Natural Language Processing
Linear Models

Felipe Bravo-Marquez

July 17, 2019

Supervised Learning

- The essence of supervised machine learning is the creation of mechanisms that can look at examples and produce generalizations. [**?**]
- We design an algorithm whose input is a set of labeled examples, and its output is a function (or a program) that receives an instance and produces the desired label.
- Example: if the task is to distinguish from spam and not-spam email, the labeled examples are emails labeled as spam and emails labeled as not-spam.
- It is expected that the resulting function will produce correct label predictions also for instances it has not seen during training.
- This approach differs from designing an algorithm to perform the task (e.g., manually designed rule-based systems).

Parameterized Functions

- Searching over the set of all possible functions is a very hard (and rather ill-defined) problem. [**?**]
- We often restrict ourselves to search over specific families of functions.
- Example: the space of all linear functions with d_{in} inputs and d_{out} outputs,
- Such families of functions are called **hypothesis classes**.
- By restricting ourselves to a specific hypothesis class, we are injecting the learner with **inductive bias**.
- Inductive bias: a set of assumptions about the form of the desired solution.
- Some hypothesis classes facilitate efficient procedures for searching for the solution. [**?**]

Linear Models

- One common hypothesis class is that of high-dimensional linear function:

$$f(x) = \vec{x} \cdot W + \vec{b}$$
$$\vec{x} \in \mathcal{R}^{d_{in}} \quad W \in \mathcal{R}^{d_{in} \times d_{out}} \quad \vec{b} \in \mathcal{R}^{d_{out}} \tag{1}$$

- The vector $\vec{x}$ is the input to the function.
- The matrix W and the vector $\vec{b}$ are the parameters.
- The goal of the learner is to set the values of the parameters W and $\vec{b}$ such that the function behaves as intended on a collection of input values $\vec{x}_{1:k} = \vec{x}_1, \ldots, \vec{x}_k$ and the corresponding desired outputs $\vec{y}_{1:k} = \vec{y}_1, \ldots, \vec{y}_k$
- The task of searching over the space of functions is thus reduced to one of searching over the space of parameters. [**?**]

Example: Language Detection

- Consider the task of distinguishing documents written in English from documents written in German.
- This is a binary classifcation problem

$$f(x) = \vec{x} \cdot \vec{w} + b \tag{2}$$

$d_{out} = 1$, $\vec{w}$ is a vector, and b is a scalar.

- The range of the linear function in is $[-\infty, \infty]$.
- In order to use it for binary classification, it is common to pass the output of $f(x)$ through the *sign* function, mapping negative values to -1 (the negative class) and non-negative values to +1 (the positive class).

Example: Language Detection

- Letter frequencies make for quite good predictors (features) for this task.
- Even more informative are counts of letter bigrams, i.e., pairs of consecutive letters.
- One may think that words will also be good predictors i.e., using a bag of word representation of documents.
- Letters, or letter-bigrams are far more robust.
- We are likely to encounter a new document without any of the words we observed in the training set.
- While a document without any of the distinctive letter-bigrams is significantly less likely. [**?**]

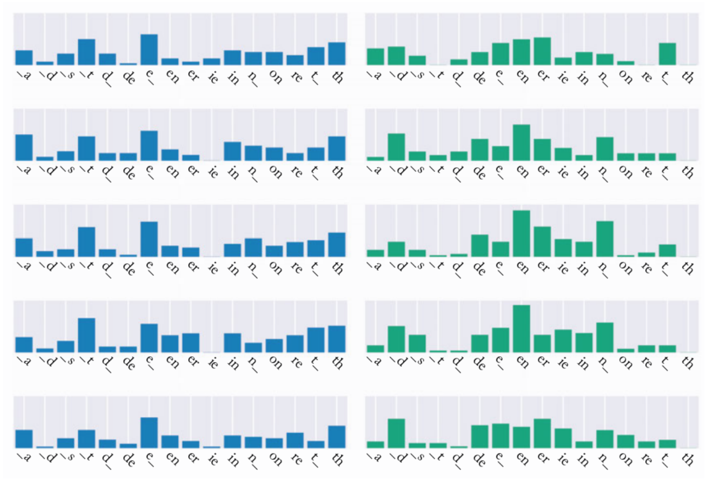
Example: Language Detection

- We assume we have an alphabet of 28 letters (a–z, space, and a special symbol for all other characters including digits, punctuations, etc.)
- Documents are represented as 28×28 dimensional vectors $\vec{x} \in \mathcal{R}^{784}$.
- Each entry $\vec{x}_{[i]}$ represents a count of a particular letter combination in the document, normalized by the document's length.
- For example, denoting by $\vec{x}_{ab}$ the entry of $\vec{x}$ corresponding to the letter bigram *ab*:

$$x_{ab} = \frac{\#ab}{|D|} \tag{3}$$

where $\#ab$ is the number of times the bigram *ab* appears in the document, and $|D|$ is the total number of bigrams in the document (the document's length).

Example: Language Detection

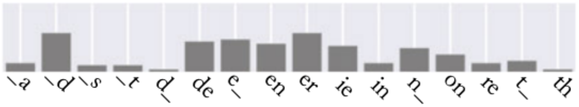


Character-bigram histograms for documents in English (left, blue) and German(right,green). Underscores denote spaces. Only the top frequent character-bigrams are showed.

[0]Source:[**?**]

Example: Language Detection

- Previous figure showed clear patterns in the data, and, given a new item, such as:



- We could probably tell that it is more similar to the German group than to the English one (observe the frequency of “th” and “ie”).
- We can’t use a single definite rule such as “if it has th its English” or “if it has ie its German”.
- While German texts have considerably less “th” than English, the “th” may and does occur in German texts, and similarly the “ie” combination does occur in English.

Example: Language Detection

- The decision requires weighting different factors relative to each other.
- We can formalize the problem in a machine-learning setup using a linear model:

$$\begin{aligned} \hat{y} &= sign(f(\vec{x})) = sign(\vec{x} \cdot \vec{w} + b) \\ &= sign(\vec{x}_{aa} \times \vec{w}_{aa} + \vec{x}_{ab} \times \vec{w}_{ab} + \vec{x}_{ac} \times \vec{w}_{ac} \cdots + b) \end{aligned} \tag{4}$$

- A document will be considered English if $f(\vec{x}) \geq 0$ and as German otherwise.

Intuition

1. Learning should assign large positive values to $\vec{w}$ entries associated with letter pairs that are much more common in English than in German (i.e., “th”).
2. It should also assign negative values to letter pairs that are much more common in German than in English (“ie”, “en”).
3. Finally, it should assign values around zero to letter pairs that are either common or rare in both languages.

Log-linear Binary classifcation

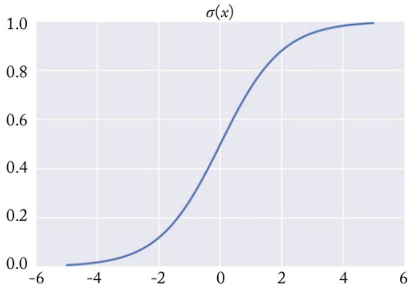
- The output $f(\vec{x})$ is in the range $[-\infty, \infty]$, and we map it to one of two classes $\{-1, +1\}$ using the *sign* function.
- This is a good fit if all we care about is the assigned class.
- We may be interested also in the confidence of the decision, or the probability that the classifier assigns to the class.
- An alternative that facilitates this is to map instead to the range $[0, 1]$, by pushing the output through a squashing function such as the sigmoid $\sigma(x)$:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

resulting in:

$$\hat{y} = \sigma(f(\vec{x})) = \frac{1}{1 + e^{-\vec{x} \cdot \vec{w} + b}} \tag{6}$$

The Sigmoid function

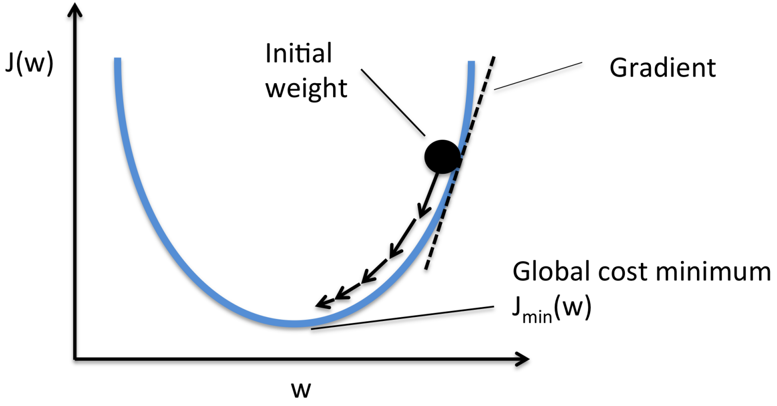


- The output $f(\vec{x})$ is in the range $[-\infty, \infty]$, and we map it to one of two classes $\{-1, +1\}$ using the *sign* function.
- This is a good fit if all we care about is the assigned class.
- We may be interested also in the confidence of the decision, or the probability that the classifier assigns to the class.

Training

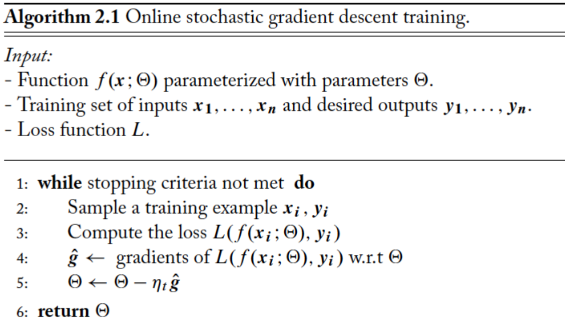
- When training a parameterized function (e.g., a linear model, a neural network) one defines a loss function $L(\hat{y}, y)$, stating the loss of predicting $\hat{y}$ when the true output is y.
- The training objective is then to minimize the loss across the different training examples.
- Functions are trained using gradient-based methods.
- They work by repeatedly computing an estimate of the loss L over the training set.
- They compute gradients of the parameters with respect to the loss estimate, and moving the parameters in the opposite directions of the gradient.
- Different optimization methods differ in how the error estimate is computed, and how moving in the opposite direction of the gradient is defined.

Gradient Descent



[0]Source: `https://sebastianraschka.com/images/faq/closed-form-vs-gd/ball.png`

Online Stochastic Gradient Descent



Algorithm 2.1 Online stochastic gradient descent training.

Input:
- Function $f(\boldsymbol{x};\Theta)$ parameterized with parameters Θ.
- Training set of inputs $\boldsymbol{x_1},\ldots,\boldsymbol{x_n}$ and desired outputs $\boldsymbol{y_1},\ldots,\boldsymbol{y_n}$.
- Loss function L.

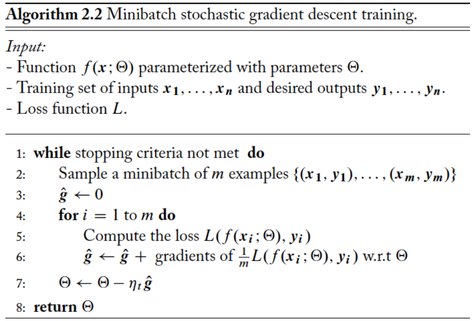
```
1: while stopping criteria not met do
2:     Sample a training example $\boldsymbol{x_i}, \boldsymbol{y_i}$
3:     Compute the loss $L(f(\boldsymbol{x_i};\Theta), \boldsymbol{y_i})$
4:     $\hat{\boldsymbol{g}} \leftarrow$ gradients of $L(f(\boldsymbol{x_i};\Theta), \boldsymbol{y_i})$ w.r.t $\Theta$
5:     $\Theta \leftarrow \Theta - \eta_t \hat{\boldsymbol{g}}$
6: return $\Theta$
```

- The learning rate can either be fixed throughout the training process, or decay as a function of the time step *t*.
- The error calculated in line 3 is based on a single training example, and is thus just a rough estimate of the corpus-wide loss *L* that we are aiming to minimize.
- The noise in the loss computation may result in inaccurate gradients (single examples may provide noisy information).

[0]Source:[**?**]

Mini-batch Stochastic Gradient Descent

- A common way of reducing this noise is to estimate the error and the gradients based on a sample of *m* examples.
- This gives rise to the minibatch SGD algorithm



Algorithm 2.2 Minibatch stochastic gradient descent training.

Input:
- Function $f(x; \Theta)$ parameterized with parameters Θ.
- Training set of inputs $x_1, \ldots, x_n$ and desired outputs $y_1, \ldots, y_n$.
- Loss function L.

```
1: while stopping criteria not met do
2:     Sample a minibatch of $m$ examples $\{(x_1, y_1), \ldots, (x_m, y_m)\}$
3:     $\hat{g} \leftarrow 0$
4:     for $i = 1$ to $m$ do
5:         Compute the loss $L(f(x_i; \Theta), y_i)$
6:         $\hat{g} \leftarrow \hat{g} +$ gradients of $\frac{1}{m} L(f(x_i; \Theta), y_i)$ w.r.t $\Theta$
7:     $\Theta \leftarrow \Theta - \eta_t \hat{g}$
8: return $\Theta$
```

- Higher values of *m* provide better estimates of the corpus-wide gradients, while smaller values allow more updates and in turn faster convergence.
- For modest sizes of *m* , some computing architectures (i.e., GPUs) allow an efficient parallel implementation of the computation in lines 3-6.

[0]Source:[**?**]

Some Loss Functions

- Hinge (or SVM loss): for binary classification problems, the classifier's output is a single scalar $\tilde{y}$ and the intended output y is in $\{+1, -1\}$. The classification rule is $\hat{y} = sign(\tilde{y})$, and a classification is considered correct if $y \cdot \tilde{y} > 0$.

$$L_{\text{hinge(binary)}}(\tilde{y}, y) = \max(0, 1 - y \cdot \tilde{y})$$

- Binary cross entropy (or logistic loss): is used in binary classification with conditional probability outputs. The classifier's output $\tilde{y}$ is transformed using the sigmoid function to the range $[0, 1]$, and is interpreted as the conditional probability $P(y = 1|x)$.

$$L_{\text{logistic}}(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

Some Loss Functions

- Categorical cross-entropy loss: is used when a probabilistic interpretation of multi-class scores is desired. It measures the dissimilarity between the true label distribution y and the predicted label distribution $\tilde{y}$.

$$L_{\text{cross-entropy}}(\hat{y}, y) = -\sum_i y_{[i]} \log(\hat{y}_{[i]})$$

- The predicted label distribution of the categorical cross-entropy loss ($\hat{y}$) is obtained by applying the softmax function the last layer of the network $\tilde{y}$:

$$\hat{y}_{[i]} = \text{softmax}(\tilde{y})_{[i]} = \frac{e^{\tilde{y}_{[i]}}}{\sum_j e^{\tilde{y}_{[j]}}}$$

- The softmax function squashes the k-dimensional output to values in the range (0,1) with all entries adding up to 1. Hence, $\hat{y}_{[i]} = P(y = i|x)$ represent the class membership conditional distribution.

Train, Test, and Validation Sets

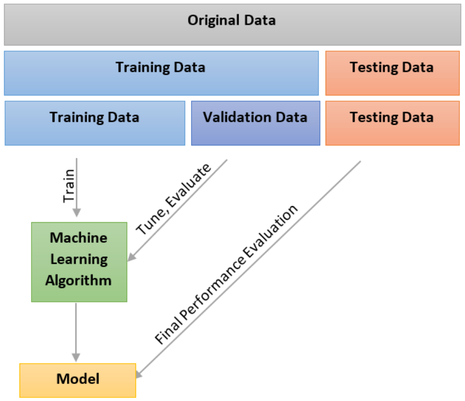
- Parameterized function are prone to overfit the data.
- Hence, performance on training data can be misleading.
- Held-out set: split training set into training and testing subsets (80% and 20% splits). Train on training and compute accuracy on testing.
- Problem: in practice you often train several models, compare their quality, and select the best one.
- Selecting the best model according to the held-out set's accuracy will result in an overly optimistic estimate of the model's quality.
- You don't know if the chosen settings of the final classifier are good in general, or are just good for the particular examples in the held-out sets.

Train, Test, and Validation Sets

- The accepted methodology is to use a three-way split of the data into train, validation (also called development), and test sets [1].
- This gives you two held-out sets: a validation set (also called development set), and a test set.
- All the experiments, tweaks, error analysis, and model selection should be performed based on the validation set.
- Then, a single run of the final model over the test set will give a good estimate of its expected quality on unseen examples.
- It is important to keep the test set as pristine as possible, running as few experiments as possible on it.
- Some even advocate that you should not even look at the examples in the test set, so as to not bias the way you design your model.

[1] An alternative approach is cross-validation, but it doesn't scale well for training deep neural networks.

Train, Test, and Validation Sets



[1] source: https://www.codeproject.com/KB/AI/1146582/validation.PNG

Questions?

Thanks for your Attention!

References I

Goldberg, Y. (2017).
Neural network methods for natural language processing.
Synthesis Lectures on Human Language Technologies, 10(1):1–309.