CS 6501 Natural Language Processing

Logistic Regression

Yangfeng Ji

September 5, 2018

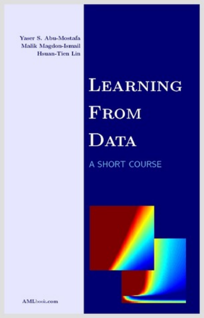
Department of Computer Science
University of Virginia

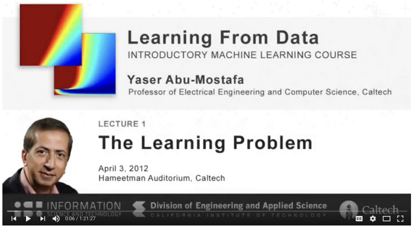
UNIVERSITY of VIRGINIA | ENGINEERING

First Independent Project

- Will be released this weekend
- Topic: text classification using Perceptron and logistic regression
- Requirement: implement these two models and compare the performance with existing implementations
- Time: about two weeks
- No late submission

About Machine Learning Background





Overview

1. Logistic Regression
2. Example
3. Better LR Models
4. Connections to Neural Networks

Classification

- Input: a text $\boldsymbol{x}$
- Output: $y \in \mathcal{Y}$, where $\mathcal{Y}$ is the predefined category set

$$\text{Text } \boldsymbol{x} \longrightarrow \boxed{\text{Classifier}} \longrightarrow \text{Category } y$$

Mathematical Formulation

- Input: a numeric representation $\boldsymbol{x}$
- Output: scores $\Psi(\boldsymbol{x}, y; \boldsymbol{\theta}) \in \mathbb{R}, \forall y \in \mathcal{Y}$

Classification

$$\hat{y} = \arg\max_{y' \in \mathcal{Y}} \Psi(\boldsymbol{x}, y'; \boldsymbol{\theta}) \tag{1}$$

Questions

Linear score function $\Psi(\boldsymbol{x}, y; \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, y)$

1. How to build $\boldsymbol{f}(\boldsymbol{x}, y)$?
   - Bag-of-words representation

2. How to learn $\boldsymbol{\theta}$?
   - Perceptron algorithm

Questions

Linear score function $\Psi(\boldsymbol{x}, y; \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, y)$

1. How to build $\boldsymbol{f}(\boldsymbol{x}, y)$?
   - Bag-of-words representation
   - n-gram feature representation
2. How to learn $\boldsymbol{\theta}$?
   - Perceptron algorithm
   - Logistic regression

Logistic Regression

Logistic Regression

On $|\mathcal{Y}|$-category classification

$$P(y|\boldsymbol{x};\boldsymbol{\theta}) = \frac{\exp(\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, y))}{\sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, y'))}$$

- $P(y|\boldsymbol{x};\boldsymbol{\theta})$: probability of y given $\boldsymbol{x}$
- $\boldsymbol{f}(\boldsymbol{x}, y)$: feature function
- $\boldsymbol{\theta}$: classification parameters
- $|\mathcal{Y}|$: number of elements in set $\mathcal{Y}$

Classification

Perceptron algorithm:

$$\hat{y} \leftarrow \arg\max_{y'} \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, y') \tag{2}$$

Logistic regression:

$$\hat{y} \leftarrow \arg\max_{y'} P(y' | \boldsymbol{x}; \boldsymbol{\theta}) \tag{3}$$

Likelihood

Given a training set $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}$

$$L(\boldsymbol{\theta}) = \prod_{i=1}^{N} P(y^{(i)} | \boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \tag{4}$$

$L(\boldsymbol{\theta})$

- likelihood: measure how good the model fits the training data
- a function of $\boldsymbol{\theta}$

Training: Maximum likelihood estimation

Learning $\boldsymbol{\theta}$ by optimizing $L(\boldsymbol{\theta})$

$$\max_{\boldsymbol{\theta}} \prod_{i=1}^{N} P(y^{(i)} | \boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \tag{5}$$

Training: Maximum likelihood estimation

Due to numeric issues, usually maximize $\log L(\boldsymbol{\theta})$ instead

$$\begin{aligned}\log L(\boldsymbol{\theta}) &= \log \prod_{i=1}^{N} P(y^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}) \\ &= \sum_{i=1}^{N} \log P(y^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}),\end{aligned}$$

or minimize the negative log likelihood function

$$\ell(\boldsymbol{\theta}) = -\log L(\boldsymbol{\theta}) \tag{6}$$

Log-linear Models

$$\begin{aligned}
\log P(y|\boldsymbol{x};\boldsymbol{\theta}) &= \log \frac{\exp(\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, y))}{\sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, y'))} \\
&= \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, y) - \log \sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, y'))
\end{aligned}$$

Log-linear Models

$$\log P(y|\boldsymbol{x};\boldsymbol{\theta}) = \log \frac{\exp(\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, y))}{\sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, y'))}$$

$$= \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, y) - \log \sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, y'))$$

Two comments

- $\log \sum \exp(\cdot)$ function, e.g., `torch.log_sum_exp`
- prediction only relies on the first part

Gradient

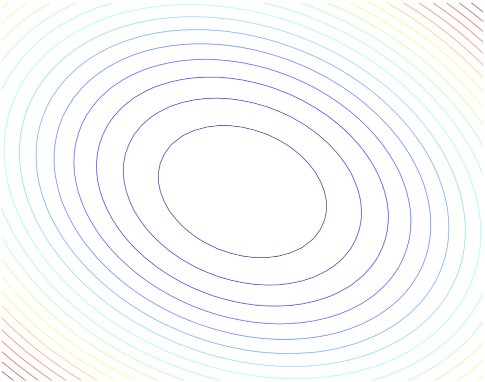
$$\begin{aligned}\frac{\partial \log P(y|\boldsymbol{x};\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} &= \frac{\partial}{\partial \boldsymbol{\theta}}\left\{\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x},y) - \log \sum_{y'\in\mathcal{Y}} \exp(\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x},y'))\right\} \\ &= \boldsymbol{f}\left(\boldsymbol{x},y\right) - \frac{\partial}{\partial \boldsymbol{\theta}} \log \sum_{y'\in\mathcal{Y}} \exp(\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x},y')) \\ &= \boldsymbol{f}\left(\boldsymbol{x},y\right) - \mathbb{E}_{Y|X}[\boldsymbol{f}\left(\boldsymbol{x},y\right)]\end{aligned}$$

Expectation of $\boldsymbol{f}$

$$\mathbb{E}_{Y|X}[\boldsymbol{f}\left(\boldsymbol{x},y\right)] = \sum_{y'\in\mathcal{Y}} \left\{P(y'|\boldsymbol{x})\boldsymbol{f}(\boldsymbol{x},y')\right\} \tag{7}$$

[Eisenstein, 2018, Sec. 2.4.2]

Optimization with Gradient



Two elements

- which direction?
- how far it goes?

Updating Rules

- Logistic regression

$$\boldsymbol{\theta}_{\mathrm{new}} \leftarrow \boldsymbol{\theta}_{\mathrm{old}} + \eta \cdot (\boldsymbol{f}\left(\boldsymbol{x}, y\right) - \mathbb{E}_{Y|X}[\boldsymbol{f}\left(\boldsymbol{x}, y\right)])$$

γ is step size (hyper-parameter).

Updating Rules

- Logistic regression

$$\boldsymbol{\theta}_{\text{new}} \leftarrow \boldsymbol{\theta}_{\text{old}} + \eta \cdot (\boldsymbol{f}(\boldsymbol{x}, y) - \mathbb{E}_{Y|X}[\boldsymbol{f}(\boldsymbol{x}, y)])$$

γ is step size (hyper-parameter).

- Perceptron

$$\boldsymbol{\theta}_{\text{new}} \leftarrow \boldsymbol{\theta}_{\text{old}} + \boldsymbol{f}(\boldsymbol{x}, y) - \boldsymbol{f}(\boldsymbol{x}, \hat{y})$$

$$\boldsymbol{\theta}_{\text{new}} \leftarrow \boldsymbol{\theta}_{\text{old}} + \eta \cdot (\boldsymbol{f}\left(\boldsymbol{x}, y\right) - \mathbb{E}_{Y|X}[\boldsymbol{f}\left(\boldsymbol{x}, y\right)])$$

Why LR is better?

$$\mathbb{E}_{Y|X}[\boldsymbol{f}\left(\boldsymbol{x}, y\right)] = \sum_{y} P(y'|\boldsymbol{x})\boldsymbol{f}\left(\boldsymbol{x}, y'\right) \tag{8}$$

Not just based on the current mistake

Training Objective

	Perceptron	Logistic Regression
Objective	Accuracy Non-differentiable	Likelihood Differentiable
Updating	$\boldsymbol{\theta}_{\text{new}} \leftarrow \boldsymbol{\theta}_{\text{old}} + \boldsymbol{f}(\boldsymbol{x}, y) - \boldsymbol{f}(\boldsymbol{x}, \hat{y})$	$\boldsymbol{\theta}_{\text{new}} \leftarrow \boldsymbol{\theta}_{\text{old}} + \eta \cdot (\boldsymbol{f}(\boldsymbol{x}, y) - \mathbb{E}_{Y\mid X}[\boldsymbol{f}(\boldsymbol{x}, y)])$
Prediction	$\hat{y} \leftarrow \arg\max_{y'} \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, y'; \boldsymbol{\theta})$	

Online vs. Batch Learning

Likelihood:

$$\log L(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log P(y^{(i)} | \boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \tag{9}$$

Online vs. Batch Learning

Likelihood:

$$\log L(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log P(y^{(i)} | \boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \tag{9}$$

Gradient:

$$\frac{\partial \log L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{i=1}^{N} \left\{ \boldsymbol{f}\left(\boldsymbol{x}^{(i)}, y^{(i)}\right) - \mathbb{E}_{Y|X}[\boldsymbol{f}\left(\boldsymbol{x}^{(i)}, y\right)] \right\} \tag{10}$$

Online vs. Batch Learning

Likelihood:

$$\log L(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log P(y^{(i)} | \boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \tag{9}$$

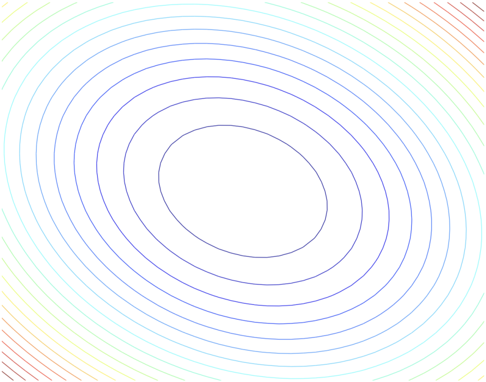
Gradient:

$$\frac{\partial \log L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{i=1}^{N} \left\{ \boldsymbol{f}\left(\boldsymbol{x}^{(i)}, y^{(i)}\right) - \mathbb{E}_{Y|X}[\boldsymbol{f}\left(\boldsymbol{x}^{(i)}, y\right)] \right\} \tag{10}$$

Update:

$$\boldsymbol{\theta}_{\text{new}} \leftarrow \boldsymbol{\theta}_{\text{old}} + \eta \frac{\partial \log L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \tag{11}$$

Online vs. Batch Learning (II)



Notation Change

Objective

$$\ell(\boldsymbol{\theta}) = -\log L(\theta) = -\sum_{i=1}^{N} \log P(y^{(i)} | \boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \tag{12}$$

Update

$$\boldsymbol{\theta}_{\text{new}} \leftarrow \boldsymbol{\theta}_{\text{old}} - \eta \frac{\partial \ell(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \tag{13}$$

[Eisenstein, 2018, Sec. 2.4.2]

Example

Example Dataset

A subset of the Yelp Dataset

`https://www.yelp.com/dataset/challenge`

	Training	Development	Test
Documents	40K	5K	5K
Words	4.7M	0.5M	0.6M

- 5 classes (user rating from 1 to 5)
- Code available on
  `https://github.com/jiyfeng/textclassification`

Building BoW Representations

Sklearn function

```
sklearn.feature_extraction.text.CountVectorizer
```

- Given a collection of texts, it will build a vocab and also convert all texts into numeric vectors
- With Logistic Regression, the classification performance

Logistic Regression

Sklearn function

```
sklearn.linear_model.LogisticRegression
```

- Classification accuracy on the development data is 61.4%

How Far We Can Go with BoW?

Tricks to reduce vocab size

- remove punctuation (default)
- lowercase (↗)
- remove low-frequency words (↗)
- remove high-frequency words (↘)
- replace numbers with a special token

How Far We Can Go with BoW?

Tricks to reduce vocab size

- remove punctuation (default)
- lowercase ($\nearrow$)
- remove low-frequency words ($\nearrow$)
- remove high-frequency words ($\searrow$)
- replace numbers with a special token

Comments

- Not always helpful (these are empirical tricks)
- Not always the case (it depends on the data/domain)

Interpretability

Weights learned from training data

$$
\begin{array}{ccc}
\text{Vocab} & w_{\text{rating}=1} & w_{\text{rating}=5} \\
\left\{\begin{array}{c} \text{SUPER} \\ \cdots \\ \text{QUICK} \\ \text{FOOD} \\ \text{FRIENDLY} \\ \text{EAT} \\ \cdots \\ \text{DELICIOUS} \end{array}\right\}
&
\begin{bmatrix} 0.33 \\ \cdots \\ -1.26 \\ 0.08 \\ -2.57 \\ -0.47 \\ \cdots \\ -3.60 \end{bmatrix}
&
\begin{bmatrix} -0.09 \\ \cdots \\ -0.01 \\ -0.09 \\ 0.16 \\ 0.00 \\ \cdots \\ 0.64 \end{bmatrix}
\end{array}
$$

Interpretability (II)

Top features

rating = 5	rating = 1
exceptional	worst
incredible	joke
phenomenal	disgusted
body	unprofessional
regret	garbage
worried	disgusting
skeptical	*luck*
hesitate	pathetic
happier	apologies
mike	horrible

Better LR Models

Ways to Improve LR Models

- Richer feature set
- Regularization
- Better optimization methods

Richer Features: bi-grams

Combine words together to form high-order features

- Uni-grams:

$$\{\textsc{fast}, \textsc{slow}, \textsc{super}\}$$

Richer Features: bi-grams

Combine words together to form high-order features

- Uni-grams:

  {FAST, SLOW, SUPER}

- Bi-grams:

  {SUPER FAST, SUPER SLOW}

Extended feature set

{FAST, SLOW, SUPER, SUPER FAST, SUPER SLOW}

Performance

Sklearn function

`sklearn.feature_extraction.text.CountVectorizer`

with `ngram_range=(1,2)`

- Even larger vocab size: from 60K to 1M
- Performance change: from 61.4% to 62.4% on the dev data

Problems with Large Feature Set

- Overlap among features

Example

{FAST, SLOW, SUPER, SUPER FAST, SUPER SLOW}

Problems with Large Feature Set

- Overlap among features

Example

{FAST, SLOW, SUPER, SUPER FAST, SUPER SLOW}

- Learning in high dimensional space — overfitting

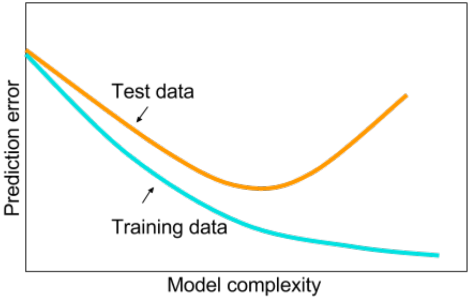
Example

62.4% on dev data

vs.

$\sim$ 100% on training data

Overfitting



[Abu-Mostafa et al., 2012]

Ways to Improve LR Models

- ✓ Richer feature set
- Regularization
- Better optimization methods

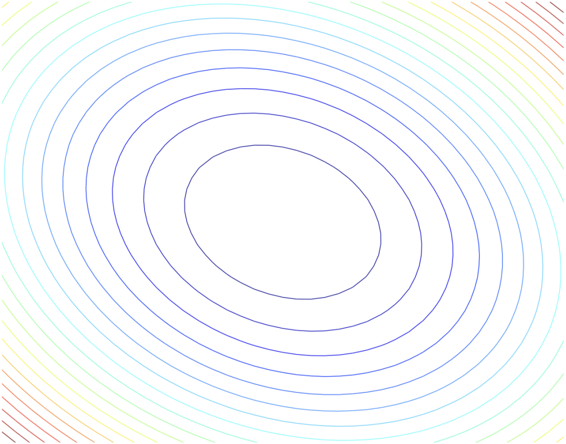
L_2 Regularization

Add an additional constraint on $\boldsymbol{\theta}$

$$\ell_{L_2}(\boldsymbol{\theta}) = -\sum_{i=1}^{N} \log P(y^{(i)} | \boldsymbol{x}^{(i)}; \boldsymbol{\theta}) + \frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2 \tag{14}$$

[Eisenstein, 2018, Sec. 2.4.1]

L_2 Regularization (Cont.)



Ways to Improve LR Models

- ✓ Richer feature set
- ✓ Regularization
- ► Better optimization methods

Optimization with Gradient Descent

At time step t

$$\boldsymbol{\theta}^{(t+1)|} \leftarrow \boldsymbol{\theta}^{(t)} - \eta^{(t)} \frac{\partial \ell(\boldsymbol{\theta}^{(t)})}{\partial \boldsymbol{\theta}^{(t)}} \tag{15}$$

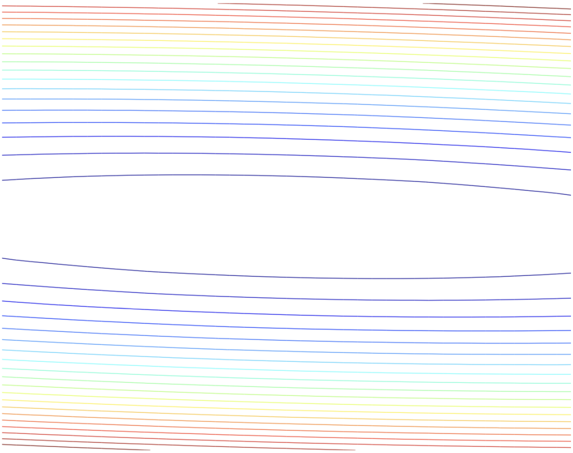
- Direction
- Step size

Adaptive Gradient Methods

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \eta \cdot \frac{1}{\sqrt{\sum_{k=1}^{t}\left(\frac{\partial \ell(\boldsymbol{\theta}^{(k)})}{\partial \boldsymbol{\theta}^{(k)}}\right)^2}} \cdot \frac{\partial \ell(\boldsymbol{\theta}^{(t)})}{\partial \boldsymbol{\theta}^{(t)}} \tag{16}$$

[Duchi et al., 2011, AdaGrad]

Adaptive Gradient Methods (Cont.)



Ways to Improve LR Models

- ✓ Richer feature set
- ✓ Regularization
- ✓ Better optimization methods

Connections to Neural Networks

About Feature Function $f(x, y)$

- LR
  - feature engineering
  - feature selection
- Neural networks
  - representation learning: learning f directly

About Optimization Methods

Having a better optimization method

- LR
  - helpful
- Neural networks
  - crucial
  - sophisticated methods (e.g., AdaGrad) is not always better than SGD

About Overfitting

- LR
  - happens if two many features
  - L_2 regularization helps
  - Other options: L_1 regularization
- Neural networks
  - happens if too many neurons
  - L_2 regularization (weight decay) helps
  - Other options: dropout

Summary

- Logistic regression
- Better LR models
    - Rich feature set
    - Regularization
    - Optimization methods

Reference

Abu-Mostafa, Y. S., Magdon-Ismail, M., and Lin, H.-T. (2012).
Learning from data, volume 4.
AMLBook New York, NY, USA:.

Duchi, J., Hazan, E., and Singer, Y. (2011).
Adaptive subgradient methods for online learning and stochastic optimization.
Journal of Machine Learning Research, 12(Jul):2121–2159.

Eisenstein, J. (2018).
Natural Language Processing.
MIT Press.