Natural Language Processing (CSE 517): Text Classification

Noah Smith

University of Washington
nasmith@cs.washington.edu

April 20, 2018

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 $\text{classify} : \mathcal{V}^\dagger \rightarrow \mathcal{L}$
3. Apply 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.

Evaluation

Accuracy:

$$
\begin{aligned}
\mathrm{A}(\text{classify}) &= p(\text{classify}(\boldsymbol{X}) = L) \\
&= \sum_{\boldsymbol{x} \in \mathcal{V}^\dagger, \ell \in \mathcal{L}} p(\boldsymbol{X} = \boldsymbol{x}, L = \ell) \cdot \begin{cases} 1 & \text{if } \text{classify}(\boldsymbol{x}) = \ell \\ 0 & \text{otherwise} \end{cases} \\
&= \sum_{\boldsymbol{x} \in \mathcal{V}^\dagger, \ell \in \mathcal{L}} p(\boldsymbol{X} = \boldsymbol{x}, L = \ell) \cdot \mathbf{1}\left\{\text{classify}(\boldsymbol{x}) = \ell\right\}
\end{aligned}
$$

where p is the *true* distribution over data. Error is $1 - \mathrm{A}$.

This is *estimated* using a test dataset $\langle \bar{\boldsymbol{x}}_1, \bar{\ell}_1 \rangle, \ldots \langle \bar{\boldsymbol{x}}_m, \bar{\ell}_m \rangle$:

$$
\hat{\mathrm{A}}(\text{classify}) = \frac{1}{m} \sum_{i=1}^{m} \mathbf{1}\left\{\text{classify}(\bar{\boldsymbol{x}}_i) = \bar{\ell}_i\right\}
$$

Issues with Test-Set Accuracy

Issues with Test-Set Accuracy

- Class imbalance: if $p(L = \text{not spam}) = 0.99$, then you can get $\hat{\mathrm{A}} \approx 0.99$ by always guessing "not spam."

Issues with Test-Set Accuracy

- Class imbalance: if $p(L = \text{not spam}) = 0.99$, then you can get $\hat{\text{A}} \approx 0.99$ by always guessing "not spam."
- Relative importance of classes or cost of error types.

Issues with Test-Set Accuracy

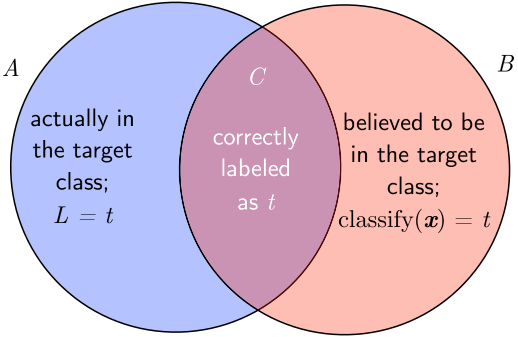
- Class imbalance: if $p(L = \text{not spam}) = 0.99$, then you can get $\hat{\mathrm{A}} \approx 0.99$ by always guessing "not spam."
- Relative importance of classes or cost of error types.
- Variance due to the test data.

Evaluation in the Two-Class Case

Suppose we have two classes, and one of them, $t \in \mathcal{L}$ is a "target."

- E.g., given a query, find relevant documents.

Precision and **recall** encode the goals of returning a "pure" set of targeted instances and capturing *all* of them.



$$\hat{\mathrm{P}}(\text{classify}) = \frac{|C|}{|B|} = \frac{|A \cap B|}{|B|}$$

$$\hat{\mathrm{R}}(\text{classify}) = \frac{|C|}{|A|} = \frac{|A \cap B|}{|A|}$$

$$\hat{F}_1(\text{classify}) = 2 \cdot \frac{\hat{\mathrm{P}} \cdot \hat{\mathrm{R}}}{\hat{\mathrm{P}} + \hat{\mathrm{R}}}$$

Another View: Contingency Table

	$L = t$	$L \neq t$	
$\text{classify}(\boldsymbol{X}) = t$	C (true positives)	$B \setminus C$ (false positives)	B
$\text{classify}(\boldsymbol{X}) \neq t$	$A \setminus C$ (false negatives)	(true negatives)	
	A		

Evaluation with > 2 Classes

Macroaveraged precision and recall: let each class be the target and report the average $\hat{\mathrm{P}}$ and $\hat{\mathrm{R}}$ across all classes.

Microaveraged precision and recall: pool all one-vs.-rest decisions into a single contingency table, calculate $\hat{\mathrm{P}}$ and $\hat{\mathrm{R}}$ from that.

Cross-Validation

Remember that $\hat{\mathrm{A}}$, $\hat{\mathrm{P}}$, $\hat{\mathrm{R}}$, and $\hat{F}_1$ are all *estimates* of the classifier's quality under the true data distribution.

- Estimates are noisy!

K-fold cross-validation:

- Partition the training set into K non-overlapping "folds" $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^K$.
- For $i \in \{1, \ldots, K\}$:
  - Train on $\boldsymbol{x}_{1:n} \setminus \boldsymbol{x}^i$, using $\boldsymbol{x}^i$ as development data.
  - Estimate quality on the ith development set: $\hat{\mathrm{A}}^i$
- Report the average:

$$\hat{\mathrm{A}} = \frac{1}{K} \sum_{i=1}^{K} \hat{\mathrm{A}}^i$$

and perhaps also the standard error.

Statistical Significance

Suppose we have two classifiers, $\mathrm{classify}_1$ and $\mathrm{classify}_2$.

Statistical Significance

Suppose we have two classifiers, $\mathrm{classify}_1$ and $\mathrm{classify}_2$.

Is $\mathrm{classify}_1$ better? The "null hypothesis," denoted H_0, is that it isn't. But if $\hat{\mathrm{A}}_1 \gg \hat{\mathrm{A}}_2$, we are tempted to believe otherwise.

Statistical Significance

Suppose we have two classifiers, $\mathrm{classify}_1$ and $\mathrm{classify}_2$.

Is $\mathrm{classify}_1$ better? The "null hypothesis," denoted H_0, is that it isn't. But if $\hat{\mathrm{A}}_1 \gg \hat{\mathrm{A}}_2$, we are tempted to believe otherwise.

How much larger must $\hat{\mathrm{A}}_1$ be than $\hat{\mathrm{A}}_2$ to *reject* H_0?

Statistical Significance

Suppose we have two classifiers, $\mathrm{classify}_1$ and $\mathrm{classify}_2$.

Is $\mathrm{classify}_1$ better? The "null hypothesis," denoted H_0, is that it isn't. But if $\hat{\mathrm{A}}_1 \gg \hat{\mathrm{A}}_2$, we are tempted to believe otherwise.

How much larger must $\hat{\mathrm{A}}_1$ be than $\hat{\mathrm{A}}_2$ to *reject* H_0?

Frequentist view: how (im)probable is the observed difference, given $H_0 = \text{true}$?

Statistical Significance

Suppose we have two classifiers, classify_1 and classify_2.

Is classify_1 better? The "null hypothesis," denoted H_0, is that it isn't. But if $\hat{\text{A}}_1 \gg \hat{\text{A}}_2$, we are tempted to believe otherwise.

How much larger must $\hat{\text{A}}_1$ be than $\hat{\text{A}}_2$ to *reject* H_0?

Frequentist view: how (im)probable is the observed difference, given $H_0 =$ true?

Caution: statistical significance is neither necessary nor sufficient for research significance or practical usefulness!

A Hypothesis Test for Text Classifiers

McNemar (1947)

1. The null hypothesis: $\mathrm{A}_1 = \mathrm{A}_2$
2. Pick significance level α, an "acceptably" high probability of incorrectly rejecting H_0.
3. Calculate the test statistic, k (explained in the next slide).
4. Calculate the probability of a *more extreme* value of k, assuming H_0 is true; this is the p-value.
5. Reject the null hypothesis if the p-value is less than α.

The p-value is $p(\text{this observation} \mid H_0 \text{ is true})$, not the other way around!

McNemar's Test: Details

Assumptions: independent (test) samples and binary measurements. Count test set error patterns:

	classify_1 is incorrect	classify_1 is correct	
classify_2 is incorrect	c_{00}	c_{10}	
classify_2 is correct	c_{01}	c_{11}	$m \cdot \hat{\mathrm{A}}_2$
		$m \cdot \hat{\mathrm{A}}_1$	

If $\mathrm{A}_1 = \mathrm{A}_2$, then c_{01} and c_{10} are each distributed according to $\mathrm{Binomial}(c_{01} + c_{10}, \frac{1}{2})$.

$$\text{test statistic } k = \min\{c_{01}, c_{10}\}$$

$$p\text{-value} = \frac{1}{2^{c_{01}+c_{10}-1}} \sum_{j=0}^{k} \binom{c_{01} + c_{10}}{j}$$

Other Tests

Different tests make different assumptions.

Sometimes we calculate an interval that would be "unsurprising" under H_0 and test whether a test statistic falls in that interval (e.g., t-test and Wald test).

In many cases, there is no closed form for estimating p-values, so we use random approximations (e.g., permutation test and paired bootstrap test).

If you do lots of tests, you need to correct for that!

Read lots more in Smith (2011), appendix B.

Features in Text Classification

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

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

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

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

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

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

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

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

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

- Disjunctions of terms
  - Clusters
  - Task-specific lexicons

Probabilistic Classification

Classification rule:

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

Naïve Bayes Classifier

$$
\begin{aligned}
p(L = \ell, F_j = f_1, \ldots, F_d = f_d) &= p(\ell) \prod_{j=1}^{d} p(F_j = f_j \mid \ell) \\
&= \pi_\ell \prod_{j=1}^{d} \theta_{f_j|j,\ell}
\end{aligned}
$$

Parameters:

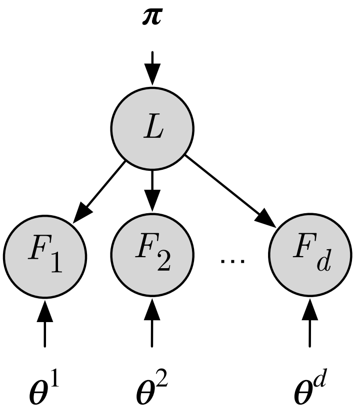
- $\boldsymbol{\pi} \in \triangle^{|\mathcal{L}|}$, the "class prior"
- For each feature function j and label ℓ, a distribution over values $\boldsymbol{\theta}_{*|j,\ell} \in \triangle^{|\mathcal{F}_j|}$

The "bag of words" version of naïve Bayes:

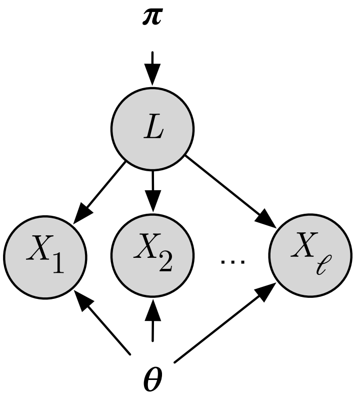
$$F_j = X_j$$

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

Probabilistic Graphical Model for Naïve Bayes



general form



bag of words

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.
- The Collins reading assumes 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 $\boldsymbol{X}$ and L.
- But why model $\boldsymbol{X}$? 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:

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

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:

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

Recall from log-linear language models:

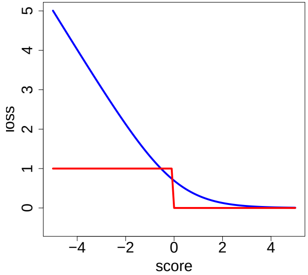
- Be wise and regularize!
- Solve with batch or stochastic gradient methods.
- w_j has an interpretation.

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

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

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

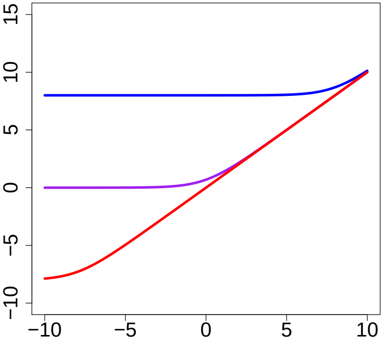
In the binary case, where "score" is the score of the correct label:



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

"Log Sum Exp"

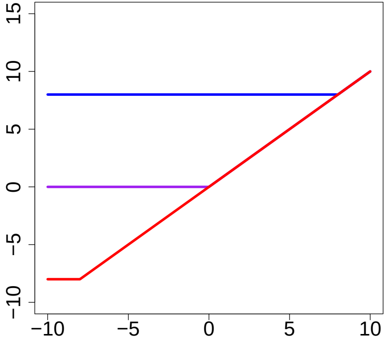
Consider the "$\log \sum \exp$" part of the objective function, with two labels, one whose score is fixed.



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

Hard Maximum

Why not use a hard max instead?

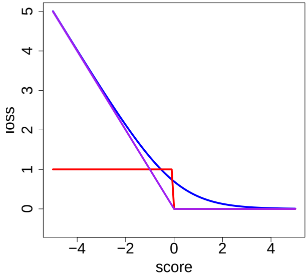


$\max(x, 8)$, $\max(x, 0)$, $\max(x, -8)$

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 $\{1, \ldots, n\}$.
  - $\hat{\ell}_t \leftarrow \text{argmax}_{\ell \in \mathcal{L}} \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_{i_t}, \ell)$
  - $\mathbf{w} \leftarrow \mathbf{w} - \alpha \left(\boldsymbol{\phi}(\boldsymbol{x}_{i_t}, \hat{\ell}) - \boldsymbol{\phi}(\boldsymbol{x}_{i_t}, \ell_{i_t})\right)$

Natural Language Processing (CSE 517): Text Classification

Noah Smith

University of Washington
nasmith@cs.washington.edu

April 25, 2018

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 $\text{classify} : \mathcal{V}^\dagger \to \mathcal{L}$
3. Apply classify to as much data as you want!

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

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:

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

Recall from log-linear language models:

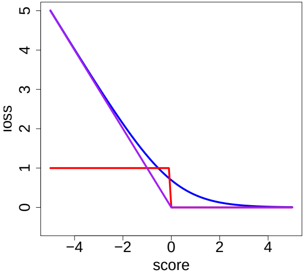
- Be wise and regularize!
- Solve with batch or stochastic gradient methods.
- w_j has an interpretation.

Log Loss 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 $\text{cost}(\ell, \ell')$ 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.

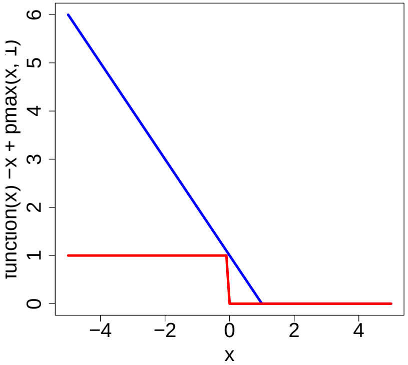
Intuition: estimate the scoring function so that

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

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

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

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



Support Vector Machines

A different motivation for the generalized hinge:

$$\hat{\mathbf{w}} = \sum_{i=1}^{n} \sum_{\ell \in \mathcal{L}} \alpha_{i,\ell} \cdot \boldsymbol{\phi}(\boldsymbol{x}_i, \ell)$$

where only a small number of $\alpha_{i,\ell}$ are nonzero.

Support Vector Machines

A different motivation for the generalized hinge:

$$\hat{\mathbf{w}} = \sum_{i=1}^{n} \sum_{\ell \in \mathcal{L}} \alpha_{i,\ell} \cdot \boldsymbol{\phi}(\boldsymbol{x}_i, \ell)$$

where only a small number of $\alpha_{i,\ell}$ are nonzero.

Those $\boldsymbol{\phi}(\boldsymbol{x}_i, \ell)$ are called "support vectors" because they "support" the decision boundary.

$$\hat{\mathbf{w}} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell') = \sum_{(i,\ell) \in \mathcal{S}} \alpha_{i,\ell} \cdot \boldsymbol{\phi}(\boldsymbol{x}_i, \ell) \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell')$$

See Crammer and Singer (2001) for the multiclass version.

Support Vector Machines

A different motivation for the generalized hinge:

$$\hat{\mathbf{w}} = \sum_{i=1}^{n} \sum_{\ell \in \mathcal{L}} \alpha_{i,\ell} \cdot \boldsymbol{\phi}(\boldsymbol{x}_i, \ell)$$

where only a small number of $\alpha_{i,\ell}$ are nonzero.

Those $\boldsymbol{\phi}(\boldsymbol{x}_i, \ell)$ are called "support vectors" because they "support" the decision boundary.

$$\hat{\mathbf{w}} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell') = \sum_{(i,\ell) \in \mathcal{S}} \alpha_{i,\ell} \cdot \boldsymbol{\phi}(\boldsymbol{x}_i, \ell) \cdot \boldsymbol{\phi}(\boldsymbol{x}, \ell')$$

See Crammer and Singer (2001) for the multiclass version.

Really good tool: SVM*light*, `http://svmlight.joachims.org`

Support Vector Machines: Remarks

- Regularization is critical; squared ℓ_2 is most common, and often used in (yet another) motivation around the idea of “maximizing margin” around the hyperplane separator.

Support Vector Machines: Remarks

- Regularization is critical; squared ℓ_2 is most common, and often used in (yet another) motivation around the idea of "maximizing margin" around the hyperplane separator.
- Often, instead of linear models that explicitly calculate $\mathbf{w} \cdot \boldsymbol{\phi}$, these methods are "kernelized" and rearrange all calculations to involve inner-products between $\boldsymbol{\phi}$ vectors.
  - Example:

$$\begin{aligned}
K_{\text{linear}}(\mathbf{v}, \mathbf{w}) &= \mathbf{v} \cdot \mathbf{w} \\
K_{\text{polynomial}}(\mathbf{v}, \mathbf{w}) &= (\mathbf{v} \cdot \mathbf{w} + 1)^p \\
K_{\text{Gaussian}}(\mathbf{v}, \mathbf{w}) &= \exp -\frac{\|\mathbf{v} - \mathbf{w}\|_2^2}{2\sigma^2}
\end{aligned}$$

  - Linear kernels are most common in NLP.

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 SVM 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 SVM 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 SVM 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).

References I

Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2(5):265–292, 2001.

Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

Noah A. Smith. *Linguistic Structure Prediction*. Synthesis Lectures on Human Language Technologies. Morgan and Claypool, 2011. URL http://www.morganclaypool.com/doi/pdf/10.2200/S00361ED1V01Y201105HLT013.pdf.

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.