Classification: Linear Models

Ninghao Liu

University of Georgia

January 16, 2024

Some contents adopted from Stanford CS221, Percy Liang.

Road Map

$\boxed{\text{Linear predictors}}$

Learning objective

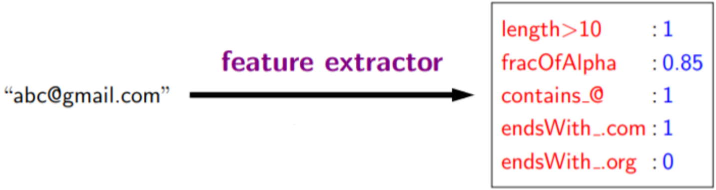
Optimization

Feature Extraction

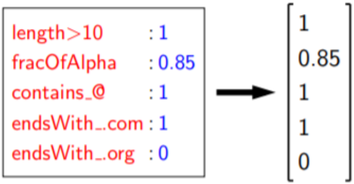
Example task: Predict y, whether a string x is an email address.

Question: What properties of x might be relevant for predicting y?

Feature extraction: Given input x, produce a set of (feature name, feature value) pairs.



Linear Classifier



For an input x, its **feature vector** is:

$$\phi(x) = [\phi(x)_1, \phi(x)_2, ..., \phi(x)_D].$$

In practice, we usually use $\boldsymbol{x}$ to denote the feature vector of input after feature extraction, i.e., $\boldsymbol{x} = \phi(x)$.

Linear Classifier

```
length>10     :-1.2
fracOfAlpha   :0.6
contains_@    :3
endsWith_.com :2.2
endsWith_.org :1.4
...
```

For each feature j, have real number w_j representing contribution of the feature to prediction.

Linear Classifier

length>10	:-1.2
fracOfAlpha	:0.6
contains_@	:3
endsWith_.com	:2.2
endsWith_.org	:1.4

length>10	:1
fracOfAlpha	:0.85
contains_@	:1
endsWith_.com	:1
endsWith_.org	:0

Figure 1: Left: Weight vector $\boldsymbol{w}$. Right: Feature vector $\phi(x)$.

Prediction score: The weighted sum of features.

$$\boldsymbol{w}^{\intercal} \cdot \boldsymbol{x} = \sum_{j}^{D} \boldsymbol{w}_j \boldsymbol{x}_j \tag{1}$$

Example:

$$-1.2(1) + 0.6(0.85) + 3(1) + 2.2(1) + 1.4(0) = 4.51$$

Linear Classifier

Weight vector $\boldsymbol{w} \in \mathbb{R}^D$.

Feature vector $\boldsymbol{x} \in \mathbb{R}^D$.

For binary classification, the linear classifier $f_{\boldsymbol{w}}$ is:

$$f_{\boldsymbol{w}}(x) = \text{sign}(\boldsymbol{w}^\intercal \cdot \boldsymbol{x}) = \begin{cases} +1, & \text{if } \boldsymbol{w}^\intercal \cdot \boldsymbol{x} > 0 \\ -1, & \text{if } \boldsymbol{w}^\intercal \cdot \boldsymbol{x} < 0 \\ ?, & \text{if } \boldsymbol{w}^\intercal \cdot \boldsymbol{x} = 0 \end{cases} \tag{2}$$

Linear Classifier: Geometric intuition

Example:

$\boldsymbol{w} = [2, -1]$

$\phi(x) = \{[2, 0], [0, 2], [2, 4]\}$

Linear Classifier: Geometric intuition

A binary classifier $f_{\boldsymbol{w}}$ defines a **hyperplane** with a normal vector $\boldsymbol{w}$.

($\mathbb{R}^2 \rightarrow$ the hyperplane is a line; $\mathbb{R}^3 \rightarrow$ the hyperplane is a plane)

Linear Classifier: Geometric intuition

Please note that, the complete definition of a linear classifier is:

$$f_{\boldsymbol{w}}(x) = sign(\boldsymbol{w}^{\intercal} \cdot \boldsymbol{x} + b), \tag{3}$$

where b is the offset parameter.

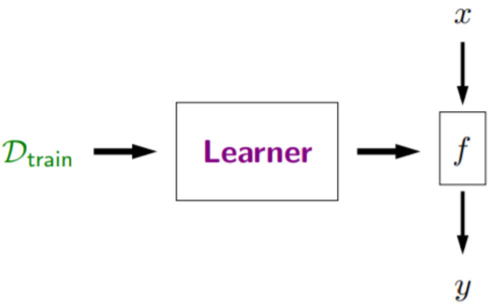
Road Map

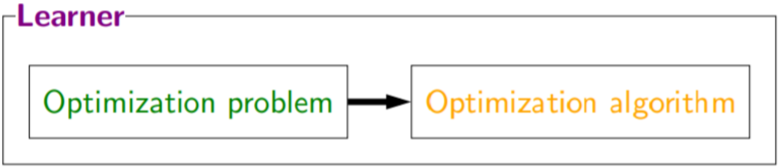
Linear predictors

$\boxed{\text{Learning objective}}$

Optimization

Linear Classifier: Overall Framework





Linear Classifier: Loss Function

Definition: **Loss function**.

A loss function $L(x, y, \boldsymbol{w})$ quantifies how *unhappy* you would be if you use $\boldsymbol{w}$ to make a prediction on x when the correct output is y.

Our goal is to find the best parameters $\boldsymbol{w}$ that can minimize the loss function.

Linear Classifier: Score and Margin

Predicted label: $y' = f_{\boldsymbol{w}}(x) = \text{sign}(\boldsymbol{w}^{\intercal} \cdot \boldsymbol{x})$
Ground-truth label: y
Example: $\boldsymbol{w} = [2, -1]$, $\boldsymbol{x} = [2, 0]$, $y = -1$

Definition: **Prediction score**.

The prediction score on an example (x, y) is $\boldsymbol{w}^{\intercal} \cdot \boldsymbol{x}$, i.e., how confident we are in prediction.

Definition: **Margin**.

The margin on an example (x, y) is $(\boldsymbol{w}^{\intercal} \cdot \boldsymbol{x})y$, i.e., how correct we are.

Linear Classifier: Score and Margin

When does a binary classifier mis-classify an example?

- margin less than 0
- margin greater than 0
- score less than 0
- score greater than 0

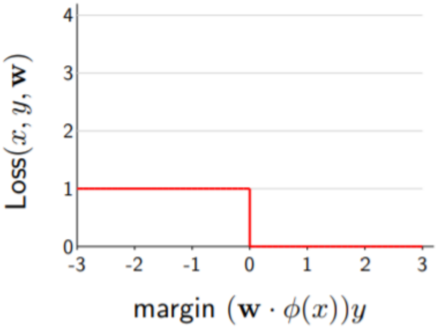
Linear Classifier: Loss function

Definition: **Zero-One loss**.

$$
\begin{aligned}
L_{0-1}(x, y, \boldsymbol{w}) &= \mathbb{1}[y' \neq y] && (4)\\
&= \mathbb{1}[f_{\boldsymbol{w}}(x) \neq y] && (5)\\
&= \mathbb{1}[(\boldsymbol{w}^{\intercal} \cdot \boldsymbol{x})y \leq 0] && (6)
\end{aligned}
$$

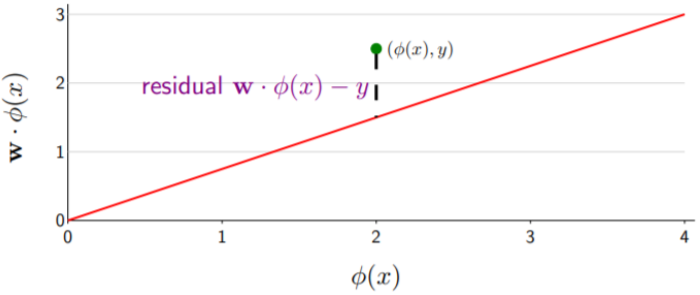
Here $\mathbb{1}$ is the indicator function, where $\mathbb{1}[\mathit{True}] = 1$ and $\mathbb{1}[\mathit{False}] = 0$.

Linear Classifier: Loss function



$$L_{0-1}(x, y, \boldsymbol{w}) = \mathbb{1}[(\boldsymbol{w}^\intercal \cdot \boldsymbol{x})y \leq 0]$$

Linear Classifier → Linear Regression



Definition: **Residual**.

The residual is $(\boldsymbol{w}^\top \cdot \boldsymbol{x}) - y$, the amount by which prediction $f_{\boldsymbol{w}}(x) = \boldsymbol{w}^\top \cdot \boldsymbol{x}$ deviates from the target y.

- Define classification errors as regression errors.

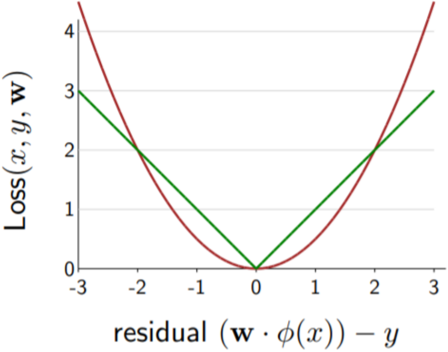
Linear Regression: Squared loss

Definition: **Squared loss**.

$$L_{squared}(x, y, \boldsymbol{w}) = (y' - y)^2 \tag{7}$$
$$= (\boldsymbol{w}^\intercal \cdot \boldsymbol{x} - y)^2 \tag{8}$$

Example: $\boldsymbol{w} = [2, -1]$, $\boldsymbol{x} = [2, 0]$, $y = -1$
$L_{squared}(x, y, \boldsymbol{w}) = ?$

Linear Regression: Losses



$$L_{squared}(x, y, \boldsymbol{w}) = (\boldsymbol{w}^\top \cdot \boldsymbol{x} - y)^2 \times \frac{1}{2}$$

$$L_{absdev}(x, y, \boldsymbol{w}) = |\boldsymbol{w}^\top \cdot \boldsymbol{x} - y|$$

Loss Minimization

So far: for one instance, $L(x, y, \boldsymbol{w})$ is easy to minimize.

How to set $\boldsymbol{w}$ to make *global* trade-offs?

- Not every instance can be happy.
- Try to make more instances happy.

Definition: **Training loss**.

$$L_{train}(\mathcal{D}_{train}, \boldsymbol{w}) = \frac{1}{|\mathcal{D}_{train}|} \sum_{(x,y)\in\mathcal{D}_{train}} L(x, y, \boldsymbol{w}) \tag{9}$$

The learning objective:

$$\min_{\boldsymbol{w}} \; L_{train}(\mathcal{D}_{train}, \boldsymbol{w}) \tag{10}$$

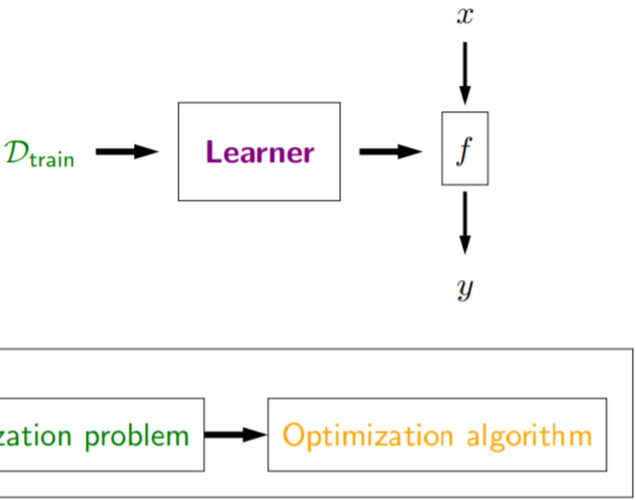
Road Map

Linear predictors

Learning objective

Optimization

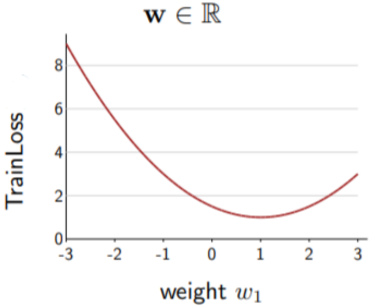
Linear Classifier: Optimization



Linear Classifier: Optimization

The learning objective:

$$\min_{\boldsymbol{w}} \; L_{train}(\mathcal{D}_{train}, \boldsymbol{w})$$



$\mathbf{w} \in \mathbb{R}^2$

Linear Classifier: Optimization

The learning objective:

$$\min_{\boldsymbol{w}} \; L_{train}(\mathcal{D}_{train}, \boldsymbol{w})$$

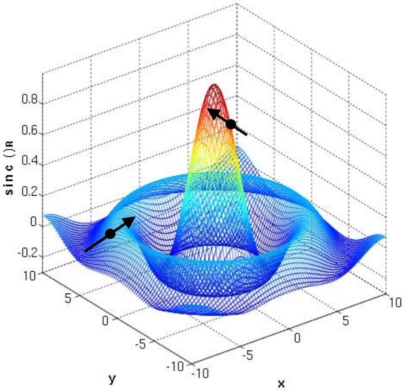
How to optimize it?

- Analytic solution (not always applicable).
- A more general solution is needed.
- **Gradient descent**.
  - An iterative algorithm.
  - At each iteration, find the fastest direction of moving $\boldsymbol{w}$ that decreases L_{train}.
  - Move $\boldsymbol{w}$ towards that direction.
  - Repeat the above two steps until reaching a stable $\boldsymbol{w}$.

Gradient

Definition: **Gradient**.

The gradient $\nabla_{\boldsymbol{w}} L_{train}$ is the direction that *increases* the training loss the most.



Gradient Descent

Algorithm Gradient Descent

Initialize $\boldsymbol{w}$;
for $t = 1, 2, ..., T$ **do**
$\quad \boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} L_{train}(\mathcal{D}_{train}, \boldsymbol{w})$
end

η: step size

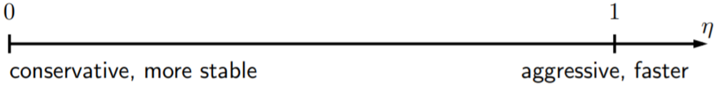
T: number of iterations

Gradient Descent

Algorithm Gradient Descent

Initialize $\boldsymbol{w}$;
for $t = 1, 2, \ldots, T$ **do**
  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} L_{train}(\mathcal{D}_{train}, \boldsymbol{w})$
end

How to choose the step size η?



Common strategies:

- Constant: e.g., $\eta = 0.01$.
- Decreasing: e.g., $\eta = 0.1/t$.

Gradient Descent

Algorithm Gradient Descent

Initialize $\boldsymbol{w}$;
for $t = 1, 2, \ldots, T$ **do**
  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} L_{train}(\mathcal{D}_{train}, \boldsymbol{w})$
end

Example: Least squares regression

$$L_{train}(\mathcal{D}_{train}, \boldsymbol{w}) = \frac{1}{|\mathcal{D}_{train}|} \sum_{(x,y) \in \mathcal{D}_{train}} (\boldsymbol{w}^{\intercal} \cdot \boldsymbol{x} - y)^2$$

$$\nabla_{\boldsymbol{w}} L_{train}(\mathcal{D}_{train}, \boldsymbol{w}) = \frac{1}{|\mathcal{D}_{train}|} \sum_{(x,y) \in \mathcal{D}_{train}} 2(\boldsymbol{w}^{\intercal} \cdot \boldsymbol{x} - y)\boldsymbol{x}$$

Gradient Descent

Algorithm Gradient Descent

Initialize $\boldsymbol{w}$;
for $t = 1, 2, \ldots, T$ **do**
  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} L_{train}(\mathcal{D}_{train}, \boldsymbol{w})$
end

Gradient descent is slow!

- Each iteration requires going over all training examples.
- Costly when the dataset is large (which is common nowadays).

Stochastic Gradient Descent

Algorithm Stochastic Gradient Descent

Initialize $\boldsymbol{w}$;
for t = *1, 2, ..., T* **do**
    for $(x, y) \in \mathcal{D}_{train}$ **do**
        $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} L(x, y, \boldsymbol{w})$
    end
end

The key idea: It's not about quality, it's about quantity.

Linear Classifier: Summary

- Linear predictor (model architecture):

$$f_{\boldsymbol{w}}(x) \text{ based on } \boldsymbol{w}^{\intercal} \cdot \phi(x).$$

- Learning objective (goal):

$$\min_{\boldsymbol{w}} L_{train}(\mathcal{D}_{train}, \boldsymbol{w}).$$

- Optimization (train the model towards the goal):

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} L_{train}(\mathcal{D}_{train}, \boldsymbol{w})$$