Vowpal Wabbit 7 Tutorial

Stephane Ross

Binary Classification and Regression Input Format

- Data in text file (can be gziped), one example/line

- Label [weight] |Namespace Feature ... Feature |Namespace ...
  - Namespace: string (can be empty)
  - Feature: string[:value] or int[:value], string is hashed to get index, value 1 by default, features not present have value 0
  - Weight 1 by default
  - Label: use {-1,1} for classification, or any real value for regression

```
 1 | 1:0.43 5:2.1 10:0.1
-1 | I went to school
10 | race=white sex=male age:25
0.5 |optflow  1:0.3 2:0.4 |harris 1:0.5 2:0.9
1 0.154 | 53 1100 8567
```

Binary Classification and Regression Training

- Train on dataset train.txt:

  ./vw -d train.txt

- -d filepath : loads data in filepath
- --passes n : iterate over dataset n times
- -c : creates a binary cache of dataset for faster loading next time (required with --passes n for n > 1)

  ./vw -d train.txt -c --passes 10

General Options

Saving, Loading and Testing Predictors

- -f filepath : where to save final predictor
  ./vw -d train.txt -c --passes 10 -f predictor.vw

- -i filepath : where to load initial predictor, 0 vector when unspecified

- -t : test predictor on data (no training)
  ./vw -d test.txt -t -i predictor.vw

- -p filepath: where to save predictions
  ./vw -d test.txt -t -i predictor.vw -p predictions.txt

- --readable_model filepath: saves a predictor in text format
  ./vw -d train.txt -c --passes 10 --readable_model p.txt

General Options
Loss Functions

- --loss_function loss
  - loss in {squared,logistic,hinge,quantile}
  - squared is default

- Train a SVM (labels must be {-1,1}):

  ./vw -d train.txt -f svm.vw --loss_function hinge

- Train a Logistic Regressor (labels must be {-1,1}):

  ./vw -d train.txt -f lg.vw --loss_function logistic

General Options

L1 and L2 Regularization

- --l1 value, default is 0
- --l2 value, default is 0

- Train a Regularized SVM:

```
./vw -d train.txt -f svm.vw --loss_function hinge --l2 0.1
```

General Options

Update Rule Options

- -l s, scales learning rate, default 0.5 or 10 for non-default rule
- --initial_t i, default 1 or 0 depending on adaptive GD
- --power_t p, default 0.5

- For SGD, this means $\alpha_t = s(i/(i + t))^p$
- Similar effect with the more complex adaptive update rules

```
./vw -d train.txt --sgd -l 0.1 --initial_t 10 --power_t 1
```

General Options
Update Rule Options

- Default is normalized adaptive invariant update rule

- Can specify any combination of --adaptive, --invariant, --normalized
  - --adaptive: uses sum of gradients to decrease learning rate (Duchi)
  - --invariant: importance aware updates (Nikos)
  - --normalized: updates adjusted for scale of each feature

- Or --sgd to use standard update rule

```
./vw -d train.txt --adaptive -l 1
./vw -d train.txt --adaptive --invariant -l 1
./vw -d train.txt --normalized -l 1
```

General Options

Other Useful Options

- -b n, default is n=18: log number of weight parameters, increase to reduce collisions from hashing
- -q ab, quadratic features between all features in namespace a* and b*
- --ignore a, removes features from namespace a*

0.5 |optflow 1:0.3 2:0.4 |harris 1:0.5 2:0.9

./vw --d train.txt --q oo --q oh

Multiclass Classification
Input Format

- One example/line
- Label [weight] |Namespace feature ... feature |Namespace ...
- Label in {1,2,...,k}

1 | 1 5 6
3 | 2 7
2 | 2 4 5 6

Multiclass Classification
Training and Testing

- ./vw -d train.txt --oaa k
  - k = nb classes

- Implemented as Reduction to Binary Classification, 2 Options:

  --oaa k: One Against All, k = nb classes

  --ect k: Error Correcting Tournament /Filter Tree, k = nb classes

  (with --ect, optional --error n, n is nb errors tolerated by code, default n=0)

  ./vw -d train.txt --ect 10 --error 2 -f predictor.vw

  ./vw -d test.txt -t -i predictor.vw

Cost-Sensitive Multiclass Classification
Input Format

- One example/line

- Label ... Label |Namespace feature ... |Namespace feature ...

- Label format: id[:cost]
  - id in {1,2,...,k}
  - cost is 1 by default

- Can specify only subset of labels (if must choose within a subset)

1:0.5 2:1.3 3:0.1 | 1 5 6

2:0.1 3:0.5 | 2 6

Cost-Sensitive Multiclass Classification Training and Testing

- ./vw -d train.txt --csoaa k
  - k = nb classes

- Implemented as a Reduction, 2 Options:

  --csoaa k: Cost-Sensitive One Against All (Reduction to Regression)

  --wap k: Weighted All Pairs (Reduction to weighted binary classification)

```
./vw -d train.txt --wap 10 -f predictor.vw
./vw -d test.txt -t -i predictor.vw
```

"Offline" Contextual Bandit
Input Format

- Cost-Sensitive Multiclass when only observe cost for 1 action/example
- Data collected a priori by "exploration" policy
- One example/line

- Label |Namespace feature ... |Namespace feature ...
- Label format: action:cost:prob
  - action is in {1,2,...,k}
  - cost for this action,
  - prob that action was chosen by exploration policy in this context

1:0.5:0.25 | 1 5 6

3:2.4:0.5 | 2 6

"Offline" Contextual Bandit
Input Format

- Can specify subset of allowed actions if not all available:

  1 2:1.5:0.3 4 | these are the features

- Can specify costs for all unobserved actions for testing learned policy (only action with a prob used/observed for training) :

  1:0.2 2:1.5:0.3 3:2.5 4:1.3 | these are the features
  1:2.2:0.5 4:1.4 | more features

“Offline” Contextual Bandit
Training and Testing

- ./vw -d train.txt --cb k
  - k = nb actions (arms)

- Implemented as a Reduction to Cost-Sensitive Multiclass
  - Optional: --cb_type {ips,dm,dr} specifies how we generate cost vectors
  - ips: inverse propensity score (unbiased estimate of costs using prob)
  - dm: direct method (regression to predict unknown costs)
  - dr: doubly robust (combination of the above 2), default

- Default cost-sensitive learner is --csoaa, but can use --wap

```
./vw -d train.txt --cb 10 --cb_type ips
./vw -d train.txt --cb 10 --cb_type dm --wap 10 -f predictor.vw
./vw -d test.txt -t -i predictor.vw
```

Sequence Predictions

Input Format

- Same format as multiclass for each prediction in a sequence
- Sequences separated by empty line in file

```
1 | This
2 | is
1 | a
3 | sequence

1 | Another
3 | sequence
```

Sequence Predictions
Training and Testing

- For SEARN:

  ./vw --d train.txt -c --passes 10 --searn k --searn_task sequence --searn_beta 0.5 --searn_passes_per_policy 2 -f policy.vw

- For DAgger:

  ./vw --d train.txt -c --passes 10 --searn k --searn_task sequence --searn_as_dagger 0.0001 -f policy.vw

- Testing:

  ./vw -d test.txt -t -i policy.vw

Sequence Predictions

Additional Features From Previous Predictions

- --searn_sequencetask_history h

  h = nb previous predictions to append, default 1

- --searn_sequencetask_bigrams

  Use bigram features from history, not used by default

- --searn_sequencetask_features f

  f = nb previous predictions to pair with observed features, default 0

- --searn_sequencetask_bigram_features

  Use bigram on history paired with observed features, not used by default

Sequence Predictions
Reduction to Cost-Sensitive Multiclass

- Searn/DAgger generates cost-sensitive multiclass examples
- Costs from rollouts of current policy for each possible prediction at each step
- --searn_rollout_oracle: Rollout expert/oracle instead
- --csoaa is default cost-sensitive learner, but can use --wap instead
- Can also use with contextual bandit --cb
  - Rolls out only one sampled prediction rather than all at each step

Sequence Predictions
Example

./vw -d train.txt -c --passes 10 --searn 20 --searn_task sequence --searn_as_dagger 0.0001 --searn_sequencetask_history 2 --searn_sequencetask_features 1 --cb 20 --cb_type ips --wap 20 -f policy.vw

./vw -d test.txt -t -i policy.vw

- Note the impressive stack of reductions:

  Structured Prediction -> Contextual Bandit -> Cost-sensitive Multiclass -> Weighted Binary

Caution

- Default –b 18 often not enough with so many stacked reductions
  - Weight vectors for all reductions stored in the same weight vector
  - Each one accessed through different offsets added to hash/feature index
  - So collisions can occur between "weight vectors" and features
  - Especially with --searn, or "extreme" multiclass problems

- Having some collisions is not always bad
  - Forces to generalize by looking at all features, can't just rely on one

- Can tweak –b through validation on held out set

Cool Trick to deal with Collisions

- Even if # parameters > # distinct features, collisions can occur

- Can copy features many times, hashed to different indices, hoping that 1 copy is collision free

- Easily done with –q: ./vw -d train.txt -q fc

1 |f These are features |c two three four

-1 |f More features |c two three four

- Can tweak both -b, and nb copies through validation on held out data.

Much More

- --lda : latent dirichlet allocation
- --bfgs: use batch lbfgs instead of stochastic gradient descent
- --rank: matrix factorization
- --csoaa_ldf --wap_ldf : cost-sensitive multiclass with label dependent features
- --active_learning : active learning
- Use VW as a library inside another program