universität innsbruck

Machine Learning for Theorem Proving

Lecture 3 (VU)

Cezary Kaliszyk

Overview

Last Lecture

- Machine Learning Overview
- Machine Learning Problems
- Lemma selection

Today

- k-nearest neighbours
- naive Bayes classifier

Premise Selection: Major methods tried

- Syntactic methods
  - Neighbours of the conjecture using various metrics
  - Most successful: Recursive methods. SInE and MePo.
- Naive Bayes classifier and k-Nearest Neighbours
- Linear / Logistic Regression
  - Needs feature and theorem space reduction
  - Kernel-based multi-output ranking
- Decision Trees (Random Forests)
- Neural Networks
  - Winnow, Perceptron
  - Deep learning

[*SNoW,MaLARea*]

Statistical ML Algorithms for Premise Selection

- **k-Nearest Neighbours:**
  - finds a fixed number (*k*) of proved facts nearest to the conjecture *c*
  - weight the dependencies each such fact *f* by the distance between *f* and *c*
  - relevance is the sum of weights across the *k* nearest neighbors
- **Naive Bayes:**
  - computes the probability of *f* being needed to prove *c*
  - based on the previous use of *f* in proving conjectures similar to *c*
  - assumes independence of features to apply the Bayes theorem
- **MePo:** (*Me*ng–*P*auls*o*n relevance filter)
  - score of a fact is $r/(r+i)$, where r is the number of relevant features and i the number of irrelevant features
  - iteratively select all top-scoring facts and add their features to the set of relevant features.
- **Combination**
  Combining multiple of the above classifiers in various ways
  (average ratings, or more complex ensemble techniques)
  can give even stronger selectors.

Adapting Basic Machine Learning Algorithms

Basic k-Nearest Neighbours:

- Given a set of training examples and a sample to evaluate
- Find *k* nearest neighbors to the sample among the training examples
- Return their average (or weighted average)

Adapting Basic Machine Learning Algorithms

Basic k-Nearest Neighbours:

- Given a set of training examples and a sample to evaluate
- Find *k* nearest neighbors to the sample among the training examples
- Return their average (or weighted average)

kNN Properties

- Very easy to implement
- Almost no learning (build a simple index)

Adapting Basic Machine Learning Algorithms

Basic k-Nearest Neighbours:

- Given a set of training examples and a sample to evaluate
- Find *k* nearest neighbors to the sample among the training examples
- Return their average (or weighted average)

kNN Properties

- Very easy to implement
- Almost no learning (build a simple index)

What does this mean for premise selection?

- find *k* proved facts nearest to the conjecture *c*
- weight dependency fact *f* by the distance between *f* and *c*
- relevance is the sum of weights across the *k* neighbors

k-NN (1/3)

We start with the basic definition of distance between two facts (similarity) characterized by binary features. We then add feature weight and add a scaling factor for these weights:

Definition: Distance of two facts (similarity)

$$s(a,b) = \sum_{f \in F(a) \cap F(b)} 1$$

k-NN (1/3)

We start with the basic definition of distance between two facts (similarity) characterized by binary features. We then add feature weight and add a scaling factor for these weights:

Definition: Distance of two facts (similarity)

$$s(a, b) = \sum_{f \in F(a) \cap F(b)} w(f)$$

k-NN (1/3)

We start with the basic definition of distance between two facts (similarity) characterized by binary features. We then add feature weight and add a scaling factor for these weights:

Definition: Distance of two facts (similarity)

$$s(a,b) = \sum_{f \in F(a) \cap F(b)} w(f)^{\tau_1}$$

k-NN (1/3)

We start with the basic definition of distance between two facts (similarity) characterized by binary features. We then add feature weight and add a scaling factor for these weights:

Definition: Distance of two facts (similarity)

$$s(a,b) = \sum_{f \in F(a) \cap F(b)} w(f)^{\tau_1}$$

To estimate the relevance we start with the similarities. We then realize that if a proof relies on more dependencies, each of them is less valuable to we divide by the number of dependencies. We additionally add a factor for the dependencies themselves.

Relevance of fact *a* for goal *g*

$$\left(\sum_{b \in N | a \in D(b)} \frac{s(b,g)}{|D(b)|} \right)$$

k-NN (1/3)

We start with the basic definition of distance between two facts (similarity) characterized by binary features. We then add feature weight and add a scaling factor for these weights:

Definition: Distance of two facts (similarity)

$$s(a,b) = \sum_{f \in F(a) \cap F(b)} w(f)^{\tau_1}$$

To estimate the relevance we start with the similarities. We then realize that if a proof relies on more dependencies, each of them is less valuable to we divide by the number of dependencies. We additionally add a factor for the dependencies themselves.

Relevance of fact *a* for goal *g*

$$\left(\sum_{b \in N \mid a \in D(b)} \frac{s(b,g)}{|D(b)|} \right) + \begin{cases} s(a,g) & \text{if } a \in N \\ 0 & \text{otherwise} \end{cases}$$

k-NN (1/3)

We start with the basic definition of distance between two facts (similarity) characterized by binary features. We then add feature weight and add a scaling factor for these weights:

Definition: Distance of two facts (similarity)

$$s(a, b) = \sum_{f \in F(a) \cap F(b)} w(f)^{\tau_1}$$

To estimate the relevance we start with the similarities. We then realize that if a proof relies on more dependencies, each of them is less valuable to we divide by the number of dependencies. We additionally add a factor for the dependencies themselves.

Relevance of fact *a* for goal *g*

$$\left(\tau_2 \sum_{b \in N | a \in D(b)} \frac{s(b, g)}{|D(b)|} \right) + \begin{cases} s(a, g) & \text{if } a \in N \\ 0 & \text{otherwise} \end{cases}$$

k-NN (2/3)

- Ok, we have adapted k-Nearest Neighbors to premise selection, but is this already good enough? What k to choose?
- In 2013 in an experimental evaluation it came out that values of k between 40 and 800 are good and multiple values are complementary.
- But actually the complementarity only says that we need different k for different cases.
- Can we thus choose *k* automatically?
- On the next slide we have the code for k-NN from multiple state-of-art ITPs, that is currently used in tools. As the core is rather small, you can study it, see how weights are used and how k is adjusted.

k-NN (3/3)

```
let knn_eval csyms (sym_ths, sym_wght) deps maxth no_adv =
  let neighbours = Array.init maxth (fun j -> (j, 0.)) in
  let ans = Array.copy neighbours in

  (* for each symbol, increase the importance of the theorems
     which contain the symbol by a given symbol weight *)
  List.iter (fun sym ->
      let ths = sym_ths sym and weight = sym_wght sym in
      List.iter (fun th ->
          if th < maxth then map_snd neighbours th ((+.) (weight ** 6.0))) ths) csyms;

  Array.fast_sort sortfun neighbours;

  let no_recommends = ref 0 in
  let add_ans k i o =
    if snd (ans.(i)) <= 0. then begin
      incr no_recommends;
      map_snd ans i (fun _ -> float_of_int (age k) +. o))
    end else map_snd ans i ((+.) o) in

  (* Additionally stop when given no_recommends reached *)
  Array.iteri (fun k (nn, o) ->
    add_ans k nn o;
    let ds = deps nn in
    let ol = 2.7 *. o /. (float_of_int (List.length ds)) in
    List.iter (fun d -> if d < maxth then add_ans k d ol) ds;
  ) neighbours;

  Array.fast_sort sortfun ans;
```

Naive Bayes

Given a known fact f and a goal to prove g we try to estimate the probability based on the features of the goal $f_1, \ldots, f_n$

Naive Bayes

Given a known fact f and a goal to prove g we try to estimate the probability based on the features of the goal $f_1, ..., f_n$

$$\begin{aligned} & P(f \text{ is relevant for proving } g) \\ = \; & P(f \text{ is relevant} \mid g\text{'s features}) \\ = \; & P(f \text{ is relevant} \mid f_1, \ldots, f_n) \\ \propto \; & P(f \text{ is relevant}) \Pi_{i=1}^{n} P(f_i \mid f \text{ is relevant}) \\ \propto \; & \#f \text{ is a proof dependency} \cdot \Pi_{i=1}^{n} \frac{\#f_i \text{ appears when } f \text{ is a proof dependency}}{\#f \text{ is a proof dependency}} \end{aligned}$$

Naive Bayes

Given a known fact f and a goal to prove g we try to estimate the probability based on the features of the goal $f_1, ..., f_n$

$$
\begin{aligned}
& P(f \text{ is relevant for proving } g) \\
= \; & P(f \text{ is relevant} \mid g\text{'s features}) \\
= \; & P(f \text{ is relevant} \mid f_1, \ldots, f_n) \\
\propto \; & P(f \text{ is relevant}) \Pi_{i=1}^{n} P(f_i \mid f \text{ is relevant}) \\
\propto \; & \#f \text{ is a proof dependency} \cdot \Pi_{i=1}^{n} \frac{\#f_i \text{ appears when } f \text{ is a proof dependency}}{\#f \text{ is a proof dependency}}
\end{aligned}
$$

The number of times f has been a proof dependency and the numbers of times particular features were present in the goal when f was a dependency can be easily cashed
in a table and a map.

Naive Bayes: adaptation to premise selection

This is already ok, but we can improve upon it looking at the features that are not present. But there are too many of them to take all into account. So we look at not present, but related features:

Naive Bayes: adaptation to premise selection

This is already ok, but we can improve upon it looking at the features that are not present. But there are too many of them to take all into account. So we look at not present, but related features:

extended features $\overline{F}(a)$ of a fact a

features of a and of the facts that were proved using a

Naive Bayes: adaptation to premise selection

This is already ok, but we can improve upon it looking at the features that are not present. But there are too many of them to take all into account. So we look at not present, but related features:

extended features $\overline{F}(a)$ of a fact a

features of a and of the facts that were proved using a

(This extension of features has been performed for one iteration only. In principle it could be repeated but experimentally gives little advantage)

Naive Bayes: adaptation to premise selection

This is already ok, but we can improve upon it looking at the features that are not present. But there are too many of them to take all into account. So we look at not present, but related features:

extended features $\overline{F}(a)$ of a fact a

features of a and of the facts that were proved using a

(This extension of features has been performed for one iteration only. In principle it could be repeated but experimentally gives little advantage)

More precise estimation of the relevance of ϕ to prove γ:

$$
\begin{aligned}
&P(a \text{ is used in } \psi\text{'s proof}) \\
&\quad \cdot \prod_{f \in F(\gamma) \cap \overline{F}(a)} P(\psi \text{ has feature } f \mid a \text{ is used in } \psi\text{'s proof}) \\
&\quad \cdot \prod_{f \in F(\gamma) - \overline{F}(a)} P(\psi \text{ has feature } f \mid a \text{ isn't used in } \psi\text{'s proof}) \\
&\quad \cdot \prod_{f \in \overline{F}(a) - F(\gamma)} P(\psi \text{ doesn't have } f \mid a \text{ used in } \psi\text{'s proof})
\end{aligned}
$$

All these probabilities can be computed efficiently

Update two functions (tables):

- $t(a)$: number of times a fact a was dependency
- $s(a,f)$:
  number of times a fact a was dependency of a fact described by feature f

Then:

$$P(a \text{ is used in a proof of (any) } \psi) = \frac{t(a)}{K}$$

$$P\big(\psi \text{ has feature } f \mid a \text{ is used in } \psi\text{'s proof}\big) = \frac{s(a,f)}{t(a)}$$

$$P\big(\psi \text{ does not have feature } f \mid a \text{ is used in } \psi\text{'s proof}\big) = 1 - \frac{s(a,f)}{t(a)}$$

$$\approx 1 - \frac{s(a,f)-1}{t(a)}$$

Naive Bayes “in practice” (1/2)

Naive Bayes classifier is again so simple, that we can study the actual C++ code of the evaluation of a score of a particular fact for a goal.

Again note sets, hashtables, and a number of ad-hoc parameter optimizations.

The addition of all these optimizations improves the performance from about 20% to about 40%, so these are significant.

Naive Bayes “in practice” (2/2)

```
double NaiveBayes::score(sample_t i, set<feature_t> symh) const {
  // number of times current theorem was used as dependency
  const long n      = tfreq[i];
  const auto sfreqh = sfreq[i];

  double s = 30 * log(n);

  for (const auto sv : sfreqh) {
    // sv.first ranges over all features of theorems depending on i
    // sv.second is the number of times sv.first appears among theorems
    // depending on i
    double sfreqv = sv.second;

    // if sv.first exists in query features
    if (symh.erase(sv.first) == 1)
      s += tfidf.get(sv.first) * log (5 * sfreqv / n);
    else
      s += tfidf.get(sv.first) * 0.2 * log (1 + (1 - sfreqv) / n);
  }

  // for all query features that did not appear in features of dependencies
  // of current theorem
  for (const auto f : symh) s -= tfidf.get(f) * 18;

  return s;
```

Additional Literature (not required)

The first paper has a more detailed description of k-NN and Naive Bayes.

Jasmin Blanchette, David Greenaway, Cezary Kaliszyk, Daniel Kühlwein, and Josef Urban.
A learning-based fact selector for Isabelle/HOL.
J. Autom. Reasoning, 57(3):219–244, 2016.

Summary

This Lecture

- Lemma selection
- k-nearest neighbours
- naive Bayes classifier

Next

- Statistical methods
- decision trees
- random forests
- ML-evaluation
- Theorem proving foundations
- Deep Learning for premise selection

Work Here / Homework

Find k nearest premises

- `http://cl-informatik.uibk.ac.at/teaching/ws23/mltp/mltp.tgz`

Propose ways to evaluate such an algorithm

- Using some metrics
- With an ATP

MePo (Isabelle’s Meng Paulson filter)

- Find, read, and be able to present the code of the relevance filter (hint: Isabelle’s source code, file with ‘mepo’ in its name...)