Vowpal Wabbit

http://hunch.net/~vw/

John Langford

Yahoo! Research

git clone git://github.com/JohnLangford/vowpal_wabbit.git
Why VW?

1. There **should exist** an open source **online** learning system.

2. **Online learning** $\Rightarrow$ **online optimization**, which is or competes with best practice for many learning algorithms.

3. VW is a **multitrick pony**, all useful, all orthogonally composable. [hashing, caching, parallelizing, feature crossing, features splitting, feature combining, etc...]

4. It’s **simple**. No strange dependencies, currently only 4092 lines of code.
On RCV1, training time = \(~3s\) [caching, pipelining]

On “large scale learning challenge” datasets \(< 10\) minutes [caching]

[ICML 2009] \(10^5\)-way personalize spam filter. [-q, hashing]

[UAI 2009] \(10^6\)-way conditional probability estimation. [library, hashing]

[ALT 2009] Log-time multiclass classification. [library, hashing]

[KDD 2009] Partial Label policy learning. [library, hashing]

[Rutgers grad] Gexample/day data feed. [–daemon]
The basic learning algorithm

Start with $\forall i : w_i = 0$, Repeatedly:

1. Get example $x \in (\infty, \infty)^\ast$.

2. Make prediction $\hat{y} = \frac{\sum_i w_i x_i}{\sqrt{|\{i : x_i \neq 0\}|}}$ clipped to interval $[0, 1]$.

3. Learn truth $y \in [0, 1]$ with importance $I$ or goto (1).

4. Update $w_i \leftarrow w_i + \frac{\eta^2(y - \hat{y})I}{\sqrt{|\{i : x_i \neq 0\}|}}$ and go to (1).
Input Format
Label [Importance] [Tag]|Namespace Feature ... |Namespace Feature ... ...

Namespace = String[:Float]
Feature = String[:Float]
Feature and Label are what you expect.

Importance is multiplier on learning rate.

Tag is an identifier for an example, echoed on example output.

Namespace is a mechanism for feature manipulation and grouping.
Valid input examples

1 | 13:3.96e-02 24:3.47e-02 69:4.62e-02

example_39|excuses the dog ate my homework

1 0.500000 example_39|excuses:0.1 the:0.01 dog ate my homework |teacher male white Bagnell AI ate breakfast
Example Input Options

[-d] [-data] <f> : Read examples from f. Multiple ⇒ use all

cat <f> | vw : read from stdin

–daemon : read from port 39524

–port <p> : read from port p

–multisource : Assemble examples piecemeal from multiple sources. For cluster parallelism.

–passes <n> : Number of passes over examples. Can’t multipass a noncached stream.

-c [–cache] : Use a cache (or create one if it doesn’t exist).

Example Output Options

Default diagnostic information:

Progressive Validation, Example Count, Label, Prediction, Feature Count


-r [-raw_predictions ] <ro> : File to output unnormalized prediction into.

-audit : Detailed information about feature_name: feature_index: feature_value: weight_value

-quiet : No default diagnostics
Playing with Options: Example Manipulation

-t [ -testonly ]: Don’t train, even if the label is there. Convenience Only.


Example: -q et

(= make an extra feature for every excuse feature and teacher feature)
Update Rule Options

- decay_learning_rate \( <d> \) \[ = \frac{1}{\sqrt{2}} \]
- initial_t \( <i> \) \[ = 1 \]
- power_t \( <p> \) \[ = 0 \]
- l \[ = 0.1 \]

\[
\eta_e = \frac{ld^{n-1}ip}{(i + \sum_{e'<e} i_{e'})p}
\]

Basic observation: there exists no one learning rate satisfying all uses.

Example: state tracking vs. online optimization.

- loss_function \{squared, log, hinge, quantile\} Switch loss function
Weight Options


-i [ -initial_regressor ] <ri> : Initial weight values. Multiple ⇒ average.

-f [ -final_regressor ] <rf> : File to store final weight values in.
Parallelization Options

-thread-bits \(<b>\) : Use \(2^b\) threads for multicore. Introduces some nondeterminism (floating point add order). Only useful with -q

-sendto \(<\text{host[:port]}>\) : Shard examples to host:port.

-predictto \(<\text{host[:port]}>\) : Send prediction to host:port. Use with -multisource

(demo)
“I have a better loss function”

1. Implement in loss_functions.cc.

2. Send a patch / github pull request.
“My online learning algorithm is better.”

1. Copy `{gd.cc, sender.cc, noop.cc}` to a new file and tweak.

2. Add flag to `parse_args.cc`

3. Implement flag in `vw.cc`

4. Send a patch / github pull request.
‘I want to solve cost sensitive partial label multitask multiclass problems.’

1. Copy `simple_label.cc` and tweak to parse and define label information.

2. Copy `gd.cc` and implement reduction algorithm. Use `offset_predict` and `offset_train` for hashing magic.

3. Add flag(s) to `parse_args.cc`.

4. Implement flag in `vw.cc`.

5. Send a patch / github pull request.
My Plans for Future Development

1. Finish scaling up. I want a kilonode program.


3. Other learning algorithms. Much good work to be done here.