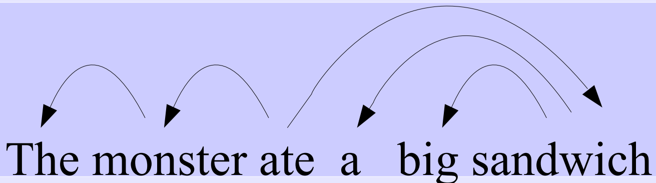
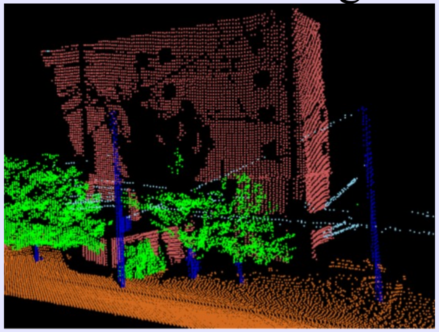
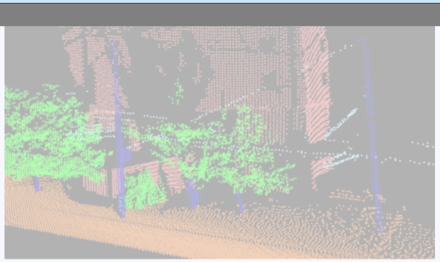
What is structured prediction?

Task	Input	Output
Machine Translation	Ces deux principes se tiennent à la croisée de la philosophie, de la politique, de l'économie, de la sociologie et du droit.	Both principles lie at the crossroads of philosophy, politics, economics, sociology, and law.
Sequence Labeling	The monster ate a big sandwich	Det Noun VerbDetAdj Noun The monster ate a big sandwich
Syntactic Analysis	The monster ate a big sandwich	
3d point cloud classification	3d range scan data	
...many more...		

What is structured prediction?

Task	Input	Output
Machine Tr	Ces deux principes se tiennent à	Both principles lie at the
Se La		
Sy An		h
3d classification		
...many more...		

Structured Prediction Haiku

A joint prediction
Across a single input
Loss measured jointly

We want to minimize...

- **Programming complexity.** Most structured predictions are *not* addressed using structured learning because of programming complexity.
- **Test loss.** If it doesn't work, game over.
- **Training time.** Debug/develop productivity, hyperparameter tuning, maximum data input.
- **Test time.** Application efficiency.

Programming complexity

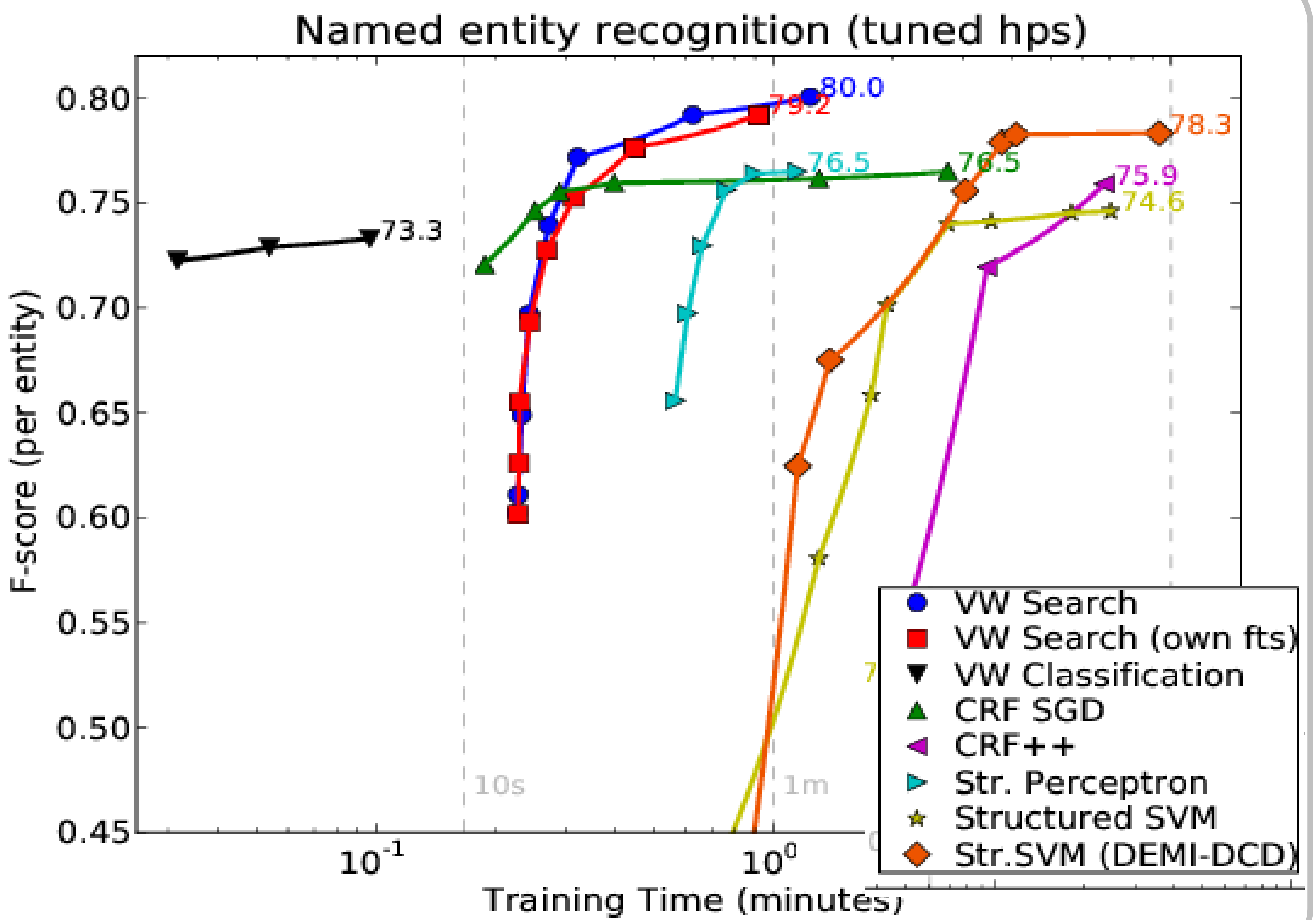
search_sequencetask.cc

File Edit Options Buffers Tools C++ YASnippet Development Cscope Help

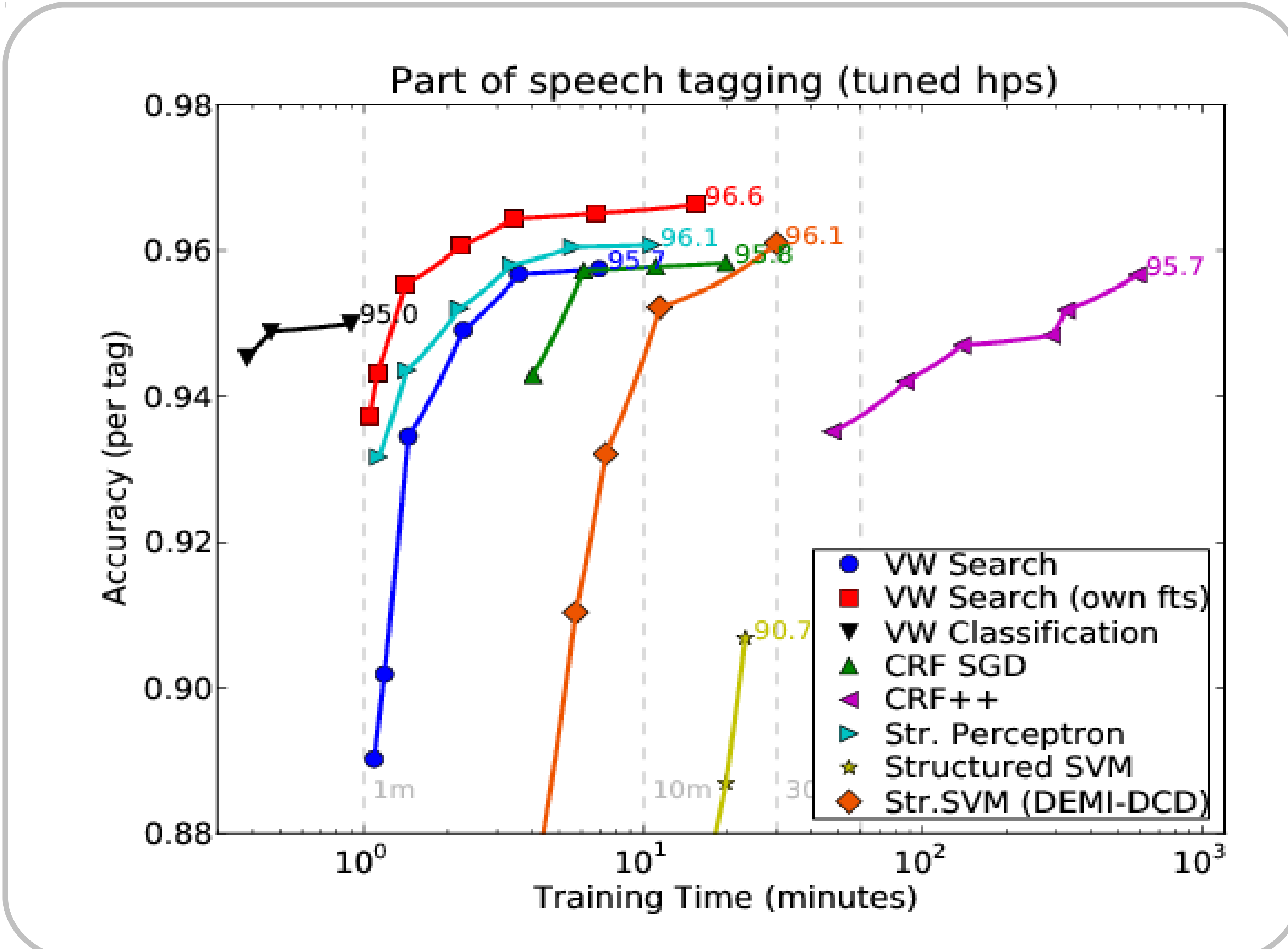
```
namespace SequenceTask {
  void initialize(Search::search& sch, size_t& num_actions, po::variables_map& vm) {
    sch.set_options( Search::AUTO_CONDITION_FEATURES  |
                     Search::AUTO_HAMMING_LOSS        |
                     Search::EXAMPLES_DONT_CHANGE     |
                     0);
  }

  void run(Search::search& sch, vector<example*>& ec) {
    for (int i=0; i<ec.size(); i++) {
      action oracle     = MULTICLASS::get_example_label(ec[i]);
      size_t prediction = Search::predictor(sch, i+1).set_input(*ec[i]).set_oracle(oracle)
          .set_condition_range(i, sch.get_history_length(), 'p').predict();

      if (sch.output().good())
        sch.output() << prediction << ' ';
    }
  }
}
```

-:**- search_sequencetask.cc   6% (34,0)   Git-master (C++/l BufFace AC yas Abbrev)

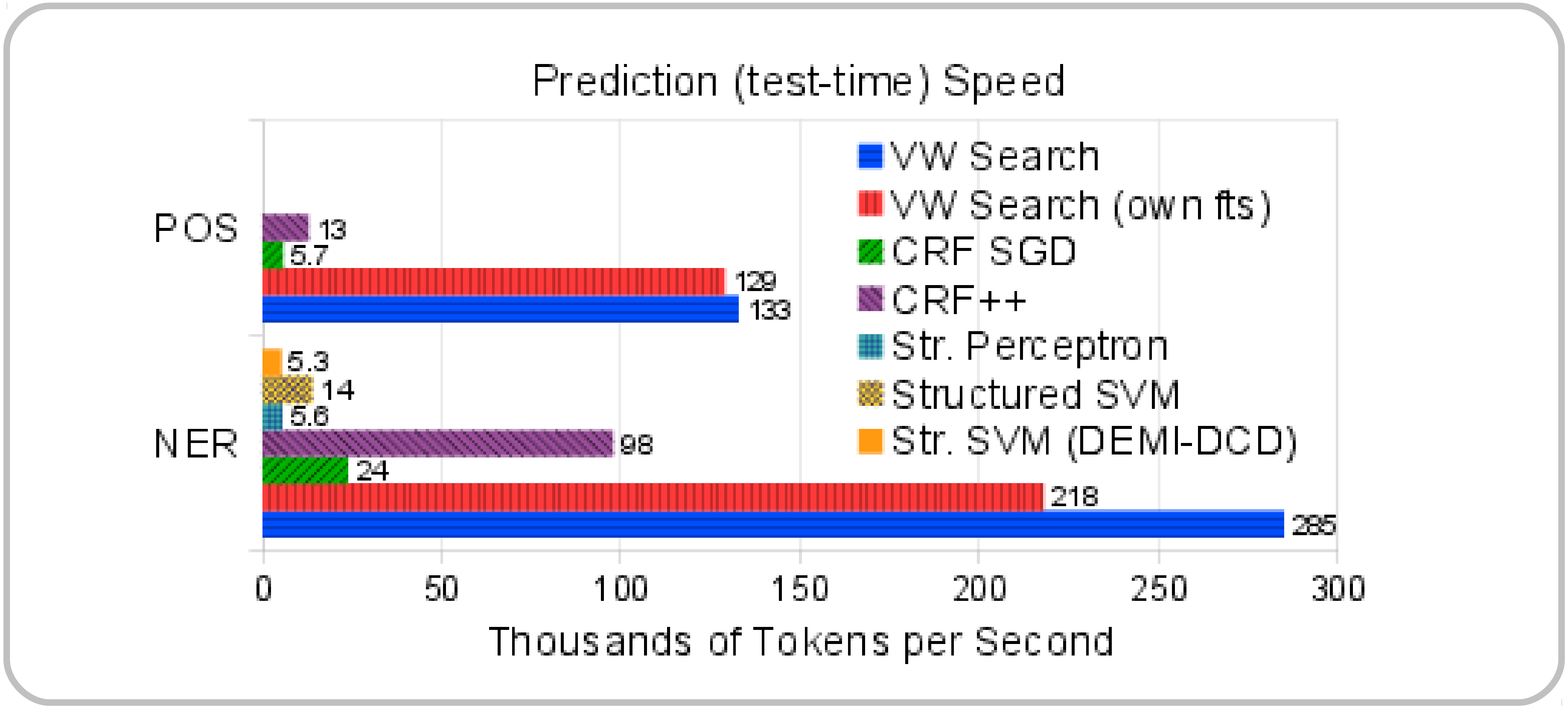
Training time versus test accuracy



Training time versus test accuracy



Test time speed



State of the art accuracy in....

- Part of speech tagging (1 million words)

vw:	6 lines of code	3 minutes to train
CRFsgd:	1068 lines	6 minutes
CRF++:	777 lines	hours

- Named entity recognition (200 thousand words)

vw:	30 lines of code	20 seconds to train
CRFsgd:		1 minute (subopt accuracy)
CRF++:		10 minutes (subopt accuracy)
SVM^{str}:	876 lines	30 minutes (subopt accuracy)

State of the art accuracy in....

- Part of speech tagging (1 million words)

wc:		3.2 seconds
vw:	6 lines of code	3 minutes to train
CRFsgd:	1068 lines	6 minutes
CRF++:	777 lines	hours

- Named entity recognition (200 thousand words)

wc:		0.8 seconds
vw:	30 lines of code	20 seconds to train
CRFsgd:		1 minute (subopt accuracy)
CRF++:		10 minutes (subopt accuracy)
SVM^{str}:	876 lines	30 minutes (subopt accuracy)

Command-line usage

```
% wget http://bilbo.cs.uiuc.edu/~kchang10/tmp/wsj.vw.zip

% unzip wsj.vw.zip

% vw -b 24 -d wsj.train.vw -c --search_task sequence \
   --search 45 --search_neighbor_features -1:w,1:w   \
   --affix -1w,+1w -f wsj.weights

<chat with your neighbor for 3 minutes>

% vw -t -i wsj.weights wsj.test.vw

<wait 0.15 seconds for 96.4% accuracy>
```

Python interface to VW

Library interface to VW (*not* a command line wrapper)

It is *actually* documented!!!

Allows you to write code like:

```
import pyvw
vw = pyvw.vw("--quiet")
ex1 = vw.example("1 |x a b |y c")
ex2 = vw.example({'x': ['a', ('b', 1.0)], \
                  'y': ['c']})
ex1.learn()
print ex2.predict()
```

iPython Notebook for Learning to Search

IP[y]: Notebook   Learning_to_Search Last Checkpoint: Oct 03 14:43 (autosaved)

File   Edit   View   Insert   Cell   Kernel   Help

Markdown   Cell Toolbar: None

The _run function executes the sequence of decisions on a given input. The input will be of whatever type our data is (so, in the above example, it will be a list of (label,word) pairs).

Here is a basic implementation of sequence labeling:

In [39]:

```
class SequenceLabeler(pyvw.SearchTask):
    def __init__(self, vw, sch, num_actions):
        pyvw.SearchTask.__init__(self, vw, sch, num_actions)
        sch.set_options( sch.AUTO_HAMMING_LOSS | sch.AUTO_CONDITION_FEATURES )

    def _run(self, sentence):
        output = []
        for n in range(len(sentence)):
            pos,word = sentence[n]
            with self.vw.example({'w': [word]}) as ex:
                pred = self.sch.predict(examples=ex, my_tag=n+1, oracle=pos, condition=(n,'p'))
                output.append(pred)
        return output
```

http://tinyurl.com/pyvwsearch

http://tinyurl.com/pyvwtalk