data point x_t, here some counterfactual steps take place - common in AL strategies. Essentially, the algorithm "make believes" that the label is 0, and calls LEARN under this setting, returning h_y^0, and then it "make believes" that the label is 1, and calls LEARN under this setting, returning h_y^1. These two counter factual models are trained an the exact same data, except the switching of the label of a single data point. The error of both models is determined (on some hold out test set, for example), and if the error difference is large, then the model with smaller error is trusted and its labeling of the data point is used as the true label and then added to S. Otherwise, the oracle is queried.

Algorithm 3 Approximating CAL aka DHM

procedure DHM($\mathcal{H}$)
  Initalize: $S_0 := \emptyset, T_0 := \emptyset$
  for $t = 1, 2, 3, ...$ **do**
    Obtain unlabeled data point x_t
    $h_y^0 \leftarrow \text{LEARN}(S_{n-1} \cup \{(x_n, 0)\}, T_{n-1})$
    $h_y^1 \leftarrow \text{LEARN}(S_{n-1} \cup \{(x_n, 1)\}, T_{n-1})$
    if h_y^0, h_y^1 exist and $\text{err}(h_y^1, S_{n-1} \cup T_{n-1}) - err(h_y^0, S_{n-1} \cup T_{n-1}) > \Delta_{n-1}$ **then**
      $S_n \leftarrow S_{n-1} \cup \{(x_t, 0)\}$
      $T_n \leftarrow T_{n-1}$
    else if h_y^0, h_y^1 exists and $\text{err}(h_y^0, S_{n-1} \cup T_{n-1}) - err(h_y^1, S_{n-1} \cup T_{n-1}) > \Delta_{n-1}$ **then**
      $S_n \leftarrow S_{n-1} \cup \{(x_t, 1)\}$
      $T_n \leftarrow T_{n-1}$
    else
      $y_t \leftarrow \text{oracle}(x_t)$ ▷ Query oracle
      $T_n \leftarrow T_{n-1} \cup \{(x_t, y_t)\}$
      $S_n \leftarrow S_{n-1}$
    end if
  end for
end procedure

1.6.4 Query By Committee (QBC)

The QBC algorithm is similar to CAL, except is stochastic in nature. Rather than evaluating all the $h \in \mathcal{V}_t$, we simply need to be able to sample any two $h \in \mathcal{V}_t$, evaluate the current x on them, then proceed as above. The other differences: if the two sampled hypotheses agree, we don't use that as a label, rather, we simply discard this x. Moreover, the stopping criteria is met when a pre-computed number or x instances have been presented to the algorithm and rejected consecutively. QBC can be implemented (approximated) by using an ensemble of classifiers. The idea here is if the ensemble all agree on the labels of a given data point, then we may assume they are all correct. If they are all trained independently, the probability of all making the same prediction and being wrong is quite low. However, if they disagree on the predictions, then this may represent an informative data point.

Algorithm 4 QBC

procedure QBC($\mathcal{H}, \mathcal{T}$)
  Initalize: $\mathcal{V}_0 := \mathcal{H}$ ▷ Labeled data and version space
  $n \leftarrow 0$ ▷ number of queries made thus far
  $t_n \leftarrow 0$ ▷ number of consecutive time steps samples are rejected
  while $t_n < T$ **do**
    Obtain unlabeled data point x
    $y_1, y_2 \leftarrow h_1(x), h_2(x) \sim \text{GibbsSample}(x, \mathcal{V}_n)$
    if $y_1 = y_2$ **then** ▷ Random sample from $\mathcal{V}_n$ consistent ?
      Reject x
      $t_n \leftarrow t_n + 1$
    else
      $y \leftarrow \text{oracle(x)}$ ▷ Query oracle
      $n \leftarrow n + 1$
      $t_n \leftarrow 0$
      $V_n \leftarrow V_{n-1} \cap \{h \in \mathcal{H} | h(x) = y\}$ ▷ Update version space
    end if
  end while
  Return any $h \in \mathcal{V}_n$
end procedure

Under finite VC dimension, d, and if the expected information gain of each query is lower bounded by $g > 0$ bits, then PAC bounds are achieved with an exponentially small fraction of the sampled x being labeled. The authors also show that the expected information gain is simply the entropy of the labels, given a sequence of version spaces (current version space, and the counterfactual future version space), is:

$$\begin{aligned} EIG &= -p_0 \log \frac{P(V_i^0)}{P(V_{i-1})} - p_1 \log \frac{P(V_i^1)}{P(V_{i-1})} \\ &= -p_0 \log p_0 - (1-p_0)\log(1-p_0) \\ &= \mathcal{H}(p_0) \end{aligned}$$

Where $\mathcal{H}(p_0)$ is the entropy. Note that the fractions in the first term is simply the measure of the next version space if the label is 0 (or 1), divided by the total measure of the current version space. This is simply the measure (restricted to the current version space) of those hypotheses consistent with $h(x) = 0$.

1.7 Comments

The theoretial work in AL is quite hairy. However, I find even more so than standard PAC learning theory, the AL theory provides a good motivatio for algorithms that approximate the theoretical algorithms. This is done, for example, buy using ensembles and strong function approximators to mimic sampling from the current version space. This produces disagreement based approaches that still, at their core, are based on version space theory. Overall, by spending some time looking at some of the theoretical approaches to AL, we can motivate more practical algorithms along these notions.

2 Lecture 2: Practical Algorithms I: Heuristics, SVM-based

We have seen thus far some motivations, definitions and some theoretically motivated algorithms. Let's look. Now we will look at some theoretically motivated algorithms that we can actually implement, which act as approximations to the theoretical algorithms.

2.1 SG-NET

SG-NET is an explicit implementation of an approximation to CAL. Denote S^m the sample of m examples. Denote $\mathcal{R}(S^m)$ the *region of uncertainty*. This is similar tothe disagreement region, restricted to the dataset. Let $\alpha = \mathbb{P}[x \in \mathcal{R}(S^m)]$. We note that α acts as an upperbound on the error (loss). Expressing $\mathcal{R}$ explicitly is not possible. However, if we maintain a superset:

$$\mathcal{R}(S^m) \subset \mathcal{R}^+(S^m).$$

With $\mathcal{R}^+$ (and similarly defined $\mathcal{R}^-$), we can sample from $\mathcal{R}^+$ and know that the true region of uncertainty is always contained within it.

The algorithm maintains two subsets, $S, G \subseteq \mathcal{V}_{S^m}$, where S is the "most specific" subset of $\mathcal{V}$, and G is the "most general" subset of $\mathcal{V}$. $S$is the tightest version space consistent on S^m and G is the largest version space consistent on S^m.

$$G = \{h \in \mathcal{V}_{S^m} | \nexists h' \in \mathcal{V}_{S^m}, h^{-1}(\{1\}) \subsetneq h'^{-1}(\{1\})\}$$
$$S = \{h \in \mathcal{V}_{S^m} | \nexists h' \in \mathcal{V}_{S^m}, h'^{-1}(\{1\}) \subsetneq h^{-1}(\{1\})\}$$

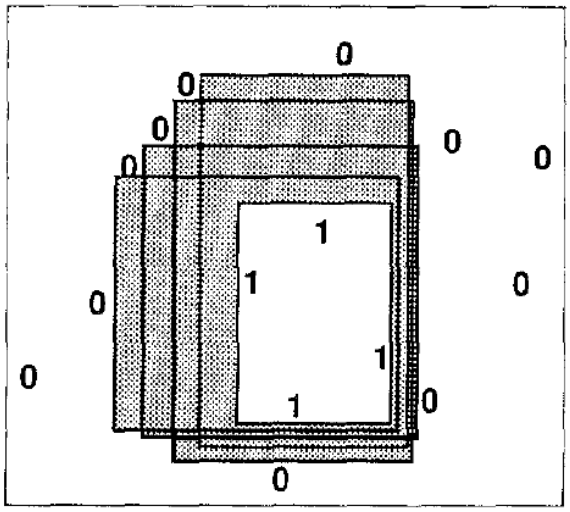


Figure 3: Intuition behind S,G for axis-aligned rectangles.

One can do AL with S, G by looking at data points that fall into the symmetric difference. If $x \in S\Delta G$ and $h^*(x) = 1$, then S will have to grow. If, however, $h^*(x) = -1$, then G will have to shrink. Either way, $S\Delta G$ will decrease over time, and since these act as upper and lower bounds to $\mathcal{V}$, we have that the version space will (hopefully) decrease over time.

2.1.1 Implementing S-net

Learning the most specific version space consistent on S^m can be done by dramatically overfitting on the positive instances within S^m. To do so, the authors suggest the following:

- Divide S^m into positive/negative labeled examples.
- Randomly sample unlabeled instances from $\mathcal{U}$, and pretend their labels are negative
- Train until convergence

I would add the suggestion of creating new instances from a generative model, trained on positive instances, and use those with "negative" labels. Moreover, one could fix the biases to the positive/negative classes (2 class classification) as non-learnable parameters, making the bias low for the negative class, and high for the postive class, biasing the network towards negative examples.

Training the $G-$net is done similarly.

The algorithm procedes as follows:

- Receive x
- if $S(x) \neq G(x)$ then query oracle, and update new dataset
- Re-train S,G using new dataset

2.2 SVM Active Learning

We will assume the binary classification setting, e.g. $y \in \{-1, 1\}$, and that the data in the feature space $\mathcal{F}$ is separable. Given inputs $\{x_1, ..., x_n\}$ in $\mathcal{X}$, and

labels $\{y_1, ..., y_n\}$, we wish to build a maximum margin separating hyperplane, and do so using the following optimization:

$$\begin{aligned} \max_{w \in \mathcal{F}} \quad & \min_i \{y_i \langle w, \Phi(x_i) \rangle\} \\ \text{subject to} \quad & ||w|| = 1 \\ & y_i \langle w, \Phi(x_i) \rangle > 0 \ \forall i \end{aligned}$$

Where $\Phi : \mathcal{X} \to \mathcal{F}$. After solving the SVM, one can perform classification via

$$x \mapsto \text{sign}(\langle w, \Phi(x) \rangle)$$

If the original space $\mathcal{X}$ is suitable (e.g. linearly separable; inner product space), Φ is simply the identity function. However, we can take advantage of RKHS $\mathcal{F}$ via a non-linear map, where the data may be linearly separable. Recall that from Mercer's theorem, using a kernal operator K, one can map data from the input space $\mathcal{X}$ to some RKHS $\mathcal{F}$. The representer theorem gives that the class of such hypotheses are of the form:

$$f(x) = \sum_{i=1}^{n} \alpha_i K(x_i, x).$$

Moreover, Mercer's theorem also states that and we can write $K(u, v) = \langle \Phi(u), \Phi(v) \rangle$, where $w = \sum_{i=1}^{n} \alpha_i \Phi(x_i)$. The SVM computes the appropriate α_i values corresponding to the maximal margin hyperplane in $\mathcal{F}$.

2.2.1 RKHS Primer

Let X be a set and H be a Hilbert space of real-valued functions on X. Define $L_x : f \mapsto f(x)$, $\forall f \in H$. E.g. $L_x \in H^*$, the dual space. H is a reproducing kernel Hilbert space (RKHS) if, $\forall x \in X$, L_x is continuous (e.g. a bounded operator, e.g. $\exists M > 0$ s.t. $|L_x(f)| = |f(x)| \leq M||f||_H$, $\forall f \in H$. By Riesz representation theorem, then $f(x) = L_x(f) = \langle f, K_x \rangle$, $\forall f \in H$.

Moreover, since K_x is itself in H, then $\exists K_y \in H$ s.t. $K_x(y) = \langle K_x, K_y \rangle$. Hence, the reproducing kernel $K : X \times X \to \mathbb{R}$ is defined: $K(x, y) = \langle K_x, K_y \rangle$.

The authors assume the data is normalized, in the sense that $\forall i, ||\Phi(x)|| = \lambda$, for some fixed constant λ. WLOG, we may assume $\lambda = 1$.

2.2.2 Version Space

The hypothesis space,

$$\mathcal{H} = \left\{ f | f(x) = \frac{\langle w, \Phi(x) \rangle}{||w||} \text{where } w \in \mathcal{W} \right\}$$

and the parameter space is simply $\mathcal{F}$. The version space is defined as:

$$\mathcal{V} = \{f \in \mathcal{H} | \forall i \in \{1, 2, ..., n\} \;\; y_i f(x_i) > 0\}.$$

Note, that since $\mathcal{H}$ is a set of hyperplanes that are in bijection with unit vectors $w \in \mathcal{W}$, we have

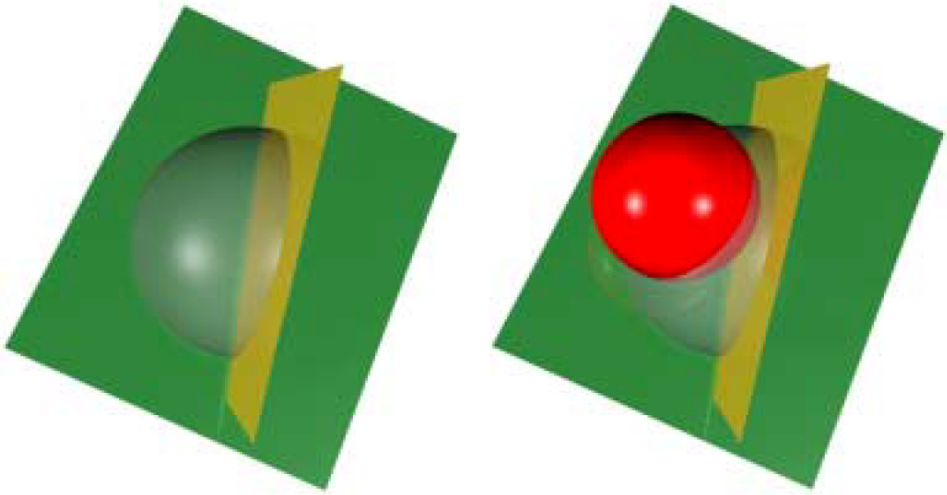
$$\mathcal{V} = \{w \in \mathcal{W} : ||w|| = 1, y_i f(x_i) > 0 \forall i \in \{1, 2, ..., n\}\}.$$

NOTE: A version space only exists if the training data is linearly separable in the feature space. Luckily, this has been shown to always be possible.

NOTE: There exists a duality between the feature space $\mathcal{F}$ and the parameter space $\mathcal{W}$: points in $\mathcal{F}$ correspond to hyperplanes in $\mathcal{W}$, and points in $\mathcal{W}$ correspond to hyperplanes in $\mathcal{F}$.

2.2.3 Duality

The version space for the SVM is the surface of the unit hyper-sphere, since $||w|| = 1$. Note, a point in $\mathcal{W}$ corresponds to a (separating) hyperplane in $\mathcal{F}$, and similarly a data point in $\mathcal{F}$ corresponds to a hyperplane in $\mathcal{W}$, since it (can) restricts the space of possible w that can be consistent with the data thus far (see figure 4). Moreover, an SVM classifier can be represented as a sphere with center as a point in the version space (e.g. on the unit sphere in $\mathcal{W}$). The solution of the SVM is such a sphere with the *largest* radius whose surface does not interesect with the limiting hyperplanes. The center of the sphere (lying in the version space, e.g. the unit sphere in $\mathcal{W}$), is the SVM, and its **radius is proportional to the margin of the SVM** in $\mathcal{F}$. The training points corresponding to hyperplanes that it touches are the support vectors.



(a) (b)

Figure 2: (a) Version space duality. The surface of the hypersphere represents unit weight vectors. Each of the two hyperplanes corresponds to a labeled training instance. Each hyperplane restricts the area on the hypersphere in which consistent hypotheses can lie. Here, the version space is the surface segment of the hypersphere closest to the camera. (b) An SVM classifier in a version space. The dark embedded sphere is the largest radius sphere whose center lies in the version space and whose surface does not intersect with the hyperplanes. The center of the embedded sphere corresponds to the SVM, its radius is proportional to the margin of the SVM in $\mathcal{F}$, and the training points corresponding to the hyperplanes that it touches are the support vectors.

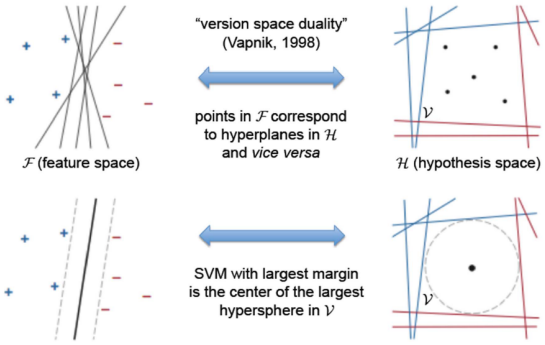


Figure 4: Separating Hyperplanes and Version Space, parameter space $\mathcal{W}$.

Essentially, solving an SVM means finding a ball of maximal radius, whose center lies on the unit sphere in $\mathcal{W}$ (e.g. the version space $\mathcal{V}$), that does not intersect with any separating hyperplanes. This geometric intuition is used to build an active learning strategy. The goal is to select samples for querying that will reduce the size of the version space as much as possible.

Definition: *Area*($\mathcal{V}$) is the surface area of the version space.

Definition: Let $\mathcal{V}_i$ be the version space after i iterations. Now, given the $(i+1)$th query, x_{i+1}, define:

$$\mathcal{V}_i^- = \mathcal{V}_i \cap \{w \in \mathcal{W} | - \langle w, \Phi(x_{i+1}) \rangle > 0\}$$
$$\mathcal{V}_i^+ = \mathcal{V}_i \cap \{w \in \mathcal{W} | \langle w, \Phi(x_{i+1}) \rangle > 0\}$$

Here, $\mathcal{V}^-$ is the version space resulting after building a new SVM when the algorithm pretends the label of x_{i+1} is -1, and $\mathcal{V}^+$ is defined similarly.

The authors have a proof that the optimal strategy is the minimize the maximum expected size of the version space, where the maximum is taken over all conditional distributions of y given x. Though computing *Area*($\mathcal{V}$) may be intractable, the authors suggest some approaches to approximate it.

Simple Margin In simple margin, the assumption is that at any given iteration, i, the current SVM separating hyperplane w_i is "centrally located" on the surface of the current version space $\mathcal{V}_i$, since otherwise if it is too close to a separating hyperplane in $\mathcal{W}$, it will have a smaller radius. Hence, in order to split the current version space $\mathcal{V}_i$ as evenly as possible, the unlabelled instance x_i is selected that is closest to w_i. In doing so, in parameter space $\mathcal{W}$ it will divide the remaining version space evenly in half. In the primal form, this is simply the data instance x_i that lies closest to the separating hyperplane, and hence minimizes: $|\langle w_i, \Phi(x) \rangle|$.

Hence, for this AL strategy, for each x in the unlabeled dataset $\mathcal{U}$ the distance of x from the separating hyperplane is computed, and the closest x is selected for labeling.

MaxMin Margin Simple margin is a very rough approximation, which assumes that the current version space is symmetric, and that w_i is centrally located. In reality, this is not the case, and also it is possible that SimpleMargin selects points that does not even change the separating hyperplane (e.g. does not even intersect the version space, in $\mathcal{W}$ space). Returning to the notion that the **radius of the largest sphere in $\mathcal{W}$ space is achieved by the SVM, and that this is proportional to the margin length, we can use the margin length as a proxy for the radius induced by the SVM**.

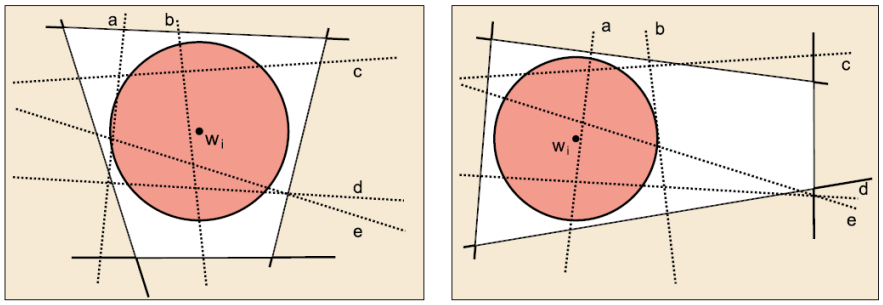
In doing so, given x_i, we can compute V_i^- and V_i^+, and the resulting margins $m_-, m_+ arecomputed$. Consider $min(Area(\mathcal{V}^-), Area(\mathcal{V}^+))$. Since $\mathcal{V}$ is being split into $\mathcal{V}^-$ and V_i^+, and we want these splits to be as equal as possible. To do this the authors suggest finding the x that maximizes $min(Area(\mathcal{V}^-), Area(\mathcal{V}^+))$. However, we do not have access to the Area(), thus we will approximate this by $min(m_-, m_+)$ and select x_i that maximizes the amount.

For this AL strategy, for each x in the unlabeled dataset $\mathcal{U}$, $|C|$ new SVM's are built, where C is the number of classes. Each newly built SVM is trained with x and the label $c \in C$. Hence, at each active learning step a total of $|C||\mathcal{U}|$ SVMs are built.

Ratio Margin This approach is similar to MaxMin approach, except that it attempts to take into account the current version space $\mathcal{V}_i$. Rather than looking at absolute sizes, the relative sizes is computed. Hence, for each x, the value: $min(\frac{m_-}{m_+}, \frac{m_+}{m_-})$ is computed, and x_i is selected which maximizes this value.

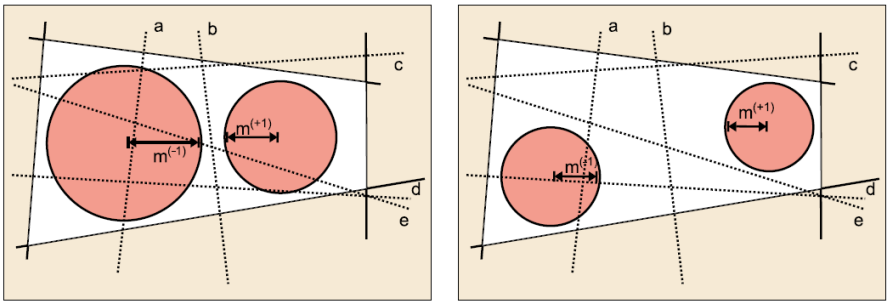
NOTES:

- These methods approximate the acquisition function that halves the version space.
- Margin can be used as an indication of version space size, even if the assumption of fixed norm is true or not.



(a) (b)

Figure 3: (a) Simple Margin will query **b**. (b) Simple Margin will query **a**.



(a) (b)

ıre 4: (a) MaxMin Margin will query **b**. The two SVMs with margins m^- and m^+ for **b** are shown. (b) Ratio Margin will query **e**. The two SVMs with margins m^- and m^+ for **e** are shown.

Figure 5: Separating Hyperplanes and Version Space, parameter space $\mathcal{W}$.

2.2.4 RESULTS

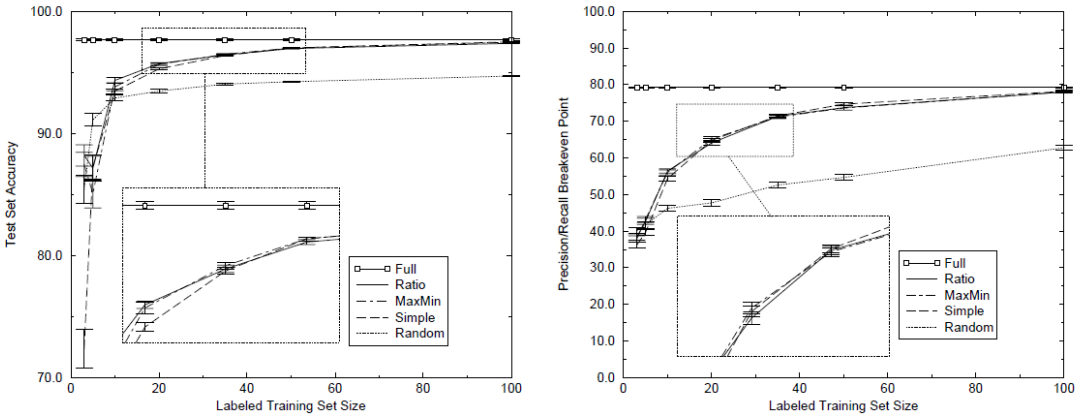


Figure 5: (a) Average test set accuracy over the ten most frequently occurring topics when using a pool size of 1000. (b) Average test set precision/recall breakeven point over the ten most frequently occurring topics when using a pool size of 1000.

Topic	Simple	MaxMin	Ratio	Equivalent Random size
Earn	86.39 ± 1.65	87.75 ± 1.40	90.24 ± 2.31	34
Acq	77.04 ± 1.17	77.08 ± 2.00	80.42 ± 1.50	> 100
Money-fx	93.82 ± 0.35	$\mathbf{94.80 \pm 0.14}$	$\mathbf{94.83 \pm 0.13}$	50
Grain	95.53 ± 0.09	95.29 ± 0.38	95.55 ± 1.22	13
Crude	95.26 ± 0.38	95.26 ± 0.15	95.35 ± 0.21	> 100
Trade	96.31 ± 0.28	96.64 ± 0.10	96.60 ± 0.15	> 100
Interest	96.15 ± 0.21	96.55 ± 0.09	96.43 ± 0.09	> 100
Ship	97.75 ± 0.11	97.81 ± 0.09	97.66 ± 0.12	> 100
Wheat	98.10 ± 0.24	98.48 ± 0.09	98.13 ± 0.20	> 100
Corn	98.31 ± 0.19	98.56 ± 0.05	98.30 ± 0.19	15

Table 1: Average test set accuracy over the top ten most frequently occurring topics (most frequent topic first) when trained with ten labeled documents. Boldface indicates statistical significance.

Figure 6: Experimental Results (Reuters dataset).

3 Lecture 3: Practical Algorithms II: Bayesian, SSL, Generative Models + Comments

3.1 Information Theory Based Active Learning Strategies

The authors take a Bayesian approach, specifically a Gaussian Process Classifier (GPC). The model class is parameterized by θ,and a given model predicts

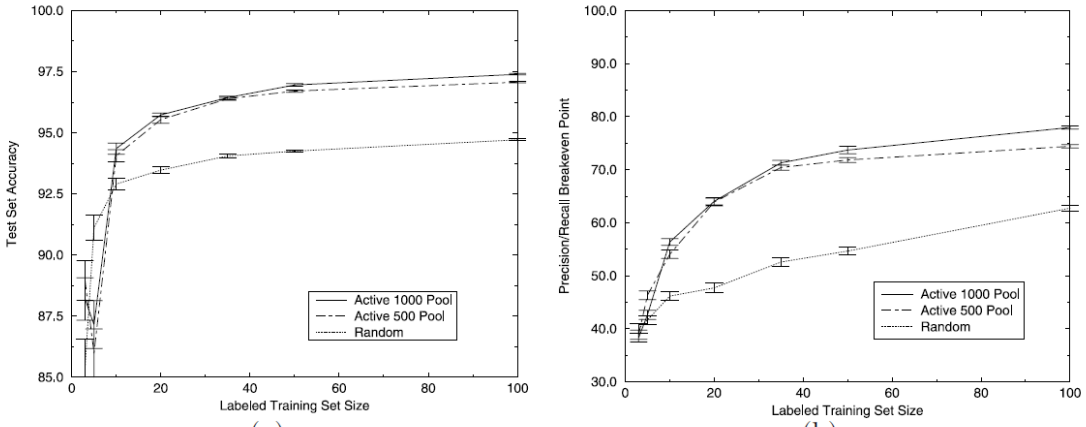


Figure 7: (a) Average test set accuracy over the ten most frequently occurring topics when using a pool sizes of 500 and 1000. (b) Average breakeven point over the ten most frequently occurring topics when using a pool sizes of 500 and 1000.

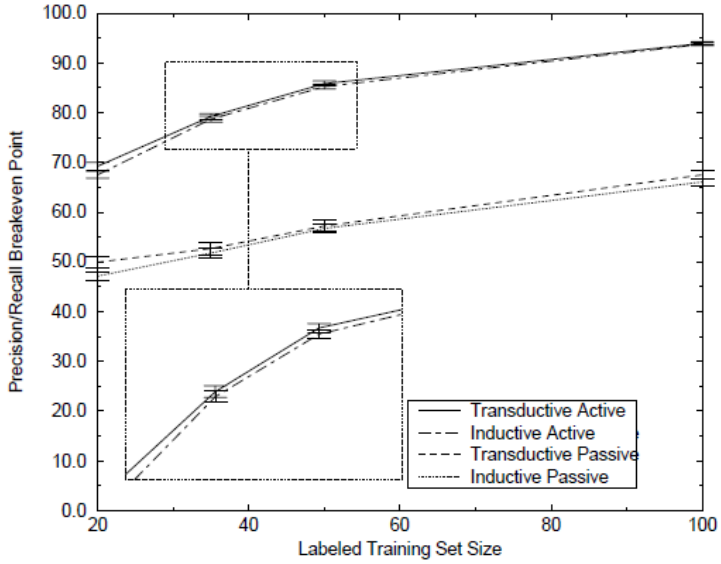


Figure 7: Comparing Pool Sizes (top). Transductive and Inductive (bottom).

$p(y|x;\theta)$. After having seen data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, a posterior distribution over parameters is inferred, $p(\theta|\mathcal{D})$. Similar to the version space methods that aim to reduce the size of the version space, the goal of information theoretic active

learning is to reduce the posterior entropy as quickly as possible. This is a measure how much, on average, the models within our model class disagree over a prediction, where the weight is given by the posterior distribution. This is done via minimizing the uncertainty about the model parameters via entropy:

$$\underset{\mathcal{D}'}{arg\,min}\ H[\theta|\mathcal{D}'] = -\int p(\theta|\mathcal{D}')\log p(\theta|\mathcal{D}')d\theta$$

Of course, rather than searching over all possible datasets $\mathcal{D}'$, we restrict ourselves to selecting the *next* datapoint, x, that **maximizes the decrease in expected posterior entropy**:

$$\underset{x}{arg\,max}\ H[\theta|\mathcal{D}] - \mathbb{E}_{y\sim p(y|x,\mathcal{D})}[H[\theta|y,x,\mathcal{D}]]$$

Note that this expectation is taken over the unseen output y values. When the dimensionality of the parameter space is high or even infinite (kernel methods), computing the entropies becomes intractable. Some methods use Monte Carlo sampling, others use a tractable distribution as proxy (e.g. variational methods) and minimize KL divergence between the two distributions.

This objective is equivalent to the conditional mutual information between the unknown output and the parameters: $I[\theta, y|x, \mathcal{D}]$. Moreover, this objective can be rearranged and represented as entropies in y space:

$$\underset{x}{arg\,max}\ H[y|x,\mathcal{D}] - \mathbb{E}_{\theta\sim p(\theta|\mathcal{D})}[H[y|x,\theta]].$$

The above is the expected information gained between y and model parameters given x, and $\mathcal{D}$. Hence, we are selecting the x that greedily improves the mutual information.

This objective has some nice benefits: entropies are now calculated in a lower dimensional space (y space), which for binary classification is just the entropy of Bernoulli random variables. The objective seeks the x instance for which the the model is marginally most uncertain about (e.g. $H[y|x,\mathcal{D}]$, marginal in that it is over all parameters), but for which individual settings of the parameters are confident (low $\mathbb{E}_{\theta\sim p(\theta|\mathcal{D})}[H[y|x,\theta]]$). **This (especially our original formulation) can be interpreted as finding the x for which the parameters under the posterior disagree about the outcome**, y, the most. Hence, they give this the name **Bayesian Active Learning by Disagreement (BALD)**.

3.1.1 Gaussian Processes

Feel free to skip this section on GPs. The main point of this paper is that they introduce an objective function: BALD, that determines the active learning query process. Any Bayesian methodology and (approximate) solution approach works, as long as it can compute (an estimate of) the BALD objective function. The authors used a Gaussian process approach, which is briefly described below.

The authors choose to use GP's for their Bayesian setting. A GP is a probabilistic model, where:

$$f \sim GP(\mu(.), k(.,.))$$
$$y|x, f \sim Bernoulli(\Phi(f(x)))$$

where f is a function sampled from a function space with mean μ and covariance kernel k. The authors assume binary classification, and Φ is the Gaussian CDF. Inferene in the GPC model is intractable. Various approximation methods can be considered, however this work is agnostic to the approach, and so doesn't care which approximation approach used. These methods compute approximations to $\mu_{x,\mathcal{D}}$, and $\sigma_{x,\mathcal{D}}$.

To compute the objective function, the first term uses any approximation of $\mu_{x,\mathcal{D}}$, and $\sigma_{x,\mathcal{D}}$, then computes the first entropy term analytically. The second term can be approximated using Taylor series expansion, which the authors how is highly accurate. Moreover, for most kernels the objective for term 2 is smooth and differentiable, so gradient-based optimization procedures can be used. In any case, I believe these are mostly design choices and specific to GPC's. The real key here is their formulation of BALD.

The results are quite positive, and test on some synthetic datasets (noisy regions, uninformative regions), as well as some publically available datasets.

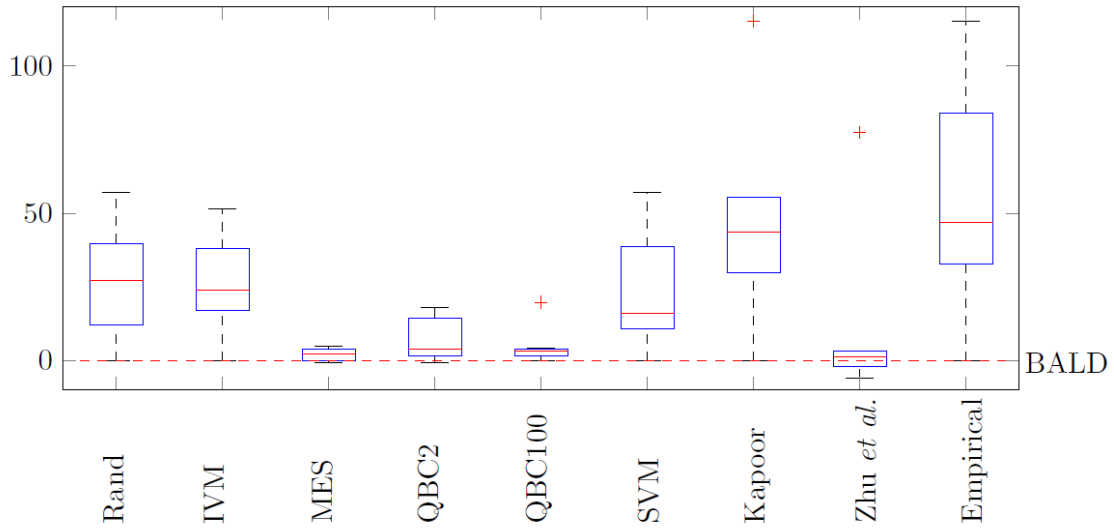


Figure 8: BALD results compared to other AL approaches. Y axis lists the extra number of samples required to reach the same accuracy as the BALD approach. SVM = SVM_{active}; Zhu et al. = AL + SLL using Gaussian fields and harmonic functions.

3.2 Deep Bayesian Active Learning: BALD

The authors have previously shown that by utilizing dropout at inference time, such a NN can be seen as a variational approximation to a GP with a Gaussian prior over the weights. Monte Carlo integration is used to approximate the posterior, in that:

$$\begin{aligned} p(y = c|x, \mathcal{D}_{train}) &= \int p(y = c|x, \omega) p(\omega|\mathcal{D}_{train}) d\omega \\ &\approx \int p(y = c|x, \omega) q_\theta^*(\omega) d\omega \\ &\approx \frac{1}{T} \sum_{i=1}^{T} p(y = c|x, \hat{\omega}_t) \end{aligned}$$

where $\hat{\omega}_t \sim q_\theta^*(\omega)$ is the dropout distribution. The authors state that deterministic models are only capable of capturing **aleatoric uncertainty**, but cannot capture **epistemic uncertainty** - uncertainty over the network parameters, which we are attempting to minimize during active learning.

Acquisition Functions

Predictive Entropy

$$\mathbb{H}[y|x, \mathcal{D}_{train}] = -\sum_c p(y = c|x, \mathcal{D}_{train}) \log p(y = c|x, \mathcal{D}_{train}).$$

BALD

$$\mathcal{I}[y, \omega|x, \mathcal{D}_{train}] = \mathbb{H}[y|x, \mathcal{D}_{train}] - \mathbb{E}_{p(\omega|\mathcal{D}_{train})}[\mathbb{H}[y|x, \omega]]$$

where $\mathcal{I}$ is the mutual information between predictions and model parameters ω.

Predictive Variation Ratios

$$1 - \underset{y}{max}\ p(y|x, \mathcal{D}_{train})$$

Maximum Mea Standard Deviation

$$\sigma_c = \sqrt{\mathbb{E}_{q(\omega)}[p(y = c|x, \omega)^2] - \mathbb{E}_{q(\omega)}[p(y = c|x, \omega)]^2}$$
$$\sigma(x) = \frac{1}{C} \sum_c \sigma_c$$

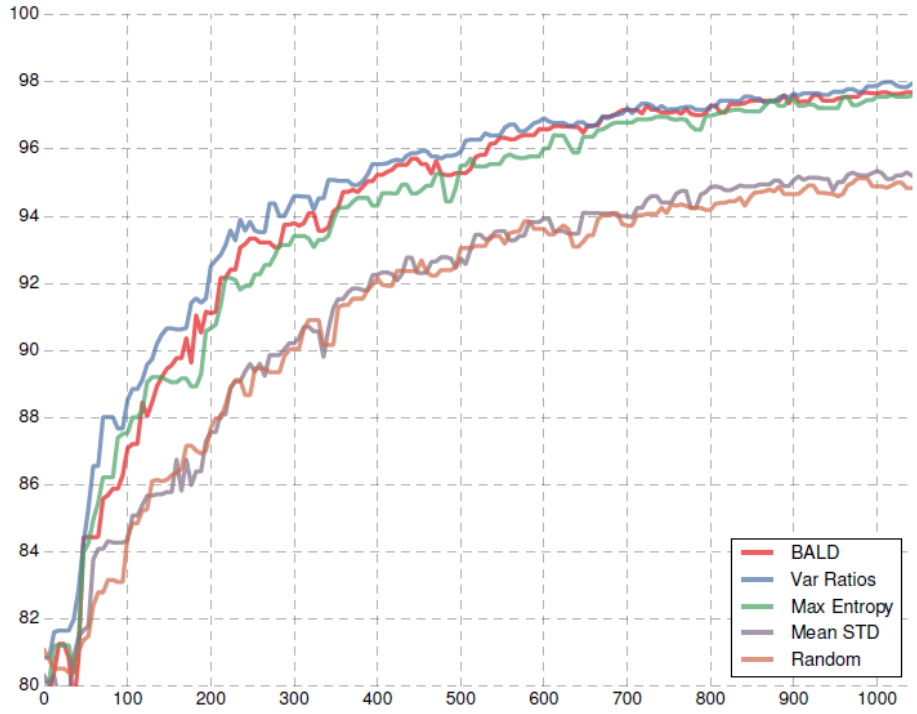
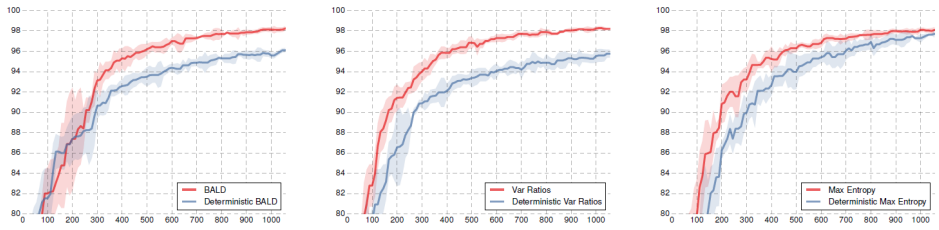


Figure 1. **MNIST test accuracy as a function of number of acquired images from the pool set** (up to 1000 images, using validation set size 100, and averaged over 3 repetitions). Four acquisition functions (*BALD*, *Variation Ratios*, *Max Entropy*, and *Mean STD*) are evaluated and compared to a *Random* acquisition function.



(a) BALD (b) Var Ratios (c) Max Entropy

Figure 2. Test accuracy as a function of number of acquired images for various acquisition functions, using both a **Bayesian CNN (red) and a deterministic CNN (blue)**.

Figure 9: Comparing acquisition functions (top). Bayesian vs deterministic (bottom).

In order to acieve accuracy of 0.9:

- BALD = 145 samples
- Var Ratios = 120 samples
- Max Ent = 165 samples
- Random = 255 samples

In order to acieve accuracy of 0.95:

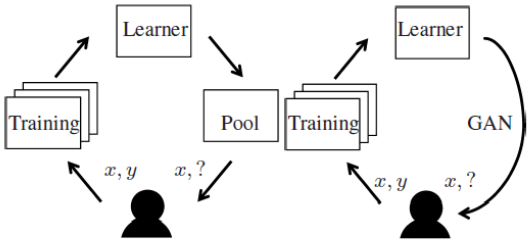
- BALD = 335 samples
- Var Ratios = 295 samples
- Max Ent = 355 samples
- Random = 835 samples

4 Active Learning using Generative Models

The approaches considered using generative models no longer require the AL algorithm to query an oracle for the label of an $x \in \mathcal{U}$, but rather, with access to a generative model, can synthetically sample from some generative model that approximates $\mathbb{P}_{data}(X)$.

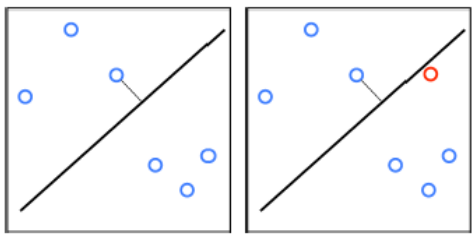
4.1 Generative Adversarial Active Learning

This paper can be seen as either AL, semi-supervised learning, self-supervised learning or simply a form of data augmentation. The idea is that there is some base classifier (here SVM), a GAN is built and used to generate synthetic instances that are close to the SVM hyperplane (SVM_{active}, simple margin). This instance is queried by an oracle, and then added to the data set, then the SVM is re-trained. The authors start with an initial seed set of 50 instances, then do batches of 10 queries at a time. The authors use binary classification (5 vs 7 for MNIST, Car vs Horse CIFAR10).



(a) Pool-based (b) GAAL

$$\min_z \left\{ \frac{1}{2} \| W^\top \phi(G(z)) + b \|^2 + \lambda \log(1 - D(G(z))) \right\}$$



(a) SVM_{active} (b) GAAL

Figure 10: GAAL.

Algorithm 1 Generative Adversarial Active Learning (GAAL)

Train generator G on all unlabeled data by solving (2)
Initialize labeled training dataset $\mathcal{S}$ by randomly picking a small fraction of the data to label
repeat
  Solve optimization problem (4) according to the current learner by descending the gradient

$$\nabla_z\{\frac{1}{2}\|W^\top\phi(G(z))+b\|^2+\lambda\log(1-D(G(z)))\}$$

  Use the solution $\{z_1, z_2, \dots\}$ and G to generate instances for querying
  Label $\{G(z_1), G(z_2), \dots\}$ by human oracles
  Add labeled data to the training dataset $\mathcal{S}$ and re-train the learner, update W, b
until Labeling budget is reached

Figure 11: GAAL Algorithm.

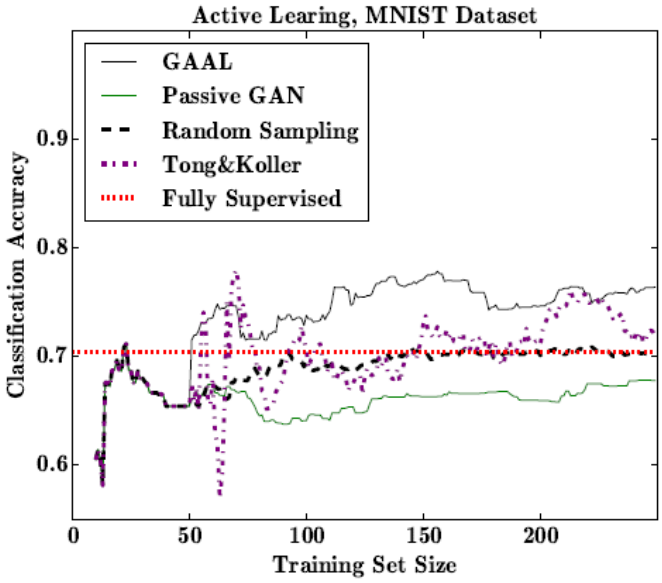


Figure 12: GAAL Results.

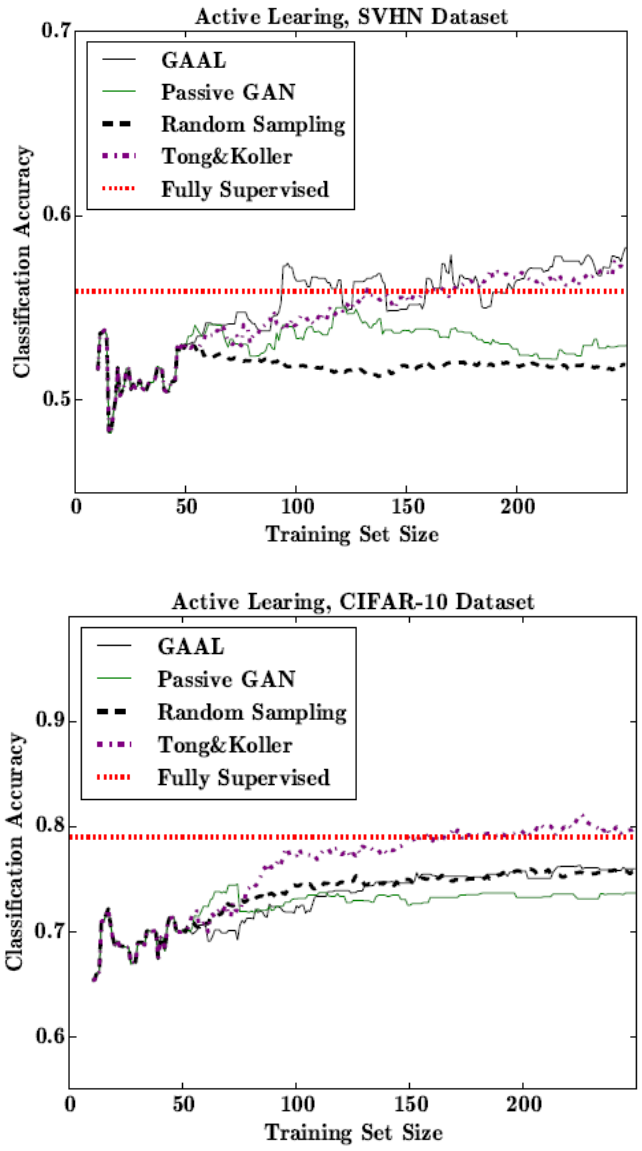


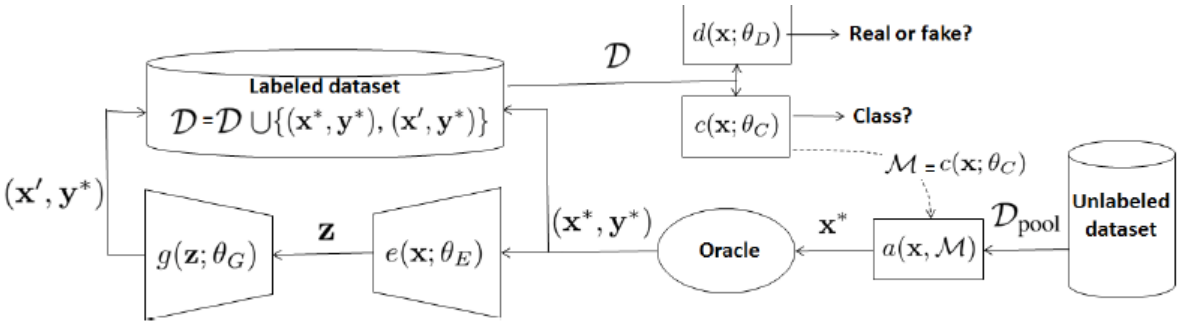
Figure 13: GAAL Results.

4.2 Bayesian Generative Active Deep Learning

The authors use BALD, AC-GAN (auxiliary classifier - GAN), VAE's to build up their method.

AC-GAN: AC-GAN, the generator generates instances conditioned on the noise variable, as well as the class label. The objective function contains two components: the normal GAN (real vs. fake) and the classifier (expected log probability of the class conditioned on real or fake image).

- The authors use an VAE to learn a latent space representation of the data, and to generate synthetic instances.
- The training process uses a sufficient number of VAE-training iterations such that the recunstrctions error is below a threshold value.
- The authors use VAE + ACGAN (e.g. adversarial autoencoder) and simply the ACGAN (no VAE componenet).
- Experiments:
- Initial training seed is 1000 for MNIST, 5000 for CIFAR10, 15000 for CIFAR100, 10000 for SVHN.
- Number of acquisition steps is 150 (50 for SVHN) where they acquire 100 (500 for SVHN) at each step from a randomly selected pool of 2000 unlabelled x.



Algorithm 1 Bayesian Generative Active Learning

Initialize network parameters $\theta_E, \theta_G, \theta_C, \theta_D$, and pre-train the classifier $c(\mathbf{x};\theta_C)$ with $\mathcal{D}$

repeat

Pick the most informative $\mathbf{x}^*$ from $\mathcal{D}_{\text{pool}}$ with $\mathbf{x}^* = \arg\max_{\mathbf{x}\in\mathcal{D}_{\text{pool}}} a(\mathbf{x},\mathcal{M})$ in (1) and (2), where $\mathcal{M}$ is represented by the classifier $c(\mathbf{x};\theta_C)$;

Request the oracle to label the selected sample, which forms $(\mathbf{x}^*,\mathbf{y}^*)$

$\mathbf{z} \leftarrow e(\mathbf{x}^*;\theta_E)$

$\mathcal{L}_{\text{prior}} \leftarrow D_{\text{KL}}(q(\mathbf{z}|\mathbf{x}^*)\|p(\mathbf{z}))$

$\mathbf{x}' = g(e(\mathbf{x}^*);\theta_G)$

$\mathcal{L}_{\text{rec}} \leftarrow \mathcal{L}_{\text{rec}}(\mathbf{x}^*,\mathbf{x}')$

Sample $\mathbf{u}\sim\mathcal{N}(\mathbf{0},\mathbf{I})$

$\mathcal{L}_{\text{ACGAN}} \leftarrow \log(d(\mathbf{x}^*)) + \log(1-d(\mathbf{x}')) + \log(1-d(g(\mathbf{u}))) + \log(\text{softmax}(c(\mathbf{x}^*))) + \log(\text{softmax}(c(\mathbf{x}'))) + \log(\text{softmax}(c(g(\mathbf{u}))))$

$\theta_E \leftarrow \theta_E - \nabla_{\theta_E}(\mathcal{L}_{\text{rec}} + \mathcal{L}_{\text{prior}})$

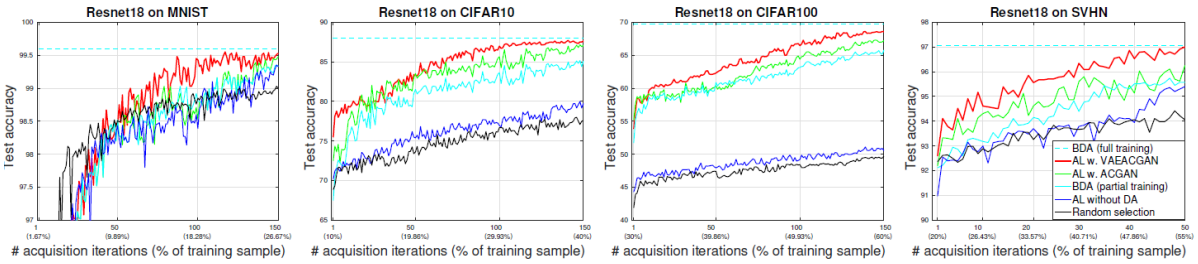
$\theta_G \leftarrow \theta_G - \nabla_{\theta_G}(\gamma\mathcal{L}_{\text{rec}} - \mathcal{L}_{\text{ACGAN}})$ (parameter $\gamma = 0.75$ (Larsen et al., 2016) in our experiments)

$\theta_D \leftarrow \theta_D - \nabla_{\theta_D}\mathcal{L}_{\text{ACGAN}}$

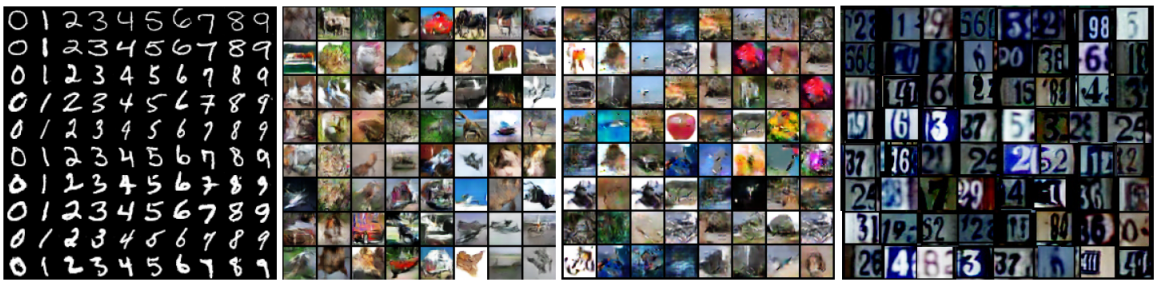
$\theta_C \leftarrow \theta_C - \nabla_{\theta_C}\mathcal{L}_{\text{ACGAN}}$

until convergence

Figure 14: Bayesian GAAL.



	AL w. VAEACGAN	AL w. ACGAN	AL w. PMDA	AL without DA	BDA (partial training)	Random selection
			MNIST			
ResNet18	$\mathbf{99.53 \pm 0.05}$	99.45 ± 0.02	99.37 ± 0.15	99.33 ± 0.10	99.33 ± 0.04	99.00 ± 0.13
ResNet18pa	$\mathbf{99.68 \pm 0.08}$	99.57 ± 0.07	99.49 ± 0.09	99.35 ± 0.11	99.35 ± 0.07	99.20 ± 0.12
			CIFAR-10			
ResNet18	$\mathbf{87.63 \pm 0.11}$	86.80 ± 0.45	82.17 ± 0.35	79.72 ± 0.19	85.08 ± 0.31	77.29 ± 0.23
ResNet18pa	$\mathbf{91.13 \pm 0.10}$	90.70 ± 0.24	87.70 ± 0.39	85.51 ± 0.21	86.90 ± 0.27	80.69 ± 0.19
			CIFAR-100			
ResNet18	$\mathbf{68.05 \pm 0.17}$	66.50 ± 0.63	55.24 ± 0.57	50.57 ± 0.20	65.76 ± 0.40	49.67 ± 0.52
ResNet18pa	$\mathbf{69.69 \pm 0.13}$	67.79 ± 0.76	59.67 ± 0.60	55.82 ± 0.31	65.79 ± 0.51	54.77 ± 0.29



(a) MNIST (b) CIFAR-10 (c) CIFAR-100 (d) SVHN

Figure 15: Bayesian GAAL.

5 Combining Active Learning and Semi-Supervised Learning using Gaussian Fields and Harmonic Functions

X. Zhu J. Lafferty, Zoubin Ghahramani

This paper uses the manifold regularized SSL approach. A kernel is used to measure distance of all the data, and labels are propagated on this Gaussian field. Weights of the graph are the distances computed between all data. Active learning is performed on top of the SSL approach by greedily selecting queries from the unlabeled data to minimize the estimated expected classification error. This can all be done efficiently using matrix methods.

5.1 Gaussian random fields and harmonic energy minimizing functions

Assume $(x_1, y_1), ..., (x_l, y_l)$ ℓ labeled data, and $\ell + 1, ..., \ell + u$ unlabeled data. For a total of $n = \ell + u$ data points. We use a metric (e.g. RBF) and compute weights $w_{ij} = \exp\Big(- \frac{1}{\sigma^2} ||x_i - x_j||_2 \Big)$. Scale parameter σ is learned or a hyperparameter. We maintain the labels for the labeled data $y \in \{0, 1\}$, and for the unlabeled data we predict a mapping $f : V \to [0, 1]$, where V is the vertex set of unlabeled data in our newly constructed weighted graph.

The energy $E(y) = \frac{1}{2} \sum_{i,j} w_{i,j}(y_i - y_j)^2$, and seek to minimize energy of our graph. Hence, we assume that nearby points on the graph should have the same labels. Let $D = \text{diag}(d_i)$, with $d_i = \sum_j w_{ij}$, and the combinatorial Laplacian is the $n \times n$ matrix $\Delta = D - W$, where W is the weight matrix. We can rewrite the energy function as $E(y) = y^t \Delta y$. The Gaussian random field is $p(y) = \frac{1}{Z_\beta} \exp(-\beta E(y))$.

Hence, we turn the binary classification problem into a regression over $[0, 1]$. The probability distribution over unlabeled nodes is Gaussian with covariance matrix $\frac{1}{\beta} \Delta_{uu}^{-1}$. Δ_{uu} is the submatrix of Δ corresponding to unlabeled data. As it turns out, the function f that minimizes the energy function $E(y)$ is harmonic and satisfies $\Delta f = 0$, and $f(j) = \frac{1}{d_j} \sum_{i \sim j} w_{ij} f(i)$, which means it is simply the mean of its neighbours.

Δ can be broken down into block matrix form by Δ_{ll}, Δ_{lu}, Δ_{ul}, Δuu, and $f = [f_l \;\; f_u]^t$, where f_l is simply the true labels on the labeled data set. It turns out that $f_u = -\Delta_{uu}^{-1} \Delta_{ul} f_l$, and $y_u \sim \mathcal{N}(f_u, \Delta_{uu}^{=1})$. Bayes classifier would predict the label of node i as class 1 if $f(i) > 0.5$ and 0 else.

5.2 Active Learning

The risk $\mathcal{R}(f) = \sum_{i=1}^{n} \sum_{y_i = 0,1} [sgn(f_i) \neq y_i] p^*(y_i | L)$, where p^* is the true distribution. We do not have access to this, so the authors state that under this model, an appropriate approximation to the risk:

$$\begin{aligned} \hat{\mathcal{R}}(f) &= \sum_{i=1}^{n} [sgn(f_i) \neq 0](1 - f_i) + [sgn(f_i) \neq 1] f_i \\ &= \sum_{i=1}^{n} min(f_i, 1 - f_i) \end{aligned}$$

After doing a round of AL, we will get a label, re-compute the Harmonic field, denoted $f^{+(x_k, y_k)}$, and the estimated risk will also change:

$$\hat{\mathcal{R}}(f^{+(x_k, y_k)}) = \sum_{i=1}^{n} min(f_i^{+(x_k, y_k)}, 1 - f_i^{+(x_k, y_k)})$$

Since before we do a round of AL we do not know what label we receive, we can compute the expected risk:

$$\hat{\mathcal{R}}(f^{+x_k}) = (1 - f_k)\hat{\mathcal{R}}(f^{+(x_k,0)}) + f_k\hat{\mathcal{R}}(f^{+,(x_k,1)})$$

The AL algorithm selects the data point x_k that minimizes the expected Risk. In order to do this, $f^{+(x_k,y_k)}$ needs to be computed after adding the oracle labeled (x_k, y_k). This involves re-trainining the dataset, which is expensive. The authors note that this can be computed cheaply in closed form, recalling that $f_u = -\Delta_{uu}^{-1}\Delta_{ul}f_l$. They show in the appendix that $f_u^{+(x_k,y_k)}$ can be computed:

$$f_u^{+(x_k,y_k)} = f_u + (y_k - f_k)\frac{(\Delta_{uu}^{-1})_k}{(\Delta_{uu}^{-1})_{kk}}$$

where $(\Delta_{uu}^{-1})_k$ is the k-th column of the inverse Laplacian on unlabeled data, and $(\Delta_{uu}^{-1})_{kk}$ is the k-th diagonal element of the same matrix.

The authors do some experiments on synthetic data, and on 20 newsgroup (text) data. They explain how they generate features, and the graph (similarity metric). They compare their querying strategy with some other baselines. They also do experiments on handwritten digit dataset

6 Setting up an Active Learning experiment

- Given $\mathcal{D} = \mathcal{L} \cup \mathcal{U}$.
- If we start with absolutely no labels, select a random sample for labeling.
- Step 1: Meet with subject matter expert (label expert) to determine labeling budget.
  - How many total samples can they label? This will be broken down into how many batches?
  - How many samples per batch? Smaller = better.
  - Better to lots of small batches than to do a few larger batches.
- Split $\mathcal{L}$ into Train/Val/Test split
- Pick your base supervised learning algorithm
- Pick AL strategy(ies) + AL based active learning.
- For each batch: do active learning on $\mathcal{U}$, then update $\mathcal{L}$,$\mathcal{U}$.

6.1 No free lunch

How to pick the right AL strategy? Obviously computational time and size of your unlabeled dataset may play a role in this. However, given that you have selected an AL strategy, you run through the entire process and end up with your labeled data set and trained classifer, how do you know if you picked the right AL strategy? How do you know if this is better than random sampling?

Some potential solutions include: use multiple strategies simultaneously (with batch size of B, pick K strategies and each strategy selects $\frac{B}{K}$ data points (take into account overlap). Another more principled strategy is to use multi-armed bandits (k-armed bandits, stochastic bandits, etc). This form of online optimization is similar in nature to *follow the leader* approaches.

6.2 Multi Armed Bandits

Bandits generalize the notion of making a choice among k possible choices. Upon selecting a choice, some utility signal is returned to the agent. The agent uses statistics on the utility returned from each bandit, and the goal is to select the best bandit, being the one with highest expected utility. More formally,

- $\exists\ k < \infty$ arms (choices)
- Arm i has unknown reward distribution $\mathcal{R}_i$
- The utility/reward received at time t from arm i: $r_{i,t} \sim \mathcal{R}_i$, and $\mathbb{E}[r_{i,t}] = \mu_i$, $\forall t$.
- $\exists\ \mu^* = \underset{j \in \{1,2,\ldots,k\}}{max} \mu_j$
- Reward gap: $\delta_j := \mu^* - \mu_i$ is the expected regret from taking arm i, instead of taking the optimal arm.
- Number of times arm i has been taken up until time t: $N_i(t)$
- Empirical mean reward of arm i after time t $\hat{mu}(i)$
- Pseudo regret after time T, $R(T) = T\mu^* - \mathbb{E}[\sum_{t=1}^{T} r_{i,t}]$

The Upper Confidence Bound (UCB) algorithm provides a theoretically justified approach to solving stochastic bandit problems. The algorithm uses the UCB objective function:

$$UBC(t) := \underset{i \in \{1,2,\ldots,k\}}{arg\ max}\ \hat{\mu}_i + c\sqrt{\frac{\log(t)}{N_i}}$$

The algorithm begins by first trying each arm once, thus we have a running mean of the rewards/utility for each of our options, and we have $N_i = 1$ for all i arms. Then, for each other iteration we simply take the arm that maximizes UCB(t), update the running mean, and the counting parameter N_i.

How does it work? The UCB algorithm balances the exploration-exploitation problem to sequential decision processes. If the UCB algorithm only selected the arm that maximizes the empirical mean, then after trying each arm once, no further exploration would take place. Such an exploitation strategy is bad if rewards are stochastic. The second term in UCB enforces infinite exploration (in the limit). $c \geq 1$ is a hyper-parameter, and the larger it is, the more often the arms with lower empirical mean will be taken. Concentrating on the fraction in the square root, we have that at any given time t, no matter which arm we are computing the UCB(t) value for, $\log(t)$ will be the same. However, the arms that have been historically selected more often (e.g. has higher running mean reward), the value N_i will be larger, thus making the second UCB term smaller. Oppositely, those actions with small running mean will be selected less often, hence N_i will stay small, while $\log(t)$ slowly grows larger. Taken together, those arms that have a lower running mean will eventually be sampled, over and over. Hence, each arm, no matter how bad it is, will be sampled "every now and then", but the best performing arms will be sampled the most.

How does this apply to AL? We can consider k active learning strategies, and view the reward as the net change in model accuracy (on some hold out validation set, for example), and sample the active learning strategies according to a bandit algorithm such as UCB. In this way, we empirically learn which active learning strategy is best. Moreover, some AL strategies are "fast starters" while others are "late bloomers", and bandits give us a principled approach that could learn which AL strategy to use, and when.