Vowpal Wabbit 6.1

http://hunch.net/~vw/

John Langford, then Miroslav Dudik, then Alekh Agarwal

git clone
git://github.com/JohnLangford/vowpal_wabbit.git

Goals of the VW project

1. State of the art in scalable, fast, efficient Machine Learning. See Miro & Alekh parts. VW is (by far) the most scalable public linear learner, and plausibly the most scalable anywhere.
2. Support research into new ML algorithms. We ML researchers can deploy new efficient algorithms on an efficient platform efficiently.
3. Simplicity. No strange dependencies, currently only 7054 lines of code.
4. It just works. A package in debian & R. Otherwise, users just type "make", and get a working system. At least a half-dozen companies use VW. Favorite App: True Love @ Eharmony.

The Tutorial Plan

1. Baseline online linear algorithm
2. What goes wrong? And fixes
   - 2.1 Importance Aware Updates
   - 2.2 Adaptive updates
3. LBFGS: Miro's turn
4. Terascale Learning: Alekh's turn.
5. Common questions we don't have time to cover.
6. Active Learning: See Daniels's presentation last year.
7. LDA: See Matt's presentation last year.

Ask Questions!

Demonstration

time zcat rcv1.train.vw.gz| vw -c

The basic learning algorithm (classic)

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

1. Get example $x \in (\infty, \infty)^*$.
2. Make prediction $\hat{y} = \sum_i w_i x_i$ clipped to interval $[0, 1]$.
3. Learn truth $y \in [0, 1]$ with importance I or goto (1).
4. Update $w_i \leftarrow w_i + \eta 2(y - \hat{y}) I x_i$ and go to (1).

Input Format

Label [Importance] [Base] ['Tag] |Namespace Feature ... |Namespace Feature ... ... \n

Namespace = String[:Float]

Feature = String[:Float]

If String is an integer, that index is used, otherwise a hash function computes an index.

Feature and Label are what you expect.

Importance is multiplier on learning rate, default 1.

Base is a baseline prediction, default 0.

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 $\Rightarrow$ use all

cat <f> | vw : read from stdin

–daemon : read from port 26542

–port <p> : read from port p

–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).

–cache_file <fc> : Use the fc cache file. Multiple $\Rightarrow$ use all. Missing $\Rightarrow$ create. Multiple+missing $\Rightarrow$ concatenate

–compressed: gzip compress cache_file.

Example Output Options

Default diagnostic information:
Progressive Validation, Example Count, Label, Prediction, Feature Count

-p [–predictions] <po>: File to dump predictions into.

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

–sendto <host[:port]> : Send examples to host:port.

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

–quiet : No default diagnostics

Example Manipulation Options

-t [–testonly] : Don't train, even if the label is there.
-q [–quadratic] <ab>: Cross every feature in namespace a* with every feature in namespace b*.
Example: -q et (= extra feature for every excuse feature and teacher feature)
–ignore <a>: Remove a namespace and all features in it.
–noconstant: Remove the default constant feature.
–sort_features: Sort features for small cache files.
–ngram <N>: Generate N-grams on features. Incompatible with sort_features
–skips <S>: ...with S skips.
–hash all: hash even integer features.

Update Rule Options

–decay_learning_rate <d> [= 1]
–initial_t <i> [= 1]
–power_t <p> [= 0.5]
-l [–learning_rate] <l> [= 10]

$$\eta_e = \frac{l d^{n-1} i^p}{(i + \sum_{e' < e} i_{e'})^p}$$

Basic observation: there exists no one learning rate satisfying all uses.
Example: state tracking vs. online optimization.
–loss_function {squared,logistic,hinge,quantile,classic} Switch loss function

Weight Options

-b [–bit_precision] <b> [=18] : log(Number of weights). Too many features in example set⇒ collisions occur.

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

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

–readable_model <filename>: As -f, but in text.

–save_per_pass Save the model after every pass over data.

–random_weights <r>: make initial weights random. Particularly useful with LDA.

–initial_weight <iw>: Initial weight value

The Tutorial Plan

1. Baseline online linear algorithm
2. What goes wrong? And fixes
   2.1 Importance Aware Updates
   2.2 Adaptive updates
3. LBFGS: Miro's turn
4. Terascale Learning: Alekh's turn.
5. Common questions we don't have time to cover.
6. Active Learning: See Daniels's presentation last year.
7. LDA: See Matt's presentation last year.

Ask Questions!

Examples with large importance weights don't work!

Common case: class is imbalanced, so you downsample the common class and present the remainder with a compensating importance weight. (but there are many other examples)

Examples with large importance weights don't work!

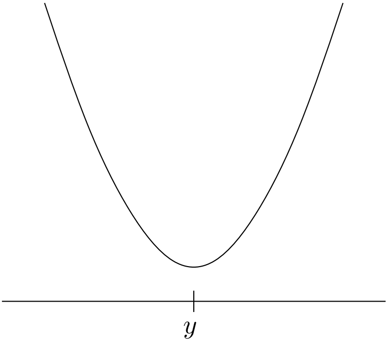
Common case: class is imbalanced, so you downsample the common class and present the remainder with a compensating importance weight. (but there are many other examples)

Actually, I lied. The preceeding update only happens for "–loss_function classic".
The update rule is really importance invariant [KL11], which helps substantially.

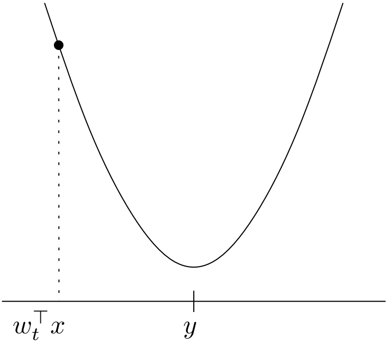
Principle

An example with importance weight h is equivalent to having the example h times in the dataset.

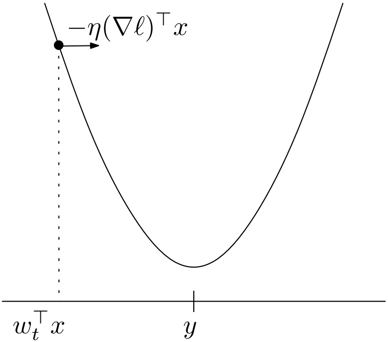
Learning with importance weights



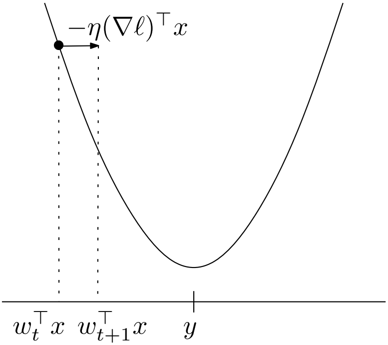
Learning with importance weights



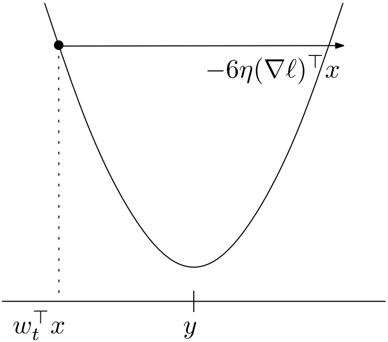
Learning with importance weights



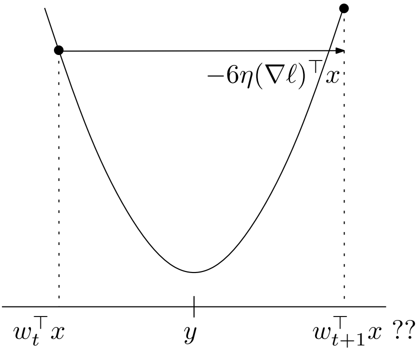
Learning with importance weights



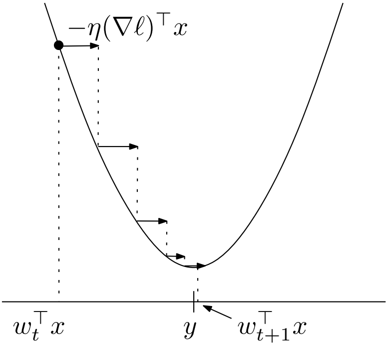
Learning with importance weights



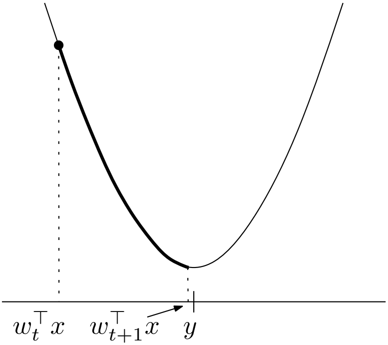
Learning with importance weights



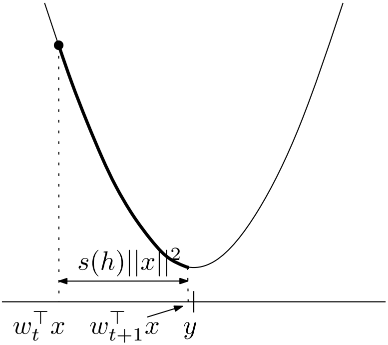
Learning with importance weights



Learning with importance weights



Learning with importance weights



What is $s(\cdot)$?

Take limit as update size goes to 0 but number of updates goes to ∞.

What is $s(\cdot)$?

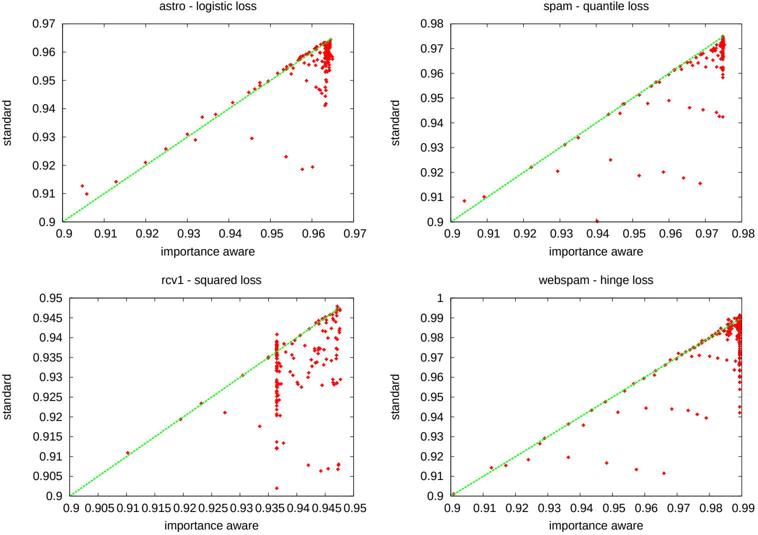
Take limit as update size goes to 0 but number of updates goes to ∞.

Surprise: simplifies to closed form.

Loss	$\ell(p, y)$	Update $s(h)$
Squared	$(y-p)^2$	$\frac{p-y}{x^\top x}\left(1-e^{-h\eta x^\top x}\right)$
Logistic	$\log(1+e^{-yp})$	$\frac{W(e^{h\eta x^\top x + yp + e^{yp}}) - h\eta x^\top x - e^{yp}}{yx^\top x}$
Hinge	$\max(0, 1-yp)$	$-y \min\left(h\eta, \frac{1-yp}{x^\top x}\right)$ for $y \in \{-1, 1\}$
τ-Quantile	$y > p: \quad \tau(y-p)$ $y \leq p: \quad (1-\tau)(p-y)$	$y > p: \quad -\tau \min(h\eta, \frac{y-p}{\tau x^\top x})$ $y \leq p: \quad (1-\tau)\min(h\eta, \frac{p-y}{(1-\tau)x^\top x})$

+ many others worked out. Similar in effect to "implicit gradient", but closed form.

Robust results for unweighted problems



It takes forever to converge!

It takes forever to converge!

Think like a physicist: Everything has units.
Let x_i be the base unit. Output $\langle w \cdot x \rangle$ has unit "probability", "median", etc...
So predictor is a unit transformation machine.

It takes forever to converge!

Think like a physicist: Everything has units.
Let x_i be the base unit. Output $\langle w \cdot x \rangle$ has unit "probability", "median", etc...
So predictor is a unit transformation machine.

The ideal w_i has units of $\frac{1}{x_i}$since doubling feature value halves weight.
Update $\propto \frac{\partial L_w(x)}{\partial w} \simeq \frac{\Delta L_w(x)}{\Delta w}$ has units of x_i.
Thus update $= \frac{1}{x_i} + x_i$ unitwise, which doesn't make sense.

Implications

1. Choose x_i near 1, so units are less of an issue.
2. Choose x_i on a similar scale to x_j so unit mismatch across features doesn't kill you.
3. Use a more sophisticated update.

General advice:

1. Many people are happy with TFIDF = weighting sparse features inverse to their occurrence rate.
2. Choose features for which a weight vector is easy to reach as a combination of feature vectors.

Adaptive Updates [DHS10, MS10]

Create per-feature learning rates.

Let $l_i = \sum_{s=1}^{t} \left(\frac{\partial \ell(w_s^\top x_s, y_s)}{\partial w_{s,i}} \right)^2$

Parameter i has learning rate

$$\eta_{t,i} = \frac{\eta}{l_i^p}$$

Adaptive Updates [DHS10, MS10]

Create per-feature learning rates.

Let $l_i = \sum_{s=1}^{t} \left(\frac{\partial \ell(w_s^\top x_s, y_s)}{\partial w_{s,i}} \right)^2$

Parameter i has learning rate

$$\eta_{t,i} = \frac{\eta}{l_i^p}$$

If $p = 1$, this deals with the units problem.

Otherwise, renormalize by $\left(\sum_i x_i^2\right)^{1/(1-p)}$ to help deal with units problem. –nonormalize turns this off.

All together

time vw -c –exact_adaptive_norm –power_t 1 -l 0.5

All together

time vw -c –exact_adaptive_norm –power_t 1 -l 0.5

The interaction of adaptive, importance invariant, renormalized updates is complex, but worked out. Thanks to Paul Mineiro who started that.
Look at local_predict() in gd.cc for details.

The Tutorial Plan

1. Baseline online linear algorithm
2. What goes wrong? And fixes
    2.1 Importance Aware Updates
    2.2 Adaptive updates
3. LBFGS: Miro's turn
4. Terascale Learning: Alekh's turn.
5. Common questions we don't have time to cover.
6. Active Learning: See Daniels's presentation last year.
7. LDA: See Matt's presentation last year.

Ask Questions!

Goals for Future Development

1. Native learning reductions. Just like more complicated losses.
2. Other learning algorithms, as interest dictates.
3. Librarification, so people can use VW in their favorite language.

How do I choose a Loss function?

Understand loss function semantics.

1. Minimizer of squared loss = conditional expectation. $f(x) = E[y|x]$ (default).
2. Minimizer of quantile = conditional quantile. $\Pr(y > f(x)|x) = \tau$
3. Hinge loss = tight upper bound on 0/1 loss.
4. Minimizer of logistic = conditional probability: $\Pr(y = 1|x) = f(x)$. Particularly useful when probabilities are small.

Hinge and logistic require labels in $\{-1, 1\}$.

How do I choose a learning rate?

1. First experiment with a potentially better algo:–exact_adaptive_norm
2. Are you trying to track a changing system? –power_t 0 (forget past quickly).
3. If the world is adversarial: –power_t 0.5 (default)
4. If the world is iid: –power_t 1 (very aggressive)
5. If the error rate is small: -l <large>
6. If the error rate is large: -l <small> (for integration)
7. If –power_t is too aggressive, setting –initial_t softens initial decay.

How do I order examples?

There are two choices:

1. Time order, if the world is nonstationary.
2. Permuted order, if not.

A bad choice: all label 0 examples before all label 1 examples.

How do I debug?

1. Is your progressive validation loss going down as you train? (no => malordered examples or bad choice of learning rate)
2. If you test on the train set, does it work? (no => something crazy)
3. Are the predictions sensible?
4. Do you see the right number of features coming up?

How do I figure out which features are important?

1. Save state
2. Create a super-example with all features
3. Start with –audit option
4. Save printout.

(Seems whacky: but this works with hashing.)

How do I efficiently move/store data?

1. Use –noop and –cache to create cache files.
2. Use –cache multiple times to use multiple caches and/or create a supercache.
3. Use –port and –sendto to ship data over the network.
4. –compress generally saves space at the cost of time.

How do I avoid recreating cachefiles as I experiment?

1. Create cache with -b <large>, then experiment with -b <small>.
2. Partition features intelligently across namespaces and use –ignore <f>.