CS 6501 Natural Language Processing

Text Classification (I): Logistic Regression

Yangfeng Ji

Information and Language Processing Lab
Department of Computer Science
University of Virginia

Overview

1. Problem Definition
2. Bag-of-Words Representation
3. Case Study: Sentiment Analysis
4. Logistic Regression
5. Overfitting and L_2 Regularization

Problem Definition

Case I: Sentiment Analysis

[Pang et al., 2002]

Case II: Topic Classification

Example topics

- Business
- Arts
- Technology
- Sports
- …

Case III: Natural Language Inference (NLI)

NLI can be formulated as text classification problems – classifying the relation between two texts

- Input:
  - A premise (e.g., "*Soccer game with multiple males playing*") and
  - A hypothesis (e.g., "*Some men are playing a sport*")
- Output: The relation between the premise and the hypothesis (e.g., Entailment, Contradiction, and Neutral)

Case IV: Multi-choice Question Answering

Picking an answer is equivalent to predict which one is the most likely answer

- Context: *"My name is Yangfeng Ji and I live in Charlottesville"*
- Question: *"Where do I live?"*
- Candidate answers:
  A. *"Beijing"*
  B. *"Seattle"*
  C. *"Charlottesville"*
  D. *"London"*

Case IV: Multi-choice Question Answering

Picking an answer is equivalent to predict which one is the most likely answer

- Context: *"My name is Yangfeng Ji and I live in Charlottesville"*
- Question: *"Where do I live?"*
- Candidate answers:
  A. *"Beijing"*
  B. *"Seattle"*
  C. *"Charlottesville"*
  D. *"London"*

To use a classifier for question answering, what should be the input?

General Setup

A formal formulation of classification problem

- **Input**: a text $\boldsymbol{x}$
  - Example: a product review on Amazon
- **Output**: $y \in \mathcal{Y}$, where $\mathcal{Y}$ is the predefined label set
  - Example: $\mathcal{Y} = \{\textsc{Positive}, \textsc{Negative}\}$

[1] In this course, we use $\boldsymbol{x}$ for both text and its representation with no distinction

General Setup

A formal formulation of classification problem

- **Input**: a text $\boldsymbol{x}$
  - Example: a product review on Amazon
- **Output**: $y \in \mathcal{Y}$, where $\mathcal{Y}$ is the predefined label set
  - Example: $\mathcal{Y} = \{\textsc{Positive}, \textsc{Negative}\}$

The pipeline of text classification:[1]

$$\text{Text} \longrightarrow \boxed{\text{Numeric Vector } \boldsymbol{x}} \longrightarrow \boxed{\text{Classifier}} \longrightarrow \text{Label } y$$

[1]In this course, we use $\boldsymbol{x}$ for both text and its representation with no distinction

Probabilistic Formulation

With the conditional probability $P(Y \mid \boldsymbol{X})$, the prediction on Y for a given text $\boldsymbol{X} = \boldsymbol{x}$ is

$$\hat{y} = \operatorname*{argmax}_{y \in \mathcal{Y}} P(Y = y \mid \boldsymbol{X} = \boldsymbol{x}) \tag{1}$$

Probabilistic Formulation

With the conditional probability $P(Y \mid \boldsymbol{X})$, the prediction on Y for a given text $\boldsymbol{X} = \boldsymbol{x}$ is

$$\hat{y} = \operatorname*{argmax}_{y \in \mathcal{Y}} P(Y = y \mid \boldsymbol{X} = \boldsymbol{x}) \tag{1}$$

Or, for simplicity

$$\hat{y} = \operatorname*{argmax}_{y \in \mathcal{Y}} P(y \mid \boldsymbol{x}) \tag{2}$$

Probabilistic Formulation

With the conditional probability $P(Y \mid \boldsymbol{X})$, the prediction on Y for a given text $\boldsymbol{X} = \boldsymbol{x}$ is

$$\hat{y} = \operatorname*{argmax}_{y \in \mathcal{Y}} P(Y = y \mid \boldsymbol{X} = \boldsymbol{x}) \tag{1}$$

Or, for simplicity

$$\hat{y} = \operatorname*{argmax}_{y \in \mathcal{Y}} P(y \mid \boldsymbol{x}) \tag{2}$$

In sklearn, this argmax is implemented by the predict function

`predict`(*X*) [source]

Predict class labels for samples in X.

Parameters:

X : ***{array-like, sparse matrix} of shape (n_samples, n_features)***

The data matrix for which we want to get the predictions.

Returns:

y_pred : ***ndarray of shape (n_samples,)***

Vector containing the class labels for each sample.

Key Questions

Recall

- The formulation defined in the previous slide

$$\hat{y} = \operatorname*{argmax}_{y \in \mathcal{Y}} P(Y = y \mid \boldsymbol{X} = \boldsymbol{x}) \tag{3}$$

- The pipeline of text classification

$$\text{Text} \longrightarrow \boxed{\text{Numeric Vector } \boldsymbol{x}} \longrightarrow \boxed{\text{Classifier}} \longrightarrow \text{Label } y$$

Key Questions

Recall

- The formulation defined in the previous slide

$$\hat{y} = \underset{y \in \mathcal{Y}}{\operatorname{argmax}}\, P(Y = y \mid \boldsymbol{X} = \boldsymbol{x}) \tag{3}$$

- The pipeline of text classification

$$\text{Text} \longrightarrow \boxed{\text{Numeric Vector } \boldsymbol{x}} \longrightarrow \boxed{\text{Classifier}} \longrightarrow \text{Label } y$$

Building a text classifier is about answering the following two questions

1. How to represent a text as $\boldsymbol{x}$?

2. How to estimate $P(y \mid \boldsymbol{x})$?

Key Questions

Recall

- The formulation defined in the previous slide

$$\hat{y} = \underset{y \in \mathcal{Y}}{\operatorname{argmax}} P(Y = y \mid \boldsymbol{X} = \boldsymbol{x}) \tag{3}$$

- The pipeline of text classification

$$\text{Text} \longrightarrow \boxed{\text{Numeric Vector } \boldsymbol{x}} \longrightarrow \boxed{\text{Classifier}} \longrightarrow \text{Label } y$$

Building a text classifier is about answering the following two questions

1. How to represent a text as $\boldsymbol{x}$?
   - Bag-of-words representation
2. How to estimate $P(y \mid \boldsymbol{x})$?

Key Questions

Recall

- The formulation defined in the previous slide

$$\hat{y} = \operatorname*{argmax}_{y \in \mathcal{Y}} P(Y = y \mid \boldsymbol{X} = \boldsymbol{x}) \tag{3}$$

- The pipeline of text classification

$$\text{Text} \longrightarrow \boxed{\text{Numeric Vector } \boldsymbol{x}} \longrightarrow \boxed{\text{Classifier}} \longrightarrow \text{Label } y$$

Building a text classifier is about answering the following two questions

1. How to represent a text as $\boldsymbol{x}$?
   - Bag-of-words representation
2. How to estimate $P(y \mid \boldsymbol{x})$?
   - Logistic regression classifiers

Key Questions

Recall

- The formulation defined in the previous slide

$$\hat{y} = \underset{y \in \mathcal{Y}}{\operatorname{argmax}} P(Y = y \mid \boldsymbol{X} = \boldsymbol{x}) \tag{3}$$

- The pipeline of text classification

$$\text{Text} \longrightarrow \boxed{\text{Numeric Vector } \boldsymbol{x}} \longrightarrow \boxed{\text{Classifier}} \longrightarrow \text{Label } y$$

Building a text classifier is about answering the following two questions

1. How to represent a text as $\boldsymbol{x}$?
   - Bag-of-words representation
2. How to estimate $P(y \mid \boldsymbol{x})$?
   - Logistic regression classifiers
   - Neural network classifiers (next lecture)

Bag-of-Words Representation

Bag-of-Words Representation

Example Texts

Text 1: I love coffee.

Text 2: I don't like tea.

Bag-of-Words Representation

Example Texts

Text 1: I love coffee.

Text 2: I don't like tea.

Step I: convert a text into a collection of tokens (e.g., tokenization)

Tokenized Texts

Tokenized text 1: `I love coffee`

Tokenized text 2: `I don t like tea`

Bag-of-Words Representation

Example Texts

Text 1: I love coffee.

Text 2: I don't like tea.

Step I: convert a text into a collection of tokens (e.g., tokenization)

Tokenized Texts

Tokenized text 1: `I love coffee`

Tokenized text 2: `I don t like tea`

Step II: build a dictionary/vocabulary

Vocabulary

`{I love coffee don t like tea}`

Bag-of-Words Representations

Step III: based on the vocab, convert each text into a numeric representation as

Bag-of-Words Representations

	I	love	coffee	don	t	like	tea
$\boldsymbol{x}^{(1)} =$	[1	1	1	0	0	0	0]$^\top$
$\boldsymbol{x}^{(2)} =$	[1	0	0	1	1	1	1]$^\top$

Bag-of-Words Representations

Step III: based on the vocab, convert each text into a numeric representation as

Bag-of-Words Representations

	I	love	coffee	don	t	like	tea
$x^{(1)} =$	[1	1	1	0	0	0	0]$^\top$
$x^{(2)} =$	[1	0	0	1	1	1	1]$^\top$

The pipeline of text classification:

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

Bag-of-words
Representation

Implementation

In sklearn, CountVectorizer implements all the three steps in one function.

```
class sklearn.feature_extraction.text.CountVectorizer(*, input='content',
encoding='utf-8', decode_error='strict', strip_accents=None, lowercase=True,
preprocessor=None, tokenizer=None, stop_words=None,
token_pattern='(?u)\\b\\w\\w+\\b', ngram_range=(1, 1), analyzer='word',
max_df=1.0, min_df=1, max_features=None, vocabulary=None, binary=False,
dtype=<class 'numpy.int64'>)
```

[source]

Additional Steps of Building Vocab

1. Convert all characters to lowercase

$$\texttt{UVa}, \texttt{UVA} \longrightarrow \texttt{uva}$$

Additional Steps of Building Vocab

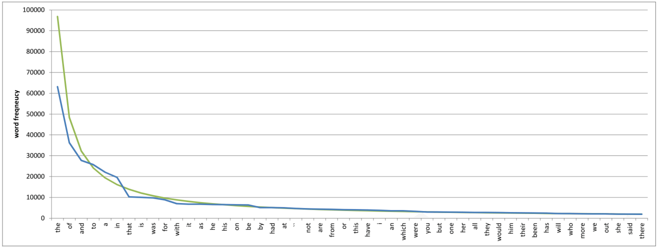
1. Convert all characters to lowercase

$$\texttt{UVa,UVA} \longrightarrow \texttt{uva}$$

Shall we always convert all words to lowercase?

`Apple` vs. `apple`

2. Map low frequency words to a special token ⟨`unk`⟩



Zipf's law: $\mathrm{freq}(w_t) \propto 1/r_t$

where $\mathrm{freq}(w_t)$ is the frequency of word w_t and r_t is the rank of this word

Information Embedded in BoW Representations

It is critical to keep in mind about what information is preserved in bag-of-words representations:

- Keep:
  - words in texts

Information Embedded in BoW Representations

It is critical to keep in mind about what information is preserved in bag-of-words representations:

- Keep:
  - words in texts
- Lose:
  - word order

```
I love coffee don t like tea
```

Information Embedded in BoW Representations

It is critical to keep in mind about what information is preserved in bag-of-words representations:

- Keep:
  - words in texts
- Lose:
  - word order

    ```
    I love coffee don t like tea
    ```

  - sentence boundary
  - sentence order
  - . . .

Case Study: Sentiment Analysis

A Dummy Predictor

Consider the following toy example (adding one more example to make it more interesting)

Tokenized Texts

	Text X	Label Y
Tokenized text 1	`I love coffee`	Positive
Tokenized text 2	`I don t like tea`	Negative
Tokenized text 3	`I like coffee`	Positive

[2] The evaluation of classifiers will be discussed in one of the future lectures.

A Dummy Predictor

Consider the following toy example (adding one more example to make it more interesting)

Tokenized Texts

	Text X	Label Y
Tokenized text 1	`I love coffee`	Positive
Tokenized text 2	`I don t like tea`	Negative
Tokenized text 3	`I like coffee`	Positive

What is the simplest classifier that we can constructed based on this "dataset"?

[2]The evaluation of classifiers will be discussed in one of the future lectures.

A Dummy Predictor

Consider the following toy example (adding one more example to make it more interesting)

Tokenized Texts

	Text X	Label Y
Tokenized text 1	`I love coffee`	POSITIVE
Tokenized text 2	`I don t like tea`	NEGATIVE
Tokenized text 3	`I like coffee`	POSITIVE

What is the simplest classifier that we can constructed based on this "dataset"?

- Predict every text as POSITIVE

[2] The evaluation of classifiers will be discussed in one of the future lectures.

A Dummy Predictor

Consider the following toy example (adding one more example to make it more interesting)

Tokenized Texts

	Text X	Label Y
Tokenized text 1	`I love coffee`	POSITIVE
Tokenized text 2	`I don t like tea`	NEGATIVE
Tokenized text 3	`I like coffee`	POSITIVE

What is the simplest classifier that we can constructed based on this "dataset"?

- Predict every text as POSITIVE
- 66.7% prediction accuracy on this dataset[2]

[2]The evaluation of classifiers will be discussed in one of the future lectures.

A Simple Predictor

Consider the following toy example, again

Tokenized Texts

Tokenized text 1: `I love coffee`

Tokenized text 2: `I don t like tea`

Tokenized text 3: `I like coffee`

What if we simply count the number of positive and negative words?

A Simple Predictor

Consider the following toy example, again

Tokenized Texts

Tokenized text 1: `I love coffee`

Tokenized text 2: `I don t like tea`

Tokenized text 3: `I like coffee`

What if we simply count the number of positive and negative words?

	`I`	`love`	`coffee`	`don`	`t`	`like`	`tea`
$\boldsymbol{x}^{(1)}$	[1	1	1	0	0	0	0]$^{\mathsf{T}}$
$\boldsymbol{w}_{\text{Pos}}$	[0	1	0	0	0	1	0]$^{\mathsf{T}}$
$\boldsymbol{w}_{\text{Neg}}$	[0	0	0	1	0	0	0]$^{\mathsf{T}}$

A Simple Predictor

Consider the following toy example, again

Tokenized Texts

Tokenized text 1: `I love coffee`

Tokenized text 2: `I don t like tea`

Tokenized text 3: `I like coffee`

What if we simply count the number of positive and negative words?

	`I`	`love`	`coffee`	`don`	`t`	`like`	`tea`
$x^{(1)}$	[1	1	1	0	0	0	0]$^{\top}$
w_{Pos}	[0	1	0	0	0	1	0]$^{\top}$
w_{Neg}	[0	0	0	1	0	0	0]$^{\top}$

The prediction of sentiment polarity can be formulated as the following

$$w_{\text{Pos}}^{\top} x = 1 > w_{\text{Neg}}^{\top} x = 0 \tag{4}$$

Another Example

The limitation of word counting

	I	love	coffee	don	t	like	tea
$\boldsymbol{x}^{(2)}$	[1	0	0	1	1	1	1]$^{\mathsf{T}}$
$\boldsymbol{w}_{\textsc{Pos}}$	[0	1	0	0	0	1	0]$^{\mathsf{T}}$
$\boldsymbol{w}_{\textsc{Neg}}$	[0	0	0	1	0	0	0]$^{\mathsf{T}}$

Another Example

The limitation of word counting

	I	love	coffee	don	t	like	tea
$\boldsymbol{x}^{(2)}$	[1	0	0	1	1	1	1]$^\mathsf{T}$
$\boldsymbol{w}_{\textsc{Pos}}$	[0	1	0	0	0	1	0]$^\mathsf{T}$
$\boldsymbol{w}_{\textsc{Neg}}$	[0	0	0	1	0	0	0]$^\mathsf{T}$

- Different words should contribute differently. e.g., `not` vs. `dislike`

Another Example

The limitation of word counting

	I	love	coffee	don	t	like	tea
$\boldsymbol{x}^{(2)}$	[1	0	0	1	1	1	1]$^{\mathsf{T}}$
$\boldsymbol{w}_{\text{Pos}}$	[0	1	0	0	0	1	0]$^{\mathsf{T}}$
$\boldsymbol{w}_{\text{Neg}}$	[0	0	0	1	0	0	0]$^{\mathsf{T}}$

- Different words should contribute differently. e.g., `not` vs. `dislike`
- Sentiment word lists are often incomplete

Example II: Positive

Din Tai Fung, every time I go eat at anyone of the locations around the King County area, I keep being reminded on why I have to keep coming back to this restaurant. $\cdots$

Logistic Regression

Linear Models

Directly modeling a linear classifier as

$$h_y(\boldsymbol{x}) = \boldsymbol{w}_y^\top \boldsymbol{x} + b_y \tag{5}$$

with

- $\boldsymbol{x} \in \mathbb{N}^V$: vector, bag-of-words representation
- $\boldsymbol{w}_y \in \mathbb{R}^V$: vector, classification weights associated with label y
- $b_y \in \mathbb{R}$: scalar, label bias in the training set y

Linear Models

Directly modeling a linear classifier as

$$h_y(\boldsymbol{x}) = \boldsymbol{w}_y^{\top}\boldsymbol{x} + b_y \tag{5}$$

with

- $\boldsymbol{x} \in \mathbb{N}^V$: vector, bag-of-words representation
- $\boldsymbol{w}_y \in \mathbb{R}^V$: vector, classification weights associated with label y
- $b_y \in \mathbb{R}$: scalar, label bias in the training set y

About Label Bias

Consider a case with highly-imbalanced examples, where we have 90 positive examples and 10 negative examples in the training set. With

$$b_{\text{Pos}} > b_{\text{Neg}},$$

a classifier can get 90% predictions correct without even resorting the texts.

Rewrite the linear decision function in the log probabilitic form

$$\log P(y \mid \boldsymbol{x}) \propto \underbrace{\boldsymbol{w}_y^\top \boldsymbol{x} + b_y}_{h_y(\boldsymbol{x})} \tag{6}$$

Logistic Regression

Rewrite the linear decision function in the log probabilitic form

$$\log P(y \mid \boldsymbol{x}) \propto \underbrace{\boldsymbol{w}_y^{\top}\boldsymbol{x} + b_y}_{h_y(\boldsymbol{x})} \tag{6}$$

or, the probabilistic form is

$$P(y \mid \boldsymbol{x}) \propto \exp(\boldsymbol{w}_y^{\top}\boldsymbol{x} + b_y) \tag{7}$$

Logistic Regression

Rewrite the linear decision function in the log probabilitic form

$$\log P(y \mid \boldsymbol{x}) \propto \underbrace{\boldsymbol{w}_y^\top \boldsymbol{x} + b_y}_{h_y(\boldsymbol{x})} \tag{6}$$

or, the probabilistic form is

$$P(y \mid \boldsymbol{x}) \propto \exp(\boldsymbol{w}_y^\top \boldsymbol{x} + b_y) \tag{7}$$

To make sure $P(y \mid \boldsymbol{x})$ is a valid definition of probability, we need to make sure $\sum_y P(y \mid \boldsymbol{x}) = 1$,

$$P(y \mid \boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_y^\top \boldsymbol{x} + b_y)}{\sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{w}_{y'}^\top \boldsymbol{x} + b_{y'})} \tag{8}$$

Alternative Form

Rewriting $\boldsymbol{x}$ and $\boldsymbol{w}$ as

- $\boldsymbol{x}^{\top} = [x_1, x_2, \cdots, x_V, 1]$
- $\boldsymbol{w}_y^{\top} = [w_1, w_2, \cdots, w_V, b_y]$

allows us to have a more concise form

$$P(y \mid \boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_y^{\top}\boldsymbol{x})}{\sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{w}_{y'}^{\top}\boldsymbol{x})} \tag{9}$$

Alternative Form

Rewriting $\boldsymbol{x}$ and $\boldsymbol{w}$ as

- $\boldsymbol{x}^\top = [x_1, x_2, \cdots, x_V, 1]$
- $\boldsymbol{w}_y^\top = [w_1, w_2, \cdots, w_V, b_y]$

allows us to have a more concise form

$$P(y \mid \boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_y^\top \boldsymbol{x})}{\sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{w}_{y'}^\top \boldsymbol{x})} \tag{9}$$

Comments:

- $\frac{\exp(a)}{\sum_{a'} \exp(a')}$ is the softmax function
- This form works with any size of $\mathcal{Y}$ — it does not have to be a binary classification problem.

Binary Classifier

Assume $\mathcal{Y} = \{\textsc{neg}, \textsc{pos}\}$, then the corresponding logistic regression classifier with $Y = \text{Pos}$ is

$$P(Y = \text{Pos} \mid \boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{w}^{\top}\boldsymbol{x})} \tag{10}$$

where $\boldsymbol{w}$ is the only parameter.

Binary Classifier

Assume $\mathcal{Y} = \{\textsc{neg}, \textsc{pos}\}$, then the corresponding logistic regression classifier with $Y = \textsc{Pos}$ is

$$P(Y = \textsc{Pos} \mid \boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{w}^{\top}\boldsymbol{x})} \tag{10}$$

where $\boldsymbol{w}$ is the only parameter.

- $P(Y = \textsc{Neg} \mid \boldsymbol{x}) = 1 - P(Y = \textsc{Pos} \mid \boldsymbol{x})$

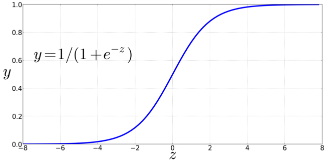
Binary Classifier

Assume $\mathcal{Y} = \{\textsc{neg}, \textsc{pos}\}$, then the corresponding logistic regression classifier with $Y = \textsc{Pos}$ is

$$P(Y = \textsc{Pos} \mid \boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{w}^{\top}\boldsymbol{x})} \tag{10}$$

where $\boldsymbol{w}$ is the only parameter.

- $P(Y = \textsc{Neg} \mid \boldsymbol{x}) = 1 - P(Y = \textsc{Pos} \mid \boldsymbol{x})$
- $\frac{1}{1+\exp(-z)}$ is the Sigmoid function



Implementation

Both the generic version and the binary version are implemented in the sklearn class: LogisticRegression

```
class sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False,
tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None,
random_state=None, solver='lbfgs', max_iter=100, multi_class='deprecated',
verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)
```

[source]

Logistic Regression (aka logit, MaxEnt) classifier.

Two Questions on Building LR Models

... of building a logistic regression classifier

$$P(y \mid \boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_y^\top \boldsymbol{x})}{\sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{w}_{y'}^\top \boldsymbol{x})} \tag{11}$$

- How to learn the parameters $\boldsymbol{W} = \{\boldsymbol{w}_y\}_{y \in \mathcal{Y}}$?

Two Questions on Building LR Models

... of building a logistic regression classifier

$$P(y \mid \boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_y^\top \boldsymbol{x})}{\sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{w}_{y'}^\top \boldsymbol{x})} \tag{11}$$

- How to learn the parameters $\boldsymbol{W} = \{\boldsymbol{w}_y\}_{y \in \mathcal{Y}}$?
- Can $\boldsymbol{x}$ be better than the bag-of-words representations?
  - Please revisit the CountVectorizer module

Review: (Log)-likelihood Function

With a collection of training examples $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{m}$, the likelihood function of $\{\boldsymbol{w}_y\}_{y\in\mathcal{Y}}$ is

$$L(\boldsymbol{W}) = \prod_{i=1}^{m} P(y^{(i)} \mid \boldsymbol{x}^{(i)}) \tag{12}$$

and the log-likelihood function is

$$\ell(\{\boldsymbol{w}_y\}) = \sum_{i=1}^{m} \log P(y^{(i)} \mid \boldsymbol{x}^{(i)}) \tag{13}$$

Log-likelihood Function of a LR Model

With the definition of a LR model

$$P(y \mid \boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_y^\top \boldsymbol{x})}{\sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{w}_{y'}^\top \boldsymbol{x})} \tag{14}$$

the log-likelihood function is

$$\begin{aligned}
\ell(\boldsymbol{W}) &= \sum_{i=1}^{m} \log P(y^{(i)} \mid \boldsymbol{x}^{(i)}) && (15)\\
&= \sum_{i=1}^{m} \left\{ \boldsymbol{w}_{y^{(i)}}^\top \boldsymbol{x}^{(i)} - \log \sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{w}_{y'}^\top \boldsymbol{x}^{(i)}) \right\} && (16)
\end{aligned}$$

Given the training examples $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{m}$, $\ell(\boldsymbol{W})$ is a function of $\boldsymbol{W} = \{\boldsymbol{w}_y\}$.

Optimization with Gradient

MLE is equivalent to minimize the Negative Log-Likelihood (NLL) as

$$\begin{aligned} \mathrm{NLL}(\boldsymbol{W}) &= -\ell(\boldsymbol{W}) \\ &= \sum_{i=1}^{m} \Big\{ -\boldsymbol{w}_{y^{(i)}}^{\mathsf{T}} \boldsymbol{x}^{(i)} + \log \sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{w}_{y'}^{\mathsf{T}} \boldsymbol{x}) \Big\} \end{aligned}$$

then, the parameter $\boldsymbol{w}_y$ associated with label y can be updated as

$$\boldsymbol{w}_y \leftarrow \boldsymbol{w}_y - \eta \cdot \frac{\partial \mathrm{NLL}(\{\boldsymbol{w}_y\})}{\partial \boldsymbol{w}_y}, \quad \forall y \in \mathcal{Y} \tag{17}$$

where η is called **learning rate**.

Optimization with Gradient (II)

Two questions answered by the update equation

(1) which direction?
(2) how far it should go?

[Jurafsky and Martin, 2019]

Optimization with Gradient (II)

Two questions answered by the update equation

(1) which direction?

(2) how far it should go?

$$\boldsymbol{w}_y \leftarrow \boldsymbol{w}_y - \underbrace{\eta}_{(2)} \cdot \underbrace{\frac{\partial \mathrm{NLL}(\{\boldsymbol{w}_y\})}{\partial \boldsymbol{w}_y}}_{(1)} \tag{18}$$

[Jurafsky and Martin, 2019]

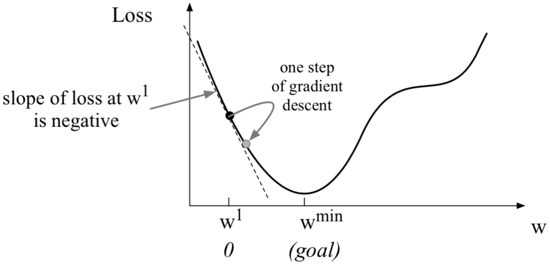
Optimization with Gradient (II)

Two questions answered by the update equation

(1) which direction?

(2) how far it should go?

$$w_y \leftarrow w_y - \underbrace{\eta}_{(2)} \cdot \underbrace{\frac{\partial \text{NLL}(\{w_y\})}{\partial w_y}}_{(1)} \tag{18}$$



[Jurafsky and Martin, 2019]

Training Procedure

Steps for parameter estimation, given the current parameter $\{\boldsymbol{w}_y\}$

1. Compute the derivative

$$\frac{\partial \text{NLL}(\{\boldsymbol{w}_y\})}{\partial \boldsymbol{w}_y}, \quad \forall y \in \mathcal{Y}$$

2. Update parameters with

$$\boldsymbol{w}_y \leftarrow \boldsymbol{w}_y - \eta \cdot \frac{\partial \text{NLL}(\{\boldsymbol{w}_y\})}{\partial \boldsymbol{w}_y}, \quad \forall y \in \mathcal{Y}$$

3. If not done, retrun to step 1

Procedure of Building a Classifier

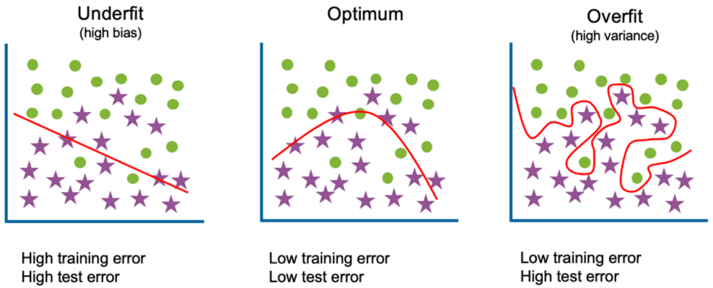
Review: the pipeline of text classification:

$$\text{Text} \longrightarrow \underset{\substack{\text{Bag-of-words}\\ \text{Representation}}}{\boxed{\text{Numeric Vector } \boldsymbol{x}}} \longrightarrow \underset{\substack{\text{Logistic}\\ \text{Regression}}}{\boxed{\text{Classifier}}} \longrightarrow \text{Category } y$$

Overfitting and L_2 Regularization

Overfitting: General Illustration

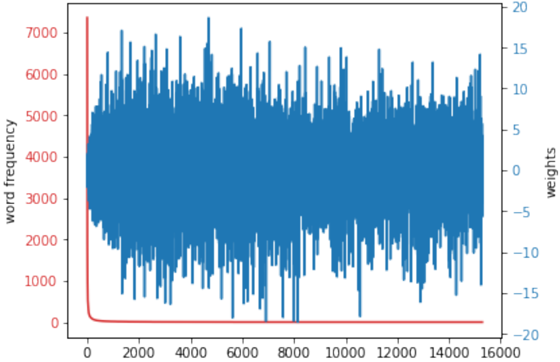
Three cases when building a classifier



Overfitting

In the demo code, we chose $\lambda = \frac{1}{C} = 0.001$ to approximate the case without regularization.

- Training accuracy: 99.89%
- Development accuracy: 52.21%



Classification Weights when Overfitting

Here are some word features and their classification weights from the previous model without regularization. Positive weights indicate the word feature contribute to positive sentiment classification and negative weights indicate the opposite contribution

	interesting	pleasure	boring	zoe	write	workings
Without Reg	0.011	-5.63	1.80	-5.68	-8.20	14.16

Classification Weights when Overfitting

Here are some word features and their classification weights from the previous model without regularization. Positive weights indicate the word feature contribute to positive sentiment classification and negative weights indicate the opposite contribution

	`interesting`	`pleasure`	`boring`	`zoe`	`write`	`workings`
Without Reg	0.011	-5.63	1.80	-5.68	-8.20	14.16

- NEGATIVE: `woody allen can write and deliver a one liner as well as anybody .`

Classification Weights when Overfitting

Here are some word features and their classification weights from the previous model without regularization. Positive weights indicate the word feature contribute to positive sentiment classification and negative weights indicate the opposite contribution

	`interesting`	`pleasure`	`boring`	`zoe`	`write`	`workings`
Without Reg	0.011	-5.63	1.80	-5.68	-8.20	14.16

- NEGATIVE: `woody allen can write and deliver a one liner as well as anybody .`
- POSITIVE: `soderbergh , like kubrick before him , may not touch the planet 's skin , but understands the workings of its spirit .`

L_2 Regularization

The commonly used regularization trick is the L_2 regularization. For that, we need to redefine the objective function of LR by adding an additional item

$$\text{Loss}(\boldsymbol{W}) = \underbrace{\sum_{i=1}^{m} \left\{ -\boldsymbol{w}_{y^{(i)}}^{\top} \boldsymbol{x}^{(i)} + \log \sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{w}_{y'}^{\top} \boldsymbol{x}^{(i)}) \right\}}_{\text{NLL}} \tag{19}$$

L_2 Regularization

The commonly used regularization trick is the L_2 regularization. For that, we need to redefine the objective function of LR by adding an additional item

$$\text{Loss}(\boldsymbol{W}) = \underbrace{\sum_{i=1}^{m} \left\{ -\boldsymbol{w}_{y^{(i)}}^{\top} \boldsymbol{x}^{(i)} + \log \sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{w}_{y'}^{\top} \boldsymbol{x}^{(i)}) \right\}}_{\text{NLL}} + \underbrace{\frac{\lambda}{2} \cdot \sum_{y \in \mathcal{Y}} \|\boldsymbol{w}_y\|_2^2}_{L_2 \text{ reg}} \tag{19}$$

- λ is the regularization parameter

L_2 Regularization in Gradient Descent

- The gradient of the loss function

$$\frac{\partial \text{Loss}(\boldsymbol{W})}{\partial \boldsymbol{w}_y} = \frac{\partial \text{NLL}(\boldsymbol{W})}{\partial \boldsymbol{w}_y} + \lambda \boldsymbol{w}_y \tag{20}$$

L_2 Regularization in Gradient Descent

- The gradient of the loss function

$$\frac{\partial \mathrm{Loss}(\boldsymbol{W})}{\partial \boldsymbol{w}_y} = \frac{\partial \mathrm{NLL}(\boldsymbol{W})}{\partial \boldsymbol{w}_y} + \lambda \boldsymbol{w}_y \tag{20}$$

- To minimize the loss, we need update the parameter as

$$\boldsymbol{w}_y \quad \leftarrow \quad \boldsymbol{w}_y - \eta\left(\frac{\partial \mathrm{NLL}(\boldsymbol{W})}{\partial \boldsymbol{w}_y} + \lambda \boldsymbol{w}_y\right) \tag{21}$$

L_2 Regularization in Gradient Descent

- The gradient of the loss function

$$\frac{\partial \text{Loss}(\boldsymbol{W})}{\partial \boldsymbol{w}_y} = \frac{\partial \text{NLL}(\boldsymbol{W})}{\partial \boldsymbol{w}_y} + \lambda \boldsymbol{w}_y \tag{20}$$

- To minimize the loss, we need update the parameter as

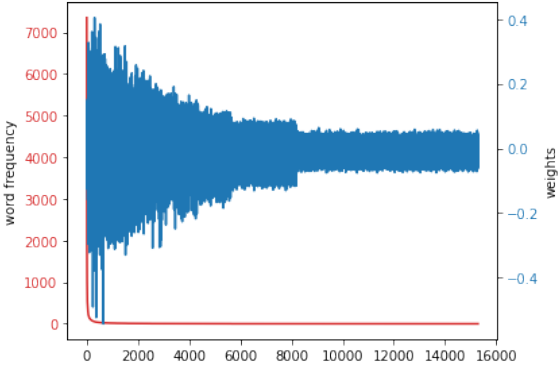
$$\begin{aligned} \boldsymbol{w}_y \quad &\leftarrow \quad \boldsymbol{w}_y - \eta\left(\frac{\partial \text{NLL}(\boldsymbol{W})}{\partial \boldsymbol{w}_y} + \lambda \boldsymbol{w}_y\right) \\ &= \quad (1 - \eta\lambda) \cdot \boldsymbol{w}_y - \eta \frac{\partial \text{NLL}(\boldsymbol{W})}{\partial \boldsymbol{w}_y} \end{aligned} \tag{21}$$

- Depending on the strength (value) of λ, the regularization term tries to keep the parameter values close to 0, which to some extent can help avoid overfitting

Learning with Regularization

We chose $\lambda = \frac{1}{C} = 10^2$

- Training accuracy: 62.54%
- Development accuracy: 63.17%



Classification Weights with Regularization

With regularization, the classification weights make more sense to us

	`interesting`	`pleasure`	`boring`	`zoe`	`write`	`workings`
Without Reg	0.011	-5.63	1.80	-5.68	-8.20	14.16
With Reg	0.16	0.36	-0.21	-0.057	-0.066	0.040

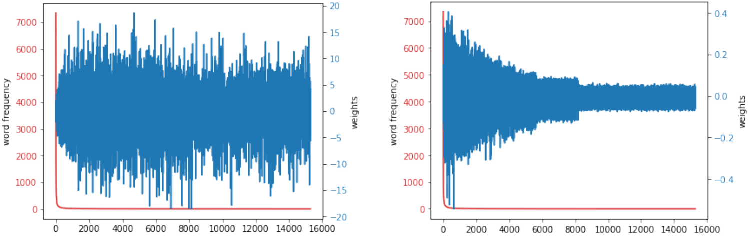
Classification Weights with Regularization

With regularization, the classification weights make more sense to us

	`interesting`	`pleasure`	`boring`	`zoe`	`write`	`workings`
Without Reg	0.011	-5.63	1.80	-5.68	-8.20	14.16
With Reg	0.16	0.36	-0.21	-0.057	-0.066	0.040

Regularization for Avoiding Overfitting

Reduce the correlation between class label and some noisy features.



(a) Overfitting (b) A better model

A similar explanation can be applied to more advanced classifiers, such as neural networks.

Reference

Jurafsky, D. and Martin, J. (2019).
Speech and language processing.

Pang, B., Lee, L., and Vaithyanathan, S. (2002).
Thumbs up?: sentiment classification using machine learning techniques.
In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics.