Text Classification

+

Machine Learning Review

Alexander Rush

January 21, 2016

Contents

Application: Spam Detection

Application: Topic Detection

Application: Sentiment Analysis

Why?

- Easy problem.
- Surface-level

Task: Text Classification

Given a sentence determine its class,

Bad Sentences

Unfortunately the story and the actors are served with a hack script. A sentimental mess that never rings true. This 100-minute movie only has about 25 minutes of decent material. Here, common sense flies out the window, along with the hail of bullets, none of which ever seem to hit Sascha.

Good Sentences

A thoughtful, provocative, insistently humanizing film. Occasionally melodramatic, it's also extremely effective. Guaranteed to move anyone who ever shook, rattled, or rolled.

Contents

Preliminary Notation

- $\mathbf{b}, \mathbf{m}$; bold letters for vectors.
- $\mathbf{B}, \mathbf{M}$; bold capital letters for matrices.
- $\mathcal{B}, \mathcal{M}$; script-case for sets.
- B, M; capital letters for constants and random variables.
- b_i, x_i; lower case for scalars or indexing into vectors.

- $\delta(i)$; one-hot vector at position i

$$\delta(2) = [0; 1; 0; \dots]$$

- $\mathbf{1}(x = y)$; indicator 1 if $x = y$, o.w. 0

Text Classification

How do we do this?

1. First extract information from the sentence.
2. Use this to construct a *representation*
3. Classify this vector is the set of possible outputs.

Input Representation

- How do we convert an input sentence into a usable mathematical representation?
- Main focus of this class, representation of language
- Point in coming lectures: *sparse* vs. *dense* representations

Sparse Features

- Define $\mathcal{F}$ to be a discrete set of predefined features.
- For a given sentence, let $f_1 \in \mathcal{F}, \ldots, f_k \in \mathcal{F}$ be the relevant features. Typically $k << |\mathcal{F}|$.
- Define the sparse representation of the input

$$\mathbf{x} = \sum_{i=1}^{k} \delta(f_i)$$

- $\mathbf{x} \in \mathbb{R}^{1 \times d_{\mathrm{in}}}$; input representation

Features 1: Sparse Bag-of-Words Features

Representation is indicators of input words,

- $\mathcal{F}$; the vocabulary of the language.
- $\mathbf{x} = \sum_i \delta(f_i)$

Example: Movie Sentiment

`A sentimental mess`

$$\mathbf{x} = v(\texttt{word:A}) + v(\texttt{word:sentimental}) + v(\texttt{word:mess})$$

$$\mathbf{x}^\top = \begin{bmatrix} 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix} = \begin{array}{cl} \left[\begin{matrix} 1 \\ \vdots \\ 1 \\ 1 \end{matrix}\right] & \begin{matrix} \texttt{word:A} \\ \vdots \\ \texttt{word:mess} \\ \texttt{word:sentimental} \end{matrix} \end{array}$$

Features 2: Sparse Word Properties

Representation can use specific aspects of text.

- $\mathcal{F}$; Spelling, inner-word capitals, trigger words, etc.
- $\mathbf{x} = \sum_i \delta(f_i)$

Example: Spam Classification

```
Your diploma puts a UUniversity Job Placement Counselor
at your disposal.
```

$$\mathbf{x} = v(\texttt{misspelling}) + v(\texttt{capital}) + v(\texttt{word:diploma}) + \ldots$$

$$\mathbf{x}^\top = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} = \begin{array}{l} \left[\begin{matrix} 1 \\ \vdots \\ 1 \\ 1 \end{matrix}\right] \end{array} \begin{array}{l} \texttt{misspelling} \\ \vdots \\ \texttt{capital} \\ \texttt{word:diploma} \end{array}$$

Output Representation

- How do we handle the output representation?
- We will use a one-hot output encoding (may be slightly different).
- In future lectures, efficiency of output encoding.

Output Classes

- $\mathcal{C} = \{1, \ldots, d_{\text{out}}\}$; possible output classes
- $c \in \mathcal{C}$; the true output class
- $\mathbf{y} = \delta(c) \in \mathbb{R}^{1 \times d_{\text{in}}}$; one-hot output representation
- Note: when classes are words, we call them *word types*.

Output Form: Binary Classification

Examples: spam/not-spam, good review/bad review, relevant/irrelevant document, many others.

- $d_{\text{out}} = 2$; two possible classes
- In our notation,

$$\begin{aligned} c = 1 \quad \mathbf{y} &= \begin{bmatrix} 1 & 0 \end{bmatrix} \text{ vs.} \\ c = 2 \quad \mathbf{y} &= \begin{bmatrix} 0 & 1 \end{bmatrix} \end{aligned}$$

- Can also use a single output *sign* representation with $d_{\text{out}} = 1$

Output Form: Multiclass Classification

Examples: Yelp stars, etc.

- $d_{\text{out}} = 5$; for examples
- In our notation, one star, two star...

$$\begin{aligned} c = 1 \quad \mathbf{y} &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix} \text{ vs.} \\ c = 2 \quad \mathbf{y} &= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \end{bmatrix} \ldots \end{aligned}$$

Examples: Word Prediction (Unit 3)

- $d_{\text{out}} > 100,000$;
- In our notation, $\mathcal{C}$ is vocabulary and each c is a word.

$$\begin{aligned} c = 1 \quad \mathbf{y} &= \begin{bmatrix} 1 & 0 & 0 & 0 & \ldots & 0 \end{bmatrix} \text{ vs.} \\ c = 2 \quad \mathbf{y} &= \begin{bmatrix} 0 & 1 & 0 & 0 & \ldots & 0 \end{bmatrix} \ldots \end{aligned}$$

Evaluation

- Consider evaluating accuracy on outputs $\mathbf{y}_1, \dots, \mathbf{y}_n$.
- Given a decisions $\hat{c}_1 \dots \hat{c}_n$ we measure accuracy as,

$$\sum_{i=1}^{n} \frac{\mathbf{1}(\delta(\hat{c}_i) = \mathbf{y}_i)}{n}$$

- Simplest of several different metrics we will explore in the class.

Contents

Supervised Machine Learning

Let,

- $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)$; supervised data
- $\mathbf{x}_i \in \mathbb{R}^{1 \times d_{\mathrm{in}}}$; input representations
- $\mathbf{y}_i \in \mathbb{R}^{1 \times d_{\mathrm{out}}}$; gold output representations (one-hot vectors)

Goal: Learn a classifier from input to output classes $\mathcal{C}$.

(Practically, store design matrix $\mathbf{X} \in \mathbb{R}^{n \times d_{\mathrm{in}}}$ and output classes.)

Review: Experimental Setup

- Data is split into three parts training, validation, and test.
- Experiments are all run on training and validation, test is final output.
- (For assignments we will distribute training and validation, and only give **x** for test.)

For small text classification data sets,

- Even more careful. Results are reported with K-fold cross-validation.
    1. Split into K folds (equal splits).
    2. For each fold, train on other K-1 folds, test on current fold.

Linear Models for Classification

Define a linear model as,

- Model,

$$\hat{\mathbf{y}} = \mathbf{x}\mathbf{W} + \mathbf{b}$$

- $\mathbf{W} \in \mathbb{R}^{d_{\mathrm{in}} \times d_{\mathrm{out}}}, \mathbf{b} \in \mathbb{R}^{1 \times d_{\mathrm{out}}}$; model parameters
- Note $\hat{\mathbf{y}}$ is **not** one-hot, informally "score" vector.

The decision rule is then,

$$\hat{c} = \arg\max_{i \in \mathcal{C}} \hat{y}_i$$

Interpreting Linear Models

Parameters give scores to possible outputs,

- $W_{f,i}$ is the score for feature f under class i
- b_c is a prior score for class i
- y_i is the total score for class i
- $\hat{c}$ is highest scoring class under the linear model.

Examples:

- For feature score,

$$[\beta_1, \beta_2] = \delta(\texttt{word:dreadful})\mathbf{W},$$

You would expect $\beta_2 > \beta_1$ (assuming $c = 2$ is "positive").

Probabilistic Linear Models

We can begin by interpreting prediction probabilistically,

- Let output be a random variable Y, with sample space $\mathcal{C}$.
- Representation be a random vector X.
- Interested in estimating parameters θ of,

$$P(Y|X;\theta)$$

We will be informal and use $p(\mathbf{y} = c|\mathbf{x})$ for $P(Y = c|X = \mathbf{x})$.

Probabilistic Linear Models

- $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)$; supervised data
- Select parameters to maximize likelihood of training data.

$$\mathcal{L}(\theta) = -\sum_{i=1}^{n} \log p(\mathbf{y}_i | \mathbf{x}_i; \theta)$$

  For linear models $\theta = (\mathbf{W}, \mathbf{b})$

- Do this by minimizing negative log-likelihood.

$$\arg\min_{\theta} \mathcal{L}(\theta)$$

Properties: Probabilistic Model

- Gives more than
- Allows variant decision rules (for instance minimum bayes' risk)
- May be less efficient to train.
-

Probabilistic Factorization

Reminder, Bayes Rule

$$p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{y})p(\mathbf{y})}{p(\mathbf{x})}$$

Can be instead written (with $\propto$ as normalizing factor)

$$p(\mathbf{y}|\mathbf{x}) \propto p(\mathbf{x}|\mathbf{y})p(\mathbf{y})$$

For NLL, $p(\mathbf{x})$ doesn't matter, estimate likelihood $p(\mathbf{x}|\mathbf{y})$ and prior $p(\mathbf{y})$.

For a sparse model, we write,

$$p(x_{f_1} = 1, \ldots, x_{f_k} = 1|\mathbf{y} = c)p(\mathbf{y} = c)$$

Naive Bayes Assumption

$$\begin{aligned}
& p(x_{f_1} = 1, \ldots, x_{f_k} = 1 | \mathbf{y} = c) p(\mathbf{y} = c) = \\
& \prod_{i=1}^{k} p(x_{f_i} = 1 | x_{f_1} = 1, \ldots, x_{f_{i-1}} = 1, \mathbf{y} = c) p(\mathbf{y} = c) \approx \\
& \prod_{i=1}^{k} p(x_{f_i} | \mathbf{y}) p(\mathbf{y})
\end{aligned}$$

First is by chain-rule, second is by assumption (Naive bayes).

Multinomial Model

Brief aside,

- $P(S;\theta)$; parameterized as a multinomial distribution (multinoulli in)
- Minimizing NLL for multinomial for data has a closed-form.

$$\theta_s = \sum_{i=1}^{n} \frac{\mathbf{1}(s_i = s)}{n}$$

$$P(S = s; \theta) = \theta_s$$

- Exercise: Derive this by minimizing $\mathcal{L}$.

Multinomial Naive Bayes

- Both the prior $p(\mathbf{y} = c)$ and the likelihood $p(\mathbf{x}|\mathbf{y})$ are parameterized as multinomial (multinoulli) distributions.
- Fit prior as,

$$p(\mathbf{y} = c) = \sum_{i=1}^{n} \frac{1(\mathbf{y}_i = c)}{n}$$

- Fit likelihood as,
  - Let

$$F_{f,c} = \sum_{i=1}^{n} \mathbf{1}(\mathbf{y}_i = c)\mathbf{1}(x_{i,f} = 1) \text{ forall } c \in \mathcal{C}, f \in \mathcal{F}$$

  - Then,

$$p(x_f|\mathbf{y} = c) = \frac{F_{f,c}}{\sum_{f' \in \mathcal{F}} F_{f',c}}$$

How does this become a linear classifier?

$$W_{f,c} = \log p(x_f = 1|\mathbf{y} = c)$$

Digression: Zipf's Law

Word features ...

Laplacian Smoothing

- In order to handle the long tail of words. Add a value of α to each element in the sample space before normalization. Where α is a smoothing hyperparameter
- $$\theta_s = \frac{\alpha + \sum_{i=1}^{n} \mathbf{1} s_i = s}{\alpha |\mathcal{S}| n}$$
- (Similar to Dirichlet prior in a Bayesian interpretation.)

For naive Bayes:

- $$\hat{\mathbf{F}} = \alpha + F$$

Naive Bayes In Practice

- Incredibly fast

The Softmax

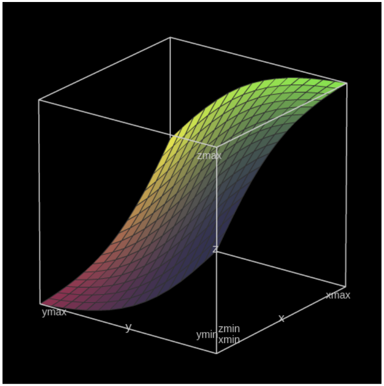
Instead of factoring into multinomial, use a soft-max to force a distribution

$$\text{softmax}(\hat{\mathbf{y}}) = \frac{\exp(\hat{\mathbf{y}})}{\sum_{c \in \mathcal{C}} \exp(\hat{y})}$$

- Exercise: Confirm always gives a distribution.
- Note denominator is known as the *partition function*.

Why is it called softmax?

$$\text{softmax}(\hat{\mathbf{y}}) = \frac{\exp(x)}{\exp(x) + \exp(y)}$$



$$\arg\max(\hat{\mathbf{y}}) = \mathbf{1}(x > y)$$

Multiclass logistic regression

$$\hat{\mathbf{y}} = \mathbf{x}\mathbf{W} + \mathbf{b}$$

$$p(\mathbf{y} = c | \mathbf{x}; \theta) = \frac{\exp(\hat{\mathbf{y}_c})}{\sum_{c' \in \mathcal{C}} \exp(\hat{\mathbf{y}_{c'}})}$$

Special Case: Binary Classification

For binary classification this is often

1. $$y = \mathbf{xW} + b$$

$$\text{softmax}() = \frac{\exp(y_1)}{(\exp(y_1) + \exp(y_2))} = \frac{1}{1 + \exp -(y_1 - y_2)} = \sigma(y_1 - y_2)$$

Logistic sigmoid function

$$\sigma(t) = \frac{1}{1 + \exp -t}$$

Which is standard for logistic regression.

A Model with Many Names

- Multinomial logistic regression
- Log-Linear (particularly in NLP)
- Softmax Regression
- Max-Entropy ()

Benefits of Logistic Regression

-
- Fitting parameters is much more difficult.

Models similar to LR will be the main focus of this class.

Fitting Parameters

Recall probabilistic objective is:

$$\mathcal{L}(\theta) = -\sum_{i=1}^{n} \log p(\mathbf{y}_i | \mathbf{x}_i; \theta)$$

And the distribution is parameterized as a softmax,

$$\begin{aligned} \log p(\mathbf{y} = c | \mathbf{x}; \theta) &= \log \operatorname{softmax}(\mathbf{x}\mathbf{W} + \mathbf{b})_c \\ &= \hat{y}_c - \log \sum_{c' \in \mathcal{C}} \exp(\hat{y}_{c'}) \end{aligned}$$

However, this is much harder to solve, no closed-form.

Computing Parameter Gradients

Consider one example $(\mathbf{x}, \mathbf{y})$ maximizing $\log p(\mathbf{y}|\mathbf{x};\theta)$,

1. Compute scores $\hat{\mathbf{y}} = \mathbf{x}\mathbf{W} + \mathbf{b}$
2. Compute log softmax of scores, $\log \text{softmax}(\hat{\mathbf{y}})$
3. Compute gradient of $\mathcal{L}$ with respect to scores $\hat{\mathbf{y}}$ for all $c \in \mathcal{C}$

$$\frac{\partial \mathcal{L}}{\partial y_c} = -\mathbf{1}(\mathbf{y} = \delta(c)) - \frac{\exp(\hat{y}_c)}{\sum_{c'} \exp(\hat{y}_{c'})}$$

4. Compute gradient of $\mathbf{b}$ for all $c \in \mathcal{C}$ (Chain rule),

$$\frac{\partial \mathcal{L}}{\partial b_c} = \frac{\partial \mathcal{L}}{\partial y_c}$$

5. Compute gradient of $\mathbf{W}$ for all $c \in \mathcal{C}, f \in \mathcal{F}$,

$$\frac{\partial \mathcal{L}}{\partial W_{f,c}} = \frac{\partial \mathcal{L}}{\partial y_c}$$

Gradient-Based Optimization: SGD

```
procedure SGD
    while training criterion is not met do
        Sample a training example $\mathbf{x}_i, \mathbf{y}_i$
        Compute the loss $\mathcal{L}(\hat{\mathbf{y}}_i, \mathbf{y}_i; \theta)$
        Compute gradients $\hat{\mathbf{g}}$ of $\mathcal{L}(\hat{\mathbf{y}}_i, \mathbf{y}_i; \theta)$ with respect to $\theta$
        $\theta \leftarrow \theta + \eta_k \hat{\mathbf{g}}$
    end while
    return $\theta$
end procedure
```

Gradient-Based Optimization: Minibatch SGD

```
**while** training criterion is not met **do**
    Sample a minibatch of $m$ examples $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_m, \mathbf{y}_m)$
    $\hat{\mathbf{g}} \leftarrow 0$
    **for** $i = 1 \text{ to } m$ **do**
        Compute the loss $\mathcal{L}(\hat{\mathbf{y}}_i, \mathbf{y}_i; \theta)$
        Compute gradients $\mathbf{g}'$ of $\mathcal{L}(\hat{\mathbf{y}}_i, \mathbf{y}_i; \theta)$ with respect to $\theta$
        $\hat{\mathbf{g}} \leftarrow \hat{\mathbf{g}} + \frac{1}{m}\mathbf{g}'$
    **end for**
    $\theta \leftarrow \theta + \eta_k \hat{\mathbf{g}}$
**end while**
**return** $\theta$
```

Softmax Notes: Calculating Log-Sum-Exp

- Calculating $\log \sum_{c' \in \mathcal{C}} \exp(\hat{y}_{c'}$ directly numerical issues.
- Instead $\log \sum_{c' \in \mathcal{C}} \exp(\hat{y}_{c'} - M) + M$ where $M = \max_{c' \in \mathcal{C}} \hat{y}_c'$

Softmax Notes: Regularization

$$\mathcal{L}(\theta) = -\sum_{i=1}^{n} \log p(\mathbf{y}_i | \mathbf{x}_i; \theta) + ||\theta||_2^2$$

Non-Probabilistic Classification

What if we just try to directly find $\mathbf{W}$ and $\mathbf{b}$?

$$\mathbf{y} = \mathbf{x}\mathbf{W} + \mathbf{b}$$

Need to find a suitable training objective.

Hinge/SVM Objective

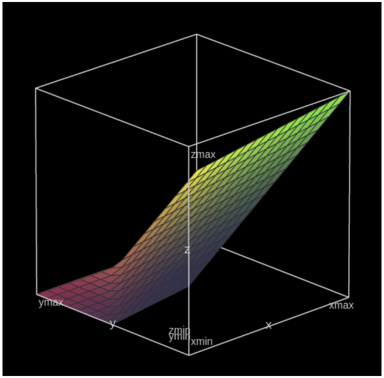
For a given example $\mathbf{y}_i$,

- Let c be defined as gold class $y_{i,c} = 1$
- Let $\bar{c}$ be defined as the highest scoring non-gold class

$$\bar{c} = \arg\max_{c \in \mathcal{C} \setminus \{c\}} \hat{y}_c$$

$$\mathcal{L}_{margin}(\theta) = \sum_{i=1}^{n} \max\{0, 1 - \hat{y}_{i,c} + \hat{y}_{i,\bar{c}}\} + ||\theta||_2^2$$

$$hinge(\hat{\mathbf{y}}) = (\max\{0, 1 - (y - x)\})$$



Margin

$$\frac{\delta \mathcal{L}}{\delta \theta_i} = \sum_i \mathbf{y}_i - \mathbf{y}_{\hat{c}}$$

Current State of the Art