Natural Language Processing (CSE 447/547M): Text Classification, Continued

Noah Smith

University of Washington
nasmith@cs.washington.edu

January 28, 2019

Administrivia

How to ask questions outside lecture:

- Use ~~Canvas~~ Piazza! Read ~~Canvas~~ Piazza first to see if your question has already been answered!
- If you want confidentiality, or aren't yet registered, then use the instructors' mailing list (but note that TAs will all see it)

Try not to sweat quiz grades. We'll drop (at least) your lowest, and in the final calculation, most of the credit is for the attempt.

Text Classification

Input: a piece of text $\boldsymbol{x} \in \mathcal{V}^\dagger$, usually a document (r.v. $\boldsymbol{X}$) Output: a label from a finite set $\mathcal{L}$ (r.v. L)

Standard line of attack:

1. Human experts label some data.
2. Feed the data to a supervised machine learning algorithm that constructs an automatic classifier $\mathrm{classify} : \mathcal{V}^\dagger \to \mathcal{L}$
3. Apply $\mathrm{classify}$ to as much data as you want!

Note: we assume the texts are segmented already, even the new ones.

Text Classification: Examples

- Library-like subjects (e.g., the Dewey decimal system)
- News stories: politics vs. sports vs. business vs. technology ...
- Reviews of films, restaurants, products: postive vs. negative
- Author attributes: identity, political stance, gender, age, ...
- Email, arXiv submissions, etc.: spam vs. not
- What is the reading level of a piece of text?
- How influential will a scientific paper be?
- Will a piece of proposed legislation pass?

Closely related: relevance to a query.

Last Time

- Evaluation of text classifiers (accuracy, precision, recall, F_1)
- Statistical significance

Features in Text Classification

Running example: $\boldsymbol{x}$ = "The vodka was great, but don't touch the hamburgers."

A different representation of the text sequence r.v. $\boldsymbol{X}$: feature r.v.s.

For $j \in \{1, \ldots, d\}$, let F_j be a discrete random variable taking a value in $\mathcal{F}_j$.

Features in Text Classification

Running example: $\boldsymbol{x}$ = "The vodka was great, but don't touch the hamburgers."

A different representation of the text sequence r.v. $\boldsymbol{X}$: feature r.v.s.

For $j \in \{1, \ldots, d\}$, let F_j be a discrete random variable taking a value in $\mathcal{F}_j$.

- Often, these are term (word and perhaps n-gram) frequencies.
  E.g., $f_{\text{hamburgers}}(\boldsymbol{x}) = 1$, $f_{\text{the}}(\boldsymbol{x}) = 2$, $f_{\text{delicious}}(\boldsymbol{x}) = 0$, $f_{\text{don't touch}}(\boldsymbol{x}) = 1$.

Features in Text Classification

Running example: $\boldsymbol{x}$ = "The vodka was great, but don't touch the hamburgers."

A different representation of the text sequence r.v. $\boldsymbol{X}$: feature r.v.s.

For $j \in \{1, \ldots, d\}$, let F_j be a discrete random variable taking a value in $\mathcal{F}_j$.

- Often, these are term (word and perhaps n-gram) frequencies.
  E.g., $f_{\text{hamburgers}}(\boldsymbol{x}) = 1$, $f_{\text{the}}(\boldsymbol{x}) = 2$, $f_{\text{delicious}}(\boldsymbol{x}) = 0$, $f_{\text{don't touch}}(\boldsymbol{x}) = 1$.
- Can also be word "presence" features.
  E.g., $f_{\text{hamburgers}}(\boldsymbol{x}) = 1$, $f_{\text{the}}(\boldsymbol{x}) = 1$, $f_{\text{delicious}}(\boldsymbol{x}) = 0$, $f_{\text{don't touch}}(\boldsymbol{x}) = 1$.

Features in Text Classification

Running example: $\boldsymbol{x}$ = "The vodka was great, but don't touch the hamburgers."

A different representation of the text sequence r.v. $\boldsymbol{X}$: feature r.v.s.

For $j \in \{1, \ldots, d\}$, let F_j be a discrete random variable taking a value in $\mathcal{F}_j$.

- Often, these are term (word and perhaps n-gram) frequencies.
  E.g., $f_{\text{hamburgers}}(\boldsymbol{x}) = 1$, $f_{\text{the}}(\boldsymbol{x}) = 2$, $f_{\text{delicious}}(\boldsymbol{x}) = 0$, $f_{\text{don't touch}}(\boldsymbol{x}) = 1$.
- Can also be word "presence" features.
  E.g., $f_{\text{hamburgers}}(\boldsymbol{x}) = 1$, $f_{\text{the}}(\boldsymbol{x}) = 1$, $f_{\text{delicious}}(\boldsymbol{x}) = 0$, $f_{\text{don't touch}}(\boldsymbol{x}) = 1$.
- Transformations on word frequencies: logarithm, idf weighting

$$\forall v \in \mathcal{V}, \text{idf}(v) = \log \frac{n}{|i : c_{\boldsymbol{x}_i}(v) > 0|}$$

Features in Text Classification

Running example: $\boldsymbol{x}$ = "The vodka was great, but don't touch the hamburgers."

A different representation of the text sequence r.v. $\boldsymbol{X}$: feature r.v.s.

For $j \in \{1, \ldots, d\}$, let F_j be a discrete random variable taking a value in $\mathcal{F}_j$.

- Often, these are term (word and perhaps n-gram) frequencies.
  E.g., $f_{\text{hamburgers}}(\boldsymbol{x}) = 1$, $f_{\text{the}}(\boldsymbol{x}) = 2$, $f_{\text{delicious}}(\boldsymbol{x}) = 0$, $f_{\text{don't touch}}(\boldsymbol{x}) = 1$.
- Can also be word "presence" features.
  E.g., $f_{\text{hamburgers}}(\boldsymbol{x}) = 1$, $f_{\text{the}}(\boldsymbol{x}) = 1$, $f_{\text{delicious}}(\boldsymbol{x}) = 0$, $f_{\text{don't touch}}(\boldsymbol{x}) = 1$.
- Transformations on word frequencies: logarithm, idf weighting

$$\forall v \in \mathcal{V}, \text{idf}(v) = \log \frac{n}{|i : c_{\boldsymbol{x}_i}(v) > 0|}$$

- Disjunctions of terms
  - Clusters
  - Task-specific lexicons

Probabilistic Classification

Let $\boldsymbol{f} = \langle f_1, \ldots, f_d \rangle$ denote the feature vector for the input text ($\boldsymbol{x}$).

Classification rule:

$$\begin{aligned}
\text{classify}(\boldsymbol{f}) &= \operatorname*{argmax}_{\ell \in \mathcal{L}} p(L = \ell \mid \boldsymbol{F} = \boldsymbol{f}) \\
&= \operatorname*{argmax}_{\ell \in \mathcal{L}} \frac{p(\ell, \boldsymbol{f})}{p(\boldsymbol{f})} \\
&= \operatorname*{argmax}_{\ell \in \mathcal{L}} p(\ell, \boldsymbol{f})
\end{aligned}$$

Naïve Bayes Classifier

$$p(L = \ell, F_j = f_1, \ldots, F_d = f_d) = p(\ell) \prod_{j=1}^{d} p(F_j = f_j \mid \ell)$$

$$= \pi_\ell \prod_{j=1}^{d} \theta_{f_j|j,\ell}$$

Parameters:

- $\boldsymbol{\pi} \in \triangle^{|\mathcal{L}|}$, the "class prior"
- For each feature function j and label ℓ, a distribution over values $\boldsymbol{\theta}_{*|j,\ell} \in \triangle^{|\mathcal{F}_j|}$

The "bag of words" version of naïve Bayes:

$$F_j = X_j$$

$$p(\ell, \boldsymbol{x}) = \pi_\ell \prod_{j=1}^{|\boldsymbol{x}|} \theta_{x_j|\ell}$$

Naïve Bayes: Remarks

- Estimation by (smoothed) relative frequency estimation: easy!

Naïve Bayes: Remarks

- Estimation by (smoothed) relative frequency estimation: easy!
- For continuous or integer-valued features, use different distributions.

Naïve Bayes: Remarks

- Estimation by (smoothed) relative frequency estimation: easy!
- For continuous or integer-valued features, use different distributions.
- The bag of words version equates to building a conditional language model for each label.

Naïve Bayes: Remarks

- Estimation by (smoothed) relative frequency estimation: easy!
- For continuous or integer-valued features, use different distributions.
- The bag of words version equates to building a conditional language model for each label.
- Some authors assume a binary version, with F_v indicating whether $v \in \mathcal{V}$ occurs in $\boldsymbol{x}$.

Generative vs. Discriminative Classification

Naïve Bayes is the prototypical *generative* classifier.

- It describes a probabilistic process—"generative story"—for $\boldsymbol{X}$ (through $\boldsymbol{F}$) and L.
- But why model a distribution over $\boldsymbol{X}$ (or $\boldsymbol{F}$)? It's always observed!

Discriminative models instead:

- seek to optimize a performance measure, like accuracy, or a computationally convenient surrogate;
- do not worry about $p(\boldsymbol{X})$;
- tend to perform better when you have reasonable amounts of data.

Discriminative Text Classifiers

- Multinomial logistic regression (also known as “max ent” and “log-linear model”)
- Support vector machines
- Neural networks
- Decision trees

I’ll briefly touch on three ways to train a classifier with a linear decision rule.

Linear Models for Classification

"Linear" decision rule:

$$\hat{\ell} = \operatorname*{argmax}_{\ell \in \mathcal{L}} \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell)$$

where $\boldsymbol{\phi} : \mathcal{V}^\dagger \times \mathcal{L} \rightarrow \mathbb{R}^d$.

Parameters: $\mathbf{w} \in \mathbb{R}^d$

Linear Models for Classification

"Linear" decision rule:

$$\hat{\ell} = \operatorname*{argmax}_{\ell \in \mathcal{L}} \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell)$$

where $\boldsymbol{\phi} : \mathcal{V}^\dagger \times \mathcal{L} \rightarrow \mathbb{R}^d$.

Parameters: $\mathbf{w} \in \mathbb{R}^d$

Some notational variants define:

- $\mathbf{w}_\ell$ for each $\ell \in \mathcal{L}$
- $\boldsymbol{\phi} : \mathcal{V}^\dagger \rightarrow \mathbb{R}^d$ (similar to what we had for naïve Bayes)

Multinomial Logistic Regression as "Log Loss"

When we discussed log-linear language models, we transformed the score into a probability distribution. Here, that would be:

$$p(L = \ell \mid \boldsymbol{x}) = \frac{\exp \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell)}{\sum_{\ell' \in \mathcal{L}} \exp \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell')}$$

Multinomial Logistic Regression as "Log Loss"

When we discussed log-linear language models, we transformed the score into a probability distribution. Here, that would be:

$$p(L = \ell \mid \boldsymbol{x}) = \frac{\exp \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell)}{\sum_{\ell' \in \mathcal{L}} \exp \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell')}$$

MLE can be rewritten as a minimization problem:

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \sum_{i=1}^{n} \underbrace{\log \left(\sum_{\ell' \in \mathcal{L}} \left(\exp \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_i, \ell') \right) \right)}_{\text{fear}} - \underbrace{\mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_i, \ell_i)}_{\text{hope}}$$

Multinomial Logistic Regression as "Log Loss"

When we discussed log-linear language models, we transformed the score into a probability distribution. Here, that would be:

$$p(L = \ell \mid \boldsymbol{x}) = \frac{\exp \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell)}{\sum_{\ell' \in \mathcal{L}} \exp \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell')}$$

MLE can be rewritten as a minimization problem:

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \sum_{i=1}^{n} \underbrace{\log \left(\sum_{\ell' \in \mathcal{L}} \left(\exp \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_i, \ell') \right) \right)}_{\text{fear}} - \underbrace{\mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_i, \ell_i)}_{\text{hope}}$$

Recall from log-linear language models:

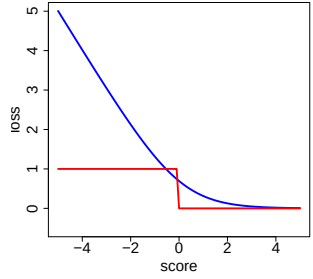
- Be wise and regularize!
- Solve with batch or stochastic gradient methods.
- w_j has an interpretation.

Log Loss for $(\boldsymbol{x}, \ell)$

Another view is to minimize the negated log-likelihood, which is known as "log loss":

$$\left(\log \sum_{\ell' \in \mathcal{L}} \exp \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell')\right) - \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell)$$

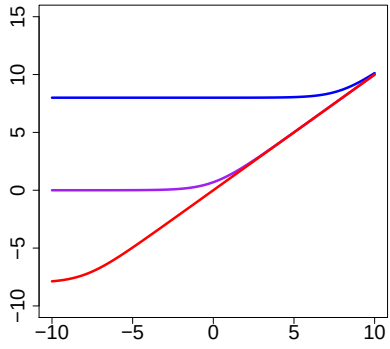
In the binary case, where "score" is the score of the correct label:



In blue is the log loss; in red is the "zero-one" loss (error).

"Log Sum Exp"

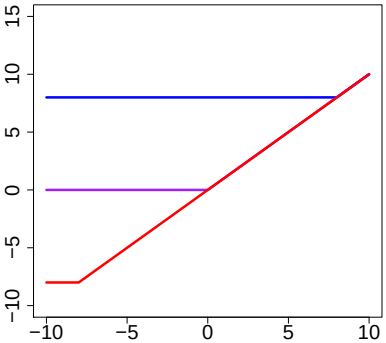
Consider the "$\log \sum \exp$" part of the objective function, with two labels, one whose score is fixed.



$\log(e^x + e^8)$, $\log(e^x + e^0)$, $\log(e^x + e^{-8})$

Hard Maximum

Why not use a hard max instead?

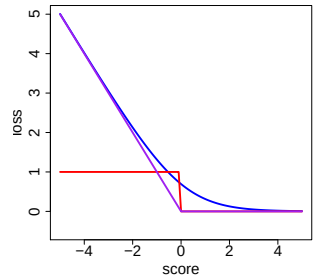


$\max(x, 8)$, $\max(x, 0)$, $\max(x, -8)$

Hinge Loss for $(\boldsymbol{x}, \ell)$

$$\left(\max_{\ell' \in \mathcal{L}} \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell')\right) - \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell)$$

In the binary case:



In purple is the hinge loss, in blue is the log loss; in red is the "zero-one" loss (error).

Minimizing Hinge Loss: Perceptron

$$\overbrace{\left(\max_{\ell' \in \mathcal{L}} \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell')\right)}^{\text{fear}} - \overbrace{\mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell)}^{\text{hope}}$$

Minimizing Hinge Loss: Perceptron

$$\overbrace{\left(\max_{\ell' \in \mathcal{L}} \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell')\right)}^{\text{fear}} - \overbrace{\mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell)}^{\text{hope}}$$

When two labels are *tied*, the function is not differentiable.

Minimizing Hinge Loss: Perceptron

$$\overbrace{\left(\max_{\ell' \in \mathcal{L}} \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell')\right)}^{\text{fear}} - \overbrace{\mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell)}^{\text{hope}}$$

When two labels are *tied*, the function is not differentiable.

But it's still *sub-differentiable*. Solution: (stochastic) subgradient descent!

Minimizing Hinge Loss: Perceptron

$$\overbrace{\left(\max_{\ell' \in \mathcal{L}} \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell')\right)}^{\text{fear}} - \overbrace{\mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell)}^{\text{hope}}$$

When two labels are *tied*, the function is not differentiable.

But it's still *sub-differentiable*. Solution: (stochastic) subgradient descent!

Perceptron algorithm:

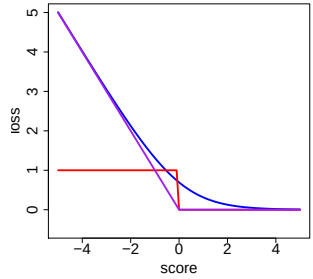
- For $t \in \{1, \ldots, T\}$:
  - Pick i_t uniformly at random from $\{1, \ldots, n\}$.
  - $\hat{\ell}_t \leftarrow \operatorname{argmax}_{\ell \in \mathcal{L}} \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_{i_t}, \ell)$
  - $\mathbf{w} \leftarrow \mathbf{w} - \alpha \left(\boldsymbol{\phi}(\boldsymbol{x}_{i_t}, \hat{\ell}) - \boldsymbol{\phi}(\boldsymbol{x}_{i_t}, \ell_{i_t})\right)$

Log Loss and Hinge Loss for $(\boldsymbol{x}, \ell)$

$$\text{log loss:} \quad \left(\log \sum_{\ell' \in \mathcal{L}} \exp \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell') \right) - \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell)$$

$$\text{hinge loss:} \quad \left(\max_{\ell' \in \mathcal{L}} \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell') \right) - \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell)$$

In the binary case, where "score" is the linear score of the correct label:



Minimizing Hinge Loss: Perceptron

$$\min_{\mathbf{w}} \sum_{i=1}^{n} \left(\max_{\ell' \in \mathcal{L}} \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_i, \ell') \right) - \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_i, \ell_i)$$

Stochastic subgradient descent on the above is called the **perceptron** algorithm.

- For $t \in \{1, \ldots, T\}$:
  - Pick i_t uniformly at random from $\{1, \ldots, n\}$.
  - $\hat{\ell}_{i_t} \leftarrow \operatorname{argmax}_{\ell \in \mathcal{L}} \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_{i_t}, \ell)$
  - $\mathbf{w} \leftarrow \mathbf{w} - \alpha \left(\boldsymbol{\phi}(\boldsymbol{x}_{i_t}, \hat{\ell}_{i_t}) - \boldsymbol{\phi}(\boldsymbol{x}_{i_t}, \ell_{i_t}) \right)$

Error Costs

Suppose that not all mistakes are equally bad.

E.g., false positives vs. false negatives in spam detection.

Error Costs

Suppose that not all mistakes are equally bad.

E.g., false positives vs. false negatives in spam detection.

Let $\mathrm{cost}(\ell, \ell')$ quantify the "badness" of substituting ℓ' for correct label ℓ.

Error Costs

Suppose that not all mistakes are equally bad.

E.g., false positives vs. false negatives in spam detection.

Let $\mathrm{cost}(\ell, \ell')$ quantify the "badness" of substituting ℓ' for correct label ℓ.

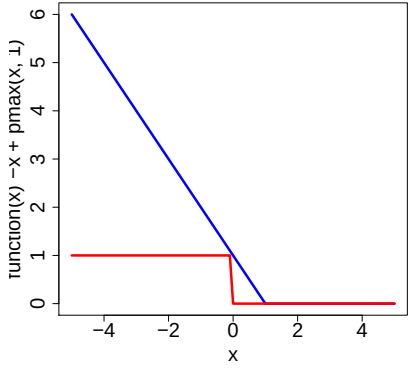
Intuition: estimate the scoring function so that

$$\mathrm{score}(\ell_i) - \mathrm{score}(\hat{\ell}) \propto \mathrm{cost}(\ell_i, \hat{\ell})$$

General Hinge Loss for $(\boldsymbol{x}, \ell)$

$$\left(\max_{\ell' \in \mathcal{L}} \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell') + \text{cost}(\ell, \ell') \right) - \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell)$$

In the binary case, with $\text{cost}(-1, 1) = 1$:



General Remarks

- Text classification: many problems, all solved with supervised learners.
  - Lexicon features can provide problem-specific guidance.

General Remarks

- Text classification: many problems, all solved with supervised learners.
  - Lexicon features can provide problem-specific guidance.
- Naïve Bayes, log-linear, and perceptron are all *linear* methods that tend to work reasonably well, with good features and smoothing/regularization.
  - You should have a basic understanding of the tradeoffs in choosing among them.

General Remarks

- Text classification: many problems, all solved with supervised learners.
  - Lexicon features can provide problem-specific guidance.
- Naïve Bayes, log-linear, and perceptron are all *linear* methods that tend to work reasonably well, with good features and smoothing/regularization.
  - You should have a basic understanding of the tradeoffs in choosing among them.
- Random forests are widely used in industry when performance matters more than interpretability.

General Remarks

- Text classification: many problems, all solved with supervised learners.
  - Lexicon features can provide problem-specific guidance.
- Naïve Bayes, log-linear, and perceptron are all *linear* methods that tend to work reasonably well, with good features and smoothing/regularization.
  - You should have a basic understanding of the tradeoffs in choosing among them.
- Random forests are widely used in industry when performance matters more than interpretability.
- Lots of papers about neural networks, but with hyperparameter tuning applied fairly to linear models, the advantage is not clear (Yogatama et al., 2015). Increasingly in large-data settings, neural nets do win.

References I

Dani Yogatama, Lingpeng Kong, and Noah A. Smith. Bayesian optimization of text representations. In *Proc. of EMNLP*, 2015. URL `http://www.aclweb.org/anthology/D/D15/D15-1251.pdf`.