**Vowpal Wabbit**

Table of Contents

[**Understanding Vowpal Wabbit:** 2](#_Toc456434065)

[**Vowpal Wabbit Input format:** 3](#_Toc456434066)

[**Start of Demo:** 4](#_Toc456434067)

[**The command we will use to create a model:** 4](#_Toc456434068)

[**CACHING:** 5](#_Toc456434069)

[**Ngram:** 5](#_Toc456434070)

[**Vowpal Wabbit Model Parameters:** 6](#_Toc456434071)

[**Making Predictions on the Test set** 8](#_Toc456434072)

[**PYTHON CODE: Covert input file into Vowpal wabbit format:** 9](#_Toc456434073)

[**http://mlwave.com/install-vowpal-wabbit-on-windows-and-cygwin/**](http://mlwave.com/install-vowpal-wabbit-on-windows-and-cygwin/)

**http://mlwave.com/movie-review-sentiment-analysis-with-vowpal-wabbit/**

**Understanding Vowpal Wabbit:**

Predict whether a house will require a new roof in the next 10 years.

0 | price:.23 sqft:.25 age:.05 2006

1 2 'second\_house | price:.18 sqft:.15 age:.35 1976

0 1 0.5 'third\_house | price:.53 sqft:.32 age:.87 1924

The first number:

0🡪   label corresponds to no roof-replacement,

1->  label corresponds to a roof-replacement

| separates label related data (what we want to predict) from features (what we always know).

The features in the 1st line are price, sqft, age, and 2006

By default, vowpal-wabbit hashes feature names into in-memory indexes unless the feature names themselves are positive integers. In this case, the first 3 features use an index derived from a hash function while the last feature uses index 2006 directly. Also the 1st 3 features have explicit values (.23, .25, and.05 respectively) while the last, 2006 has a implicit default value of 1.

Line 2:

The 2 is an optional importance weight which implies that this example counts twice. A missing importance weight defaults to 1.

'second\_house is the tag, it is used elsewhere to identify the example.

Line3:  This is an initial prediction. Sometimes you have multiple interacting learning systems and want to be able to predict an offset rather than an absolute value.

**Vowpal Wabbit Input format:**

1 'horse |f color\_brown avg\_age:6.5 has\_legs:4 a creature on the farm |a wikipedia\_mentions:15

-1 'oak |f color\_brown avg\_age:75 prospers near ponds and lakes |a wikipedia\_mentions:5

...

Where 1 is the label, 'horse is the identifier, |f and |a are feature spaces (useful to create feature pairs with -q fa or to ignore certain features --ignore a).

**Start of Demo:**

c:\cygwin64\cygwin.bat

cd vowpal\_wabbit

cd vowpalwabbit

./vw --help

./vw --version

**The command we will use to create a model:**

vw <rotten.train.vw> -[c](rotten.train.vw.cache) -k --passes 300 --ngram 7 -b 24 --ect 5 -f rotten.model.vw

Where:

* vw is the Vowpal Wabbit executable
* rotten.train.vw is our train set
* -c -k means to use a cache for multiple passes, and kill any existing cache
* --passes 300 means to make 300 passes over our data set, you use multiple passes with --passes
* --ngram 7 tells Vowpal Wabbit to create [n-grams](http://en.wikipedia.org/wiki/N-gram) (7-grams in this case).
* -b 24 tells Vowpal Wabbit to use 24-bit hashes (18-bit hashes is default)
* -f rotten.model.vw means “save model as ‘rotten.model.vw’”.
* --ect (error correcting tournament [[pdf](http://hunch.net/~beygel/tournament.pdf)]) in very simple terms tells Vowpal Wabbit that there are 5 possible labels and we want it to pick one.

**CACHING:**

If you use multiple passes with --passes, you would need to also pass -c so vw can cache the data in a faster to handle format (passes > 1 should be much faster). By default, the cache file name will be the data-set file with .cache appended. In this case: house\_dataset.cache. You may also override the default cache file name by passing: --cache\_file housing.cache. A cache file can greatly speed up training when you run many experiments (with different options) on the same data-set even if each experiment is only a single pass. So if you want to experiment with the same data-set over and over, it is highly recommended to pass -cevery time you train. If the cache exists and is newer than the data-set, it will be used, if it doesn't exist, it'll be created the first time -c is used.

**Ngram:**

Multiple passes over the data allows Vowpal Wabbit to better fit its model.

n-grams increase performance because a phrase like “this movie was not good” would score positive sentiment for the token “good”. If 2-grams were used the model could detect negative sentiment in the token “not good”.

Vowpal Wabbit is so incredibly fast in part due to the [hashing trick](http://en.wikipedia.org/wiki/Feature_hashing#Feature_vectorization_using_the_hashing_trick). With many features and a small-sized hash collisions start occurring. These collisions may influence the results.

One could also use --oaa (one against all) instead of --ect (error correcting tournament) but “ect” at times outperforms “oaa”. In this case it produces a lower average loss.

**Vowpal Wabbit Model Parameters:**

num sources = 1

only one input file in our example. But you can specify multiple files

Num weight bits = 18

Only 18 bits of the hash function will be used. You could adjust the number of bits using -b bits

learning rate = 0.5

 default learning rate is 0.5

On these larger data-sets, our learning rate will by default decay towards 0

A higher learning rate will make the model converge faster but a too high learning rate may over-fit and end up be worse on average.

initial\_t = 0

Learning rates should often decay over time, and this specifies the initial time.

power\_t = 0.5

This specifies the power on the learning rate decay

 You can adjust this --power\_t p where p is in the range [0,1].

 0 means the learning rate does not decay

1 is very aggressive

 0.5 is a minimax optimal choice

average since example example current current current

loss last counter weight label predict features

0.000000 0.000000 1 1.0 0.0000 0.0000 5

0.666667 1.000000 2 3.0 1.0000 0.0000 5

average loss computes the [progressive validation](http://hunch.net/~jl/projects/prediction_bounds/progressive_validation/coltfinal.pdf) loss.

 average loss computes the [progressive validation](http://hunch.net/~jl/projects/prediction_bounds/progressive_validation/coltfinal.pdf) loss. The critical thing to understand here is that progressive validation loss deviates like a test set, and hence is a reliable indicator of success on the first pass over any data-set.

since last is the progressive validation loss since the last printout.

example counter tells you which example is printed out. In this case, it's example 2.

example weight tells you the sum of the importance weights of examples seen so far. In this case it's 3, because the second example has an importance weight of 2.

current label tells you the label of the second example.

current predict tells you the prediction (before training) on the current example.

current features tells you how many features the current example has. This is great for debugging, and you'll note that we have 5 features when you expect 4. This happens, because VW has a default constant feature which is always added in. Use the --noconstant command-line option to turn it off

**passes 300 \* number of records in train data🡪 example counter**

You'll notice that by example 47 (25 passes over 3 examples result in 75 examples), the since lastcolumn has dropped to 0, implying that by looking at the same (3 lines of) data 25 times we have reached a perfect predictor

average since example example current current current

loss last counter weight label predict features

0.000000 0.000000 1 1.0 0.0000 0.0000 5

0.666667 1.000000 2 3.0 1.0000 0.0000 5

0.589385 0.531424 5 7.0 1.0000 0.2508 5

0.378923 0.194769 11 15.0 1.0000 0.8308 5

0.184476 0.002182 23 31.0 1.0000 0.9975 5

0.090774 0.000000 47 63.0 1.0000 1.0000 5

**Making Predictions on the Test set**

Now we have our model we can tell Vowpal Wabbit to predict the labels for our test set.

To run Vowpal Wabbit in test mode and create predictions:

vw <rotten.test.vw> -t -i <rotten.model.vw> -p <rotten.preds.txt>

Where:

* vw is the Vowpal Wabbit executable
* rotten.test.vw the location to our test set
* -t tells to test only (no learning)
* -i rotten.model.vw says to use rotten.model.vw as the model
* -p rotten.preds.txt means “save predictions as ‘rotten.preds.txt’”

Predicted Rating:

157290 4

156140 1

156061 3

156555 2

156567 0

**PYTHON CODE: Covert input file into Vowpal wabbit format:**

**import** csv  
**import** re  
  
location\_train = **"C:\\Users\\ilanigam17\\Desktop\\ads\\ADS\_Projects\\VowpalWabbit\\train\\train.tsv"**location\_test = **"C:\\Users\\ilanigam17\\Desktop\\ads\\ADS\_Projects\\VowpalWabbit\\test\\test.tsv"**location\_train\_vw = **"rotten.train.vw"** *#will be created*location\_test\_vw = **"rotten.test.vw"** *#will be created  
  
#cleans a string "I'm a string!?" returns as "i m a string"***def** clean(s):  
 **return " "**.join(re.findall(**r'\w+'**, s,flags = re.UNICODE | re.LOCALE)).lower()  
*#creates Vowpal Wabbit-formatted file from tsv file***def** to\_vw(location\_input\_file, location\_output\_file, test = False):  
 **print "\nReading:"**,location\_input\_file,**"\nWriting:"**,location\_output\_file  
 **with** open(location\_input\_file) **as** infile, open(location\_output\_file, **"wb"**) **as** outfile:  
 *#create a reader to read train file* reader = csv.DictReader(infile, delimiter=**"\t"**)  
 *#for every line* **for** row **in** reader:  
 *#if test set label doesnt matter/or isnt available* **if** test:  
 label = **"1"  
 else**:  
 label = str(int(row[**'Sentiment'**]) + 1)  
 phrase = clean(row[**'Phrase'**])  
 outfile.write(label +  
 **" '"** + row[**'PhraseId'**] +  
 **" |f "** +  
 phrase +  
 **" |a "** +  
 **"word\_count:"** + str(phrase.count(**" "**) + 1)  
 + **"\n"**)  
  
to\_vw(location\_train, location\_train\_vw)  
to\_vw(location\_test, location\_test\_vw, test=True)