Implementing poisson loss in VW

Sharat Chikkerur

Introduction

Poisson basics:

$$
\begin{aligned}
P(y|\lambda) &= \frac{e^{-\lambda}\lambda^{y}}{y!} \\
E[y] &= \lambda \\
var[y] &= \lambda
\end{aligned}
$$

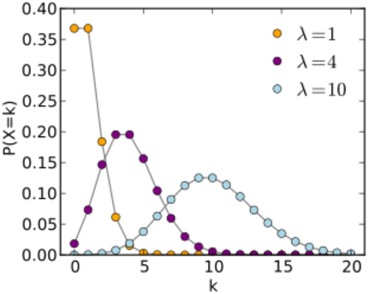


Figure: Poisson distribution [1]

[1] https://en.wikipedia.org/wiki/Poisson_distribution

Poisson regression

When to use poisson regression

- Need to model count or rate data (e.g. clicks in an hour)
- Prediction needs to be positive
- Variance is proportional to the mean

In a GLM model, we implement poisson regression as

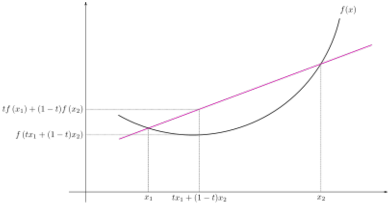
$$\log(\lambda) = w^T x$$

- The formulation is identical with a log-linear models
- The difference b/w log-linear and poisson is the choice of loss function

Detour: Jensen's inequality

For a convex function $\phi()$ and a random variable X, we have

$$\phi(E[X]) \leq E[\phi(x)]$$



For the $-\log()$ function we have

$$log(E[X]) \geq E[\log(X)]$$

A log-linear model will attempt to predict $E[\log(X)]$ and is bound to under predict the true estimate $log(E[X])$

Poisson loss

$$P(y|\lambda) = \frac{e^{-\lambda}\lambda^y}{y!}, \log(\lambda) = w^T x$$

To optimize the parameters w, we have a choice of metrics to optimize

- Negative log-likelihood [2]

$$l(w) = -\log(P(y;w)) = \lambda - y\log(\lambda) = \exp(w^T x) - yw^T x$$

- Deviance

$$D(w) = y\log(\frac{y}{\lambda}) - (y - \lambda) = y\log y - yw^T x - y + \exp(w^T x)$$

[2] We ignore the $y!$ term since it does not depend on parameters

Gradient descent

We choose the deviance since it is similar to RMSE semantics in squared loss. The gradient update is similar to other generalized linear models.

$$\nabla l(w) = \left(\exp(w^T x) - y\right) x = (\text{prediction} - \text{label}) x$$

This update is similar to other generalized linear models.

Importance aware update

For an importance weight of h, we apply a scaling factor $s(h)$ instead of scaling gradient by h to reduce noise. $s(h)$, approximates doing predict-update steps for h individual instances. The scaling factor $s(h)$ satisfies

$$
\begin{aligned}
ds &= \eta \frac{\partial l(p,y)}{\partial p}\Big|_{p=(w-s(h))^T x} \\
&= \exp(p) - y\big|_{p=(w-s(h))^T x} \\
&= \exp((w-s(h))^T x - y)
\end{aligned}
$$

Here, $p = w^T x$ and $l(p,y) = \exp(p) - yp$

Importance aware update (cont.)

The solution is given by

$$s(h) = \frac{\log\left(\frac{e^{-\eta h x^2 y}\left(e^{x(\eta h x y + w)} - e^{wx} + y\right)}{y}\right)}{x^2}$$

This can also be written as

$$s(t) = \frac{\log\left(\frac{e^{-acdt}\left(e^{b}\left(e^{acdt} - 1\right) + d\right)}{d}\right)}{c}$$

Here, $t = h, c = x^2, b = wx, d = y, a = \eta$. For derivations see, `http://bit.ly/1Us0oVe` and `http://bit.ly/25ZG6Gp`

Using Poisson regression

- Training
  - `--loss_function=poisson --link=poisson`
  - link function is needed to get predictions in the original domain. By default vw reports $\log(\lambda)$.
- Testing
  - `--loss_function=poisson --link=poisson`
  - loss function is needed to report poisson loss (deviance) on stderr. By default vw reports squared loss.

Demo

```
#!/bin/bash
export VW=/usr/local/bin/vw

# Examine data file
head poisson.dat

#Train poisson model using l = 0.01 posses 100
${VW} -d poisson.dat -f model \textcolor{red}{--loss_function poisson --link poisson|} -b 2 -l 0.01 --pass

# Get predictions
${VW} -d poisson.dat -i model -p poisson.train.predict --loss_function poisson -t

# Compare labels with predictions
cut -f1 -d " " poisson.dat > labels; paste labels poisson.train.predict | less
```

Log-linear vs. poisson regression

`https://github.com/JohnLangford/vowpal_wabbit/blob/master/python/examples/poisson_regression.ipynb`