Word Probabilities Algorithm and Resources

Abstract

Estimate the half-life of words in memory. The longer the half-life , the easier it is to remember words.

Source

Paper

Duolingo source code

Implementation

Half Life Regression

Probability

The probability of recalling a word correctly is given by

$$\mathrm{p} = 2^{-\Delta/\mathrm{h}}$$

where, p is the probability of correctly recalling a word; Δ is the lag time since the item was last practised; and h is the half life of the strength of a learner's memory.

For example, consider the following cases:

- $\Delta = 0$ implies the word was recently practised and $\mathrm{p} = 1$
- $\Delta = \infty$ or $\Delta \gg \mathrm{h}$ implies the word has been practised a long time ago and $\mathrm{p} = 0$. Thus, the word has been forgotten.
- $\Delta = \mathrm{h}$ implies $\mathrm{p} = 0.5$. The learner is on the verge of forgetting the word.

Define true probability as a function of the columns of a feature vector $\mathbf{x}$. An example can be number of correct recall divided by total interactions with the word.

Half Life

The estimate of the half life is given by

$$\hat{\mathrm{h}}_{\boldsymbol{\theta}} = 2^{\boldsymbol{\theta}\cdot\mathbf{x}}$$

where $\mathbf{x}$ is the feature vector summarising the student's previous exposure to the word; and $\boldsymbol{\theta}$ denotes the weight vector. (In most machine learning literature, weight vectors are denoted by $\mathbf{w}$)

Feature Vector

An individual feature vector $\mathbf{x}$ consists of counts of various interactions (for example, the number of times the word has been seen; the number of times the word has been recalled in a challenge; the number of times the word has been recalled incorrectly in a challenge; etc.) of a student with a word.

Loss Function

The loss function is an L_2 regularised loss function and is given by

$$\mathcal{L}(\mathrm{p}, \Delta, \mathbf{x}; \boldsymbol{\theta}) = (\mathrm{p} - \hat{\mathrm{p}})^2 + \alpha\left(\mathrm{h} - \hat{\mathrm{h}}\right)^2 + \lambda\,||\boldsymbol{\theta}||_2^2$$

Here, λ is the regularisation rate and is used to ensure that the weights do not end up becoming too large. Large weights are a sign of overfitting. α is a weight which decides the relative importance of the second term compared to the first. Thus, $0 \leq \alpha \leq 1$. Even though probability is a function of half-life, experimental results are better when the second term is present in the loss function. Since the true values of h can never be known, it is calculated as

$$\mathrm{h} = -\frac{\Delta}{\log_2 \mathrm{p}}$$

Learning

Learning is achieved by finding weights that minimise the loss function. The optimal weights are given by

$$\boldsymbol{\theta}^* = \min_{\boldsymbol{\theta}} \sum_{\mathrm{i}=1}^{\mathrm{D}} \mathcal{L}\left(\mathrm{p_i},\, \Delta_{\mathrm{i}},\, \mathbf{x}_{\mathrm{i}};\, \boldsymbol{\theta}\right)$$

Since the loss function is smooth, the weights weights can be updated using gradient descent optimisation. Adagrad is a good algorithm, which retains gradient history while training.

Data and Data Model

```
Feature vector: x
    correct recall
    wrong recall
    learnt
    review

Time stamp: t
    time lapse since student's last encounter with word

Probability: p
    ((2*correct) + learnt) / (2*(correct + wrong) + (learnt + review))

Weights: w
    computed while training

Half life: h
    dot(x, w)



Data from flash cards
    First attempt
        Mark as learnt
            learnt = learnt + 1
        Mark as review
            review = review + 1
    Use hint
        Mark as learnt
            learnt = learnt + 0.75
            review = review + 0.25
        Mark as review
            learnt = learnt + 0.25
            review = review + 0.75
    Show details
        Mark as learnt
            learnt = learnt + 0.6
            review = review + 0.4
        Mark as review
            learnt = learnt + 0.4
            review = review + 0.6
```

```
Data from challenges
    tap 1
    tap 2
    result
    if result = wrong
        wrong recall of both words += 1
    else
        correct recall of both words += 1




Database for flash cards
    student id
    word id
    action

Database for challenges
```

```
    student id
    challenge id (may not be useful)
    list [tap1 tap2 result tap1 tap 2 result .....]
```

Algorithm

```
# Assume data is in csv
# Import csv
# Extract relevant data into a numpy array
# Each row of the array represents one data point

set hyperparameters for training
 lr: learning rate
 w: weights for feature vector (one extra dimension for bias)
 w_h: alpha in the loss function
 w_l2: lambda in the loss function
 min_h: minimum value for h. 15 minutes (measured in days)
 max_h: maximum value for h. 9 months (measured in days)

train_model = initialise StuWordRegression class

for row in data_array:
 # row is the current feature vector
 p = calc_p(row) # depends on the probability model used
 t = calc_delta(timestamps)
 h = -delta /log(p)


 train_model.train(row, p, h, t)

 # once training is complete, save the weights. Use these weights for
predictions on the test set.
 # always keep a track of dL_p, dl_h. These are gradients and if they
become 0 or a large number, it means the training has failed and needs
to be restarted with different learning rates, weights, or other
hyperparameters.
 # every 100th or 1000th iteration (depending on size of the dataset),
validate the training with a validation set. Look at the training error
and the validation error.
 # training is the most time consuming process of machine learning. It
will require many restarts, and sometimes early stopping criteria will
need to be implemented (for example, validation error starts growing
after falling).
 # the test set should NEVER be used to change weights while training.
Use the validation set to see if there is progress in training.
```

Resources

Overfitting

Gradient descent

Gradient descent algorithms (Adagrad may not be the best solution, depending on the data)

Cross validation, tuning

Maximum Likelihood Estimation

Introduction to theano (theano might be useful because gradients need not be calculated manually. Also, a lot of gradient descent algorithms can be implemented on the fly. Has a bit of a learning curve)

theano documentation

keras (uses theano as a backend. Useful for implementing neural networks easily)