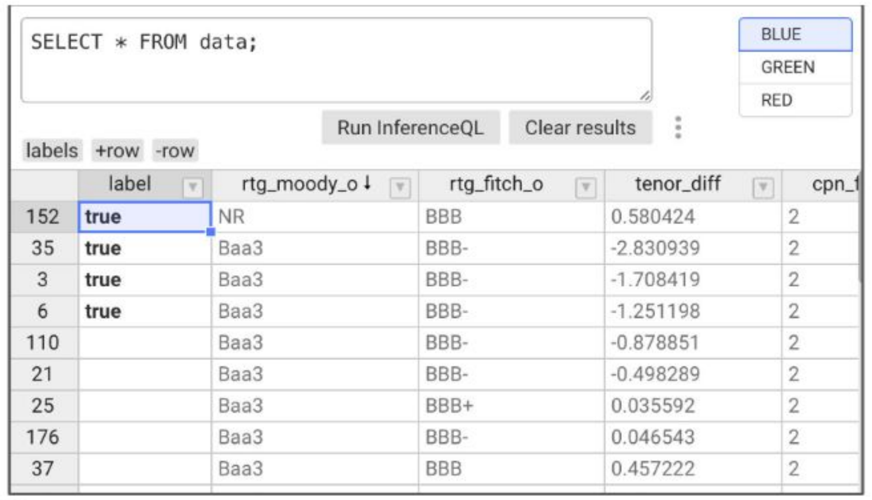
Probabilistic Programming for Bond Trading

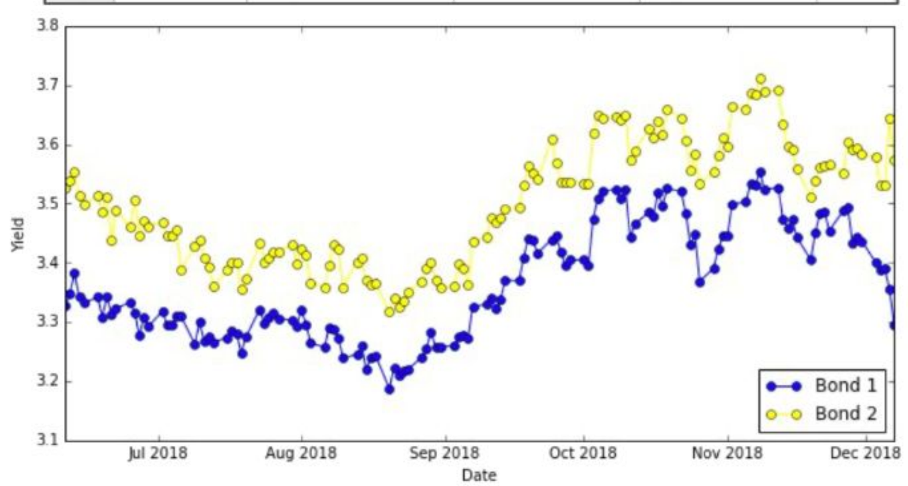
Veronica Weiner*[1, 2], Jameson Quinn*[1, 3], Harish Tella[1], Vikash Mansinghka[1]

1: MIT, 2: Probabilistic Computing Associates, 3: Jameson Quinn

- Machine-assisted bond trading is a challenging problem that could potentially be solved more effectively via probabilistic programming (PP) than standard machine learning (ML) techniques.
- We demonstrate a query for few-shot-learning-based search of bond trades implemented in a PP system. Our prototype uses an ML architecture combining domain-speci!c feature engineering, CrossCat modeling [1], few-shot learning queries [2] implemented via InferenceQL [3], and other query types (including CrossCat-based measures of bond similarity) implemented via BayesDB [3].
- Additionally, we introduce a novel algorithm for active learning implemented in a PP system that can help traders find bonds that match their chosen strategy.
- Initial experimental results are provided with simulated data to show this algorithm has the potential to increase search efficiency compared with non-active alternatives.

References: [1] Mansinghka V. et al, JMLR, 2016. [2] Charcut, N, MIT Thesis, 2020. [3] Schaeclte U. et al, PROBPROG, 2020 [4] Saad, F. et al, AISTATS, 2017.





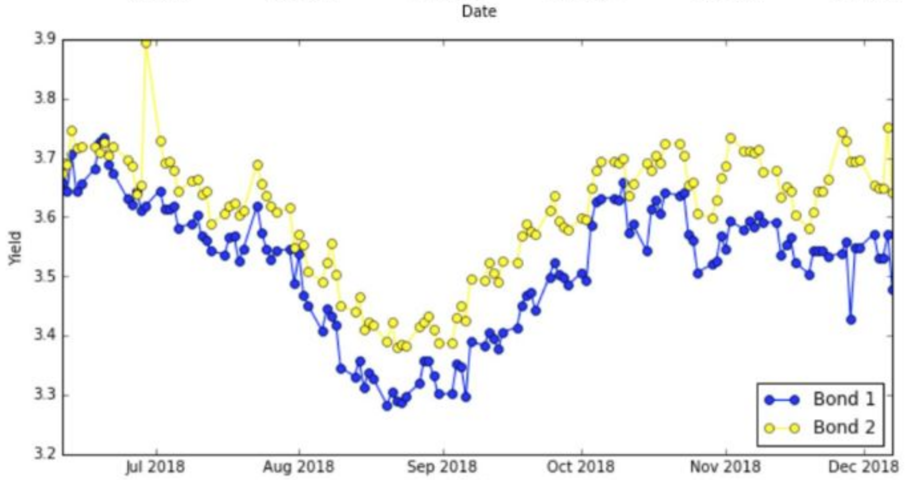


Figure 1. Results from machine-assisted bond trading. (top) Interface to collect labels from bond traders and execute probabilistic queries: a JavaScript spreadsheet in the probabilistic programming system (PPS) InferenceQL. (middle) A labeled bond trade from a set of trades labeled as interesting, showing increased yield in the recent time period. (bottom) The top bond trade from a search query to recommend trades based on a small number of provided labels.

```
function view_1() {
  var cluster_id = sample_categorical({"1": 0.00, "2": 0.00, "3": 0.08, /* And 24 others */});

  if (cluster_id == "1") {
    return {
      "tenor_n": sample_gaussian({"mu": 3.56, "sigma": 1.09}),
      "tenor_diff": sample_gaussian({"mu": 0.85, "sigma": 0.69}),
      "gspread_diff": sample_gaussian({"mu": 13.49, "sigma": 4.63}),
    };
  /* Some clusters omitted. */
  } else if (cluster_id == "27") {
    return {/* Samples */};
  }
}

function view_2() {
  var cluster_id = sample_categorical({"1": 0.22, "2": 0.10, "3": 0.22, /* And 4 others */});

  if (cluster_id == "1") {
    return {
      "rtg_moody_o": sample_categorical({"Aa3": 0.02, "Baa1": 0.20, "Aaa": 0.02, /* And 8 others */}),
      "gspread_o": sample_gaussian({"mu": 142.45, "sigma": 20.27}),
      "oas_n": sample_gaussian({"mu": 151.49, "sigma": 18.62}),
      "rtg_fitch_o": sample_categorical({"BBB": 0.07, "A-": 0.04, "BBB+": 0.05, /* And 8 others */}),
      /* Some columns omitted. */
    };
  /* Some clusters omitted. */
  } else if (cluster_id == "7") {
    return {/* Samples */};
  }
}

function view_3() {
  var cluster_id = sample_categorical({"1": 1.00});

  if (cluster_id == "1") {
    return {
      "structure_n": sample_categorical({"SENIOR": 0.96, "SRBN": 0.02, "SECURED": 0.02})
    };
  }
}

function model() {
  var view_1_samples = view_1();
  var view_2_samples = view_2();
  var view_3_samples = view_3();

  return {...view_1_samples, ...view_2_samples, ...view_3_samples};
}
```

Figure 2. A probabilistic program learned from bond schedules. This JavaScript source code generates a set of variables that characterize a bond at a moment in time. Repeated invocations generate a synthetic population of bonds. This program was learned from real bonds data, as in [Saad et al. 2019], by (i) modeling the data using CrossCat, a hierarchical Bayesian nonparametric model for multivariate data; (ii) truncating the CrossCat model; and (iii) compiling to Javascript. Once such a model is available, query functionality is provided by both Python and JavaScript.

Require: `Ensemble` probabilistic program G_0, Data table V
Note that G_0 is a sum of N separate `Subprograms` M_i^V, with associated importance weights p_i^V. If these were generated by MCMC, the weights will all be equal.
Require: sparse label set $\mathcal{Y}$

$(M^{V+\mathcal{Y}}, p^{V+\mathcal{Y}}) = \text{EMPTY-ENSEMBLE}(N)$
for $i \in \{1..N\}$ **do**
  $(M_i^{V+\mathcal{Y}}, p_i^{V+\mathcal{Y}}) = \text{MCMC-UPDATE-MODEL-AND-WEIGHTS}(M_i^V, p_i^V, \mathcal{Y})$ ▷ Incorporate $\mathcal{Y}$ into `Subprogram` i
end for
$s = \text{ZERO-VECTOR}(V.rows)$ ▷ Initialize vector of predictive entropy by row
for row $r \in V.rows$ **do**
  $\vec{x}_{[r,v]} = r.data$ ▷ Get the row data associated with r
  $s_{temp} = 0$ ▷ Accumulator for total weight across `Subprograms`
  $p_{label} = 0$ ▷ Accumulator for label probability across `Subprograms`
  for label $l \in \{\texttt{true}, \texttt{false}\}$ **do** ▷ Iterate over possible labels
    for $i \in \{1..N\}$ **do** ▷ Iterate over `Subprograms` in the updated `Ensemble`
      $p^{V+\mathcal{Y}+y} = \text{EMPTY-VECTOR}(N)$ ▷ Vector to hold expected posterior weights of `Subprograms`
      $p_{label} = p_{label} + p_i^{V+\mathcal{Y}} \text{CONDITIONAL-PROB}(\{Y_r = l\} | V, M_i^{V+\mathcal{Y}})$ ▷ accumulate probability of label l
      $p_i^{V+\mathcal{Y}+y} = p^{V+\mathcal{Y}+y} \exp(\text{LOGPDF}(M_i^{V+\mathcal{Y}}, col{:}\, Y = l, \vec{x}_{[r,v]})$ ▷ Weight for `Subprogram` i conditional on l
      $s_{temp} = s_{temp} - p_i^{V+\mathcal{Y}+y} \log(p_i^{V+\mathcal{Y}+y})$ ▷ Accumulate unnormalized entropy including label
    end for
    $s_r = s_r + p_{label} \left(\frac{s_{temp}}{\sum_i p_i^{V+\mathcal{Y}+y}} + log(\sum_i p_i^{V+\mathcal{Y}+y}) \right)$ ▷ Normalize entropy and incorporate its expectation into s_r
  end for
end for
return s ▷ List of expected posterior ensemble-level entropy for each row. Lower is better for labeling next.

Figure 3. (top) Active learning algorithm: Pseudocode for using an ensemble of probabilistic programs to rank rows with missing data for a label column.

Figure 4. (right) Active learning outperforms alternatives. Small gains in accuracy, or (equivalently) small reductions in the amount of expert labeling needed to get a given level of accuracy, could make big differences for companies using Active Few Shot Learning to inform buy and sell decisions.