CSIT5900
Machine Learning (Supervised Learning) - The Ideas

Fangzhen Lin

Department of Computer Science and Engineering
Hong Kong University of Science and Technology

ML tasks

- Classification: classify the given input (data) into one of the pre-defined classes. Examples: Is the mail spam/not spam? Is the applicant qualified/not qualified? Which meat is the dish made of, chicken/pork/beef/lamb/duck? Which digit is the handwritten one most likely stands for?
  - Binary (two-class, binomial) classification: decide whether the input is in the class or not.
  - Multi-class classification: decide which class the input belongs to.
- Regression: predict a value from the given input. Examples: predict the stock price of a given company on a given date; predict the electricity bill of a future month.

Feature Extraction

As in designing agents, instead of using the raw input, we extract/compute features that are relevant to the task in hand.

- To predict which direction the robot should move given the readings from the sonar rings, we extract/compute 8 features $s_1, ..., s_8$ corresponding to the 8 surrounding directions.
- To classify if a given string is an email address, we extract features such as "presence of @", "endWith .com", "endWith of .org", etc.
- What features to use for classifying a given email into spam/not spam?
- What features to use for classifying a given image (a matrix of pixels) into pretty/not pretty?

Feature Vectors

If we have n features h_1,...,h_n for an input x, then we can represent a state by a dictionary:

$$\{h_1 : v_1, ..., h_n : v_n\}$$

where v_i is the value of the feature h_i. Example: for a string input "abc@google.com", we may have:

$$\{\text{presence of @} : 1,\ \text{endWith .com} : 1,\ \text{endWith .org} : 0\}$$

Given that the feature names are fixed, there is no need to have them, so a state can be represented by a *feature vector*: for an input x, its feature vector is $\phi(x) = [\phi_1(x), ..., \phi_n(x)]$, where $\phi_i(x)$ is the value of the *i*th feature of x.

Weight Vector and Score

An *n*-dimension weight vector $\boldsymbol{w}$ is a an *n*-vector of real numbers. *Assuming* features are also real-valued, then the score of input x under weight vector $\boldsymbol{w}$ is the the weighted sum of its features:

$$score(x, \boldsymbol{w}) = \boldsymbol{w} \cdot \phi(x) = \sum_{i=1}^{n} w_i \phi_i(x).$$

Linear Predictors

- Prediction: given input x, predict output y.
- Linear predictors: use $score(x, \boldsymbol{w}) = \boldsymbol{w} \cdot \phi(x)$ to predict y.
  - Binary classifier: output $sign(score(x, \boldsymbol{w}))$, where

$$sign(u) = \begin{cases} 1, & \text{if } u > 0 \\ 0, & \text{if } u = 0 \\ -1, & \text{if } u < 0 \end{cases}$$

    Meaning: the input x is in the class if the output is 1, not in the class if the output is -1, and don't care/can't tell/flip a coin if the output is 0.
  - Progression: just use $score(x, \boldsymbol{w})$.
- It's called a linear predictor because the relationships between the score and the features are linear.

Linear Predictors

How good a linear predictor can be depends crucially on if you have a right set of features - remember that it's linear in the features.
Consider the input x to be your body temperature (in Celsius), and the output y to be that smaller it is, healthier you are.

Linear Predictors

Some possible equations for y:

$$y = |x - 37| \text{ or } y = (x - 37)^2.$$

None of them is linear! So we need to invent some features so that the output is linear in the features!
Consider

$$y = (x - 37)^2 = x^2 - 74x + 37^2.$$

So we introduce two features: $\phi_1(x) = x$ and $\phi_2(x) = x^2$. Now y is linear in $\phi(x)$!
Question: What features would make exclusive or linear?

Supervised Learning

- Goal: learn a function of x: $y = f(x)$.
- Supervised learning: learn a function of $\phi(x)$: $y = f(\phi(x))$ from a training set $\mathcal{D}_t$ of feature vectors and their labels:

$$\mathcal{D}_t = \{[\phi(x_1), y_1], ..., [\phi(x_k), y_k]\}.$$

- Learning linear predictor: learn a weight vector $\boldsymbol{w}$ so that $sign(\boldsymbol{w} \cdot \phi(x))$ (binary classifier) or $\boldsymbol{w} \cdot \phi(x)$ (regression) agrees well with the training set.

Data

- Supervised learning is as good as your training set.
- A good training set needs to have a good coverage.
- Learning is more than just fitting the data from the training set, it uses data to come up with a generalization.
- What good is a perfect match like the following (rote learning): for any x, if for some y $[\phi(x), y]$ is in $\mathcal{D}_t$ then output y else pick a random answer (or say don't know).
- Split your data into three sets: training set, for training your function; validation set, for fine tuning your trained function; and test set, for testing how good your learned function is.

Learning as Optimization

- Recall that linear predictor uses *sign*(*score*($x, \boldsymbol{w}$)) (binary classifier) or just *score*($x, \boldsymbol{w}$) (regression) to make prediction:

$$score(x, \boldsymbol{w}) = \boldsymbol{w} \cdot \phi(x) = \sum_{i=1}^{n} w_i \phi_i(x).$$

- So it all comes down to learning the weight vector from $\mathcal{D}_t$.
- One way is to cast it as the problem of minimizing *training loss*: compute the weight vector $\boldsymbol{w}^*$ so that

$$\boldsymbol{w}^* = arg\ min_{\boldsymbol{w}} \text{Loss}(\mathcal{D}_t, \boldsymbol{w}).$$

Loss

Loss($\mathcal{D}_t, \boldsymbol{w}$) is supposed to measure the "loss", the "differences" between the desired output and the actual output using $\boldsymbol{w}$ for those in the training set $\mathcal{D}_t$. It's normally the average of the individual losses:

$$\text{Loss}(\mathcal{D}_t, \boldsymbol{w}) = \frac{1}{|\mathcal{D}_t|} \sum_{(x,y)\in\mathcal{D}_t} \text{Loss}(x, y, \boldsymbol{w}),$$

where Loss($x, y, \boldsymbol{w}$) is the "loss", or the "difference" between the desired output (y) and the predicted output using $\boldsymbol{w}$.

Loss

For regression, two popular loss functions are

- Absolute difference loss:

$$\text{Loss}_{abs}(x, y, \boldsymbol{w}) = |score(x, \boldsymbol{w}) - y|.$$

- Squared loss:

$$\text{Loss}_{sq}(x, y, \boldsymbol{w}) = (score(x, \boldsymbol{w}) - y)^2.$$

Loss

Consider an extreme case of $\mathcal{D}_t = \{(1,1),(1,0),(1,11)\}$, and

$$\boldsymbol{w}^* = arg\ min_{\boldsymbol{w}} \text{Loss}(\mathcal{D}_t, \boldsymbol{w}).$$

- Under Loss_{abs}, $\boldsymbol{w}^* = 1$ (median). The median is less sensitive to some outliers.
- Under Loss_{sq}, $\boldsymbol{w}^* = 12/3$ (mean). The mean tries to take into account all. This is a popular loss function because it's easier to use.

Loss

Consider $\text{Loss}_{sq}(\mathcal{D}_t, \boldsymbol{w})$:

$$\begin{aligned}
&= 1/|\mathcal{D}_t| \sum_{(x,y)\in\mathcal{D}_t} \text{Loss}_{sq}(x, y, \boldsymbol{w}) \\
&= 1/3 \sum_{(x,y)\in\mathcal{D}_t} (score(x, \boldsymbol{w}) - y)^2 \\
&= 1/3[(score(1, \boldsymbol{w}) - 1)^2 + (score(1, \boldsymbol{w}) - 0)^2 + (score(1, \boldsymbol{w}) - 11)^2] \\
&= 1/3[(w - 1)^2 + w^2 + (w - 11)^2]
\end{aligned}$$

Thus to compute

$$\boldsymbol{w}^* = arg\ min_{\boldsymbol{w}} \text{Loss}(\mathcal{D}_t, \boldsymbol{w}).$$

we compute the derivative of $\text{Loss}_{sq}(\mathcal{D}_t, \boldsymbol{w})$ and set it to 0:

$$(w - 1) + w + (w - 11) = 0.$$

Loss for Binary Classification

Margin: Given input x, its feature vector $\phi(x)$, the current weight vector $\boldsymbol{w}$, and its correct label y, the margin on (x, y) is

$$\begin{aligned} margin(x, y, \boldsymbol{w}) &= [\boldsymbol{w} \cdot \phi(x)]y \\ &= score(x, \boldsymbol{w})y. \end{aligned}$$

The margin is for binary classifiers, and measures how correct (incorrect) the current classification is.
A corresponding loss function, called zero-one loss, is the following Boolean valued function:

$$Loss_{0-1}(x, y, \boldsymbol{w}) = margin(x, y, \boldsymbol{w}) \leq 0.$$

Algorithms

Once a loss function is chosen, it is time to compute the weight vector that will minimize it over the training set. This is a computationally hard problem, and we can try a local search ("hill climbing" for maximizing and "descending a canyon" for minimizing) for it:

```
Initialize $\boldsymbol{w}$ (e.g. 0);
while $C$ {
    $\boldsymbol{w} = successor(\boldsymbol{w})$;
    update $C$; }
```

where C is the condition for keep updating the weights, e.g. it can be $i \leq N$ for some fixed N or could be a condition about the desired accuracy; $successor(\boldsymbol{w})$ returns a "better" $\boldsymbol{w}$ and is computed using training instances and the loss function.
In ML, two popular ways of "descending the canyon" are gradient descend and stochastic gradient descend.

Gradient descend (GD)

In gradient descend, we use the entire training set $\mathcal{D}_t$ to compute the gradient of the loss function on the current weight vector and decrease the weight by a "fraction" of it:

$$\textit{successor}(\boldsymbol{w}) = \boldsymbol{w} - c\nabla_{\boldsymbol{w}}\text{Loss}(\mathcal{D}_t, \boldsymbol{w}),$$

where c is the learning rate or step size.
Intuitively, $\nabla_{\boldsymbol{w}}\text{Loss}(\mathcal{D}_t, \boldsymbol{w})$ is the direction that increases the loss the most.

Gradient descend - least squared loss

Recall

$$\begin{aligned}
\text{Loss}_{sq}(\mathcal{D}_t, \boldsymbol{w}) &= \frac{1}{|\mathcal{D}_t|} \sum_{(x,y)\in\mathcal{D}_t} \text{Loss}_{sq}(x, y, \boldsymbol{w}) \\
&= \frac{1}{|\mathcal{D}_t|} \sum_{(x,y)\in\mathcal{D}_t} (score(x, \boldsymbol{w}) - y)^2 \\
&= \frac{1}{|\mathcal{D}_t|} \sum_{(x,y)\in\mathcal{D}_t} (\boldsymbol{w} \cdot \phi(x) - y)^2
\end{aligned}$$

Now use chain rule to compute the gradient:

$$\nabla_{\boldsymbol{w}} \text{Loss}_{sq}(\mathcal{D}_t, \boldsymbol{w}) = \frac{1}{|\mathcal{D}_t|} \sum_{(x,y)\in\mathcal{D}_t} 2(\boldsymbol{w} \cdot \phi(x) - y)\phi(x).$$

This is slow when $\mathcal{D}_t$ is large!

Stochastic gradient descend (SGD)

Same idea but instead of doing a pass on the entire training set in each step, it go through training instances one at a time:

```
$successor(\boldsymbol{w}, x, y) = \boldsymbol{w} - c\nabla_{\boldsymbol{w}}\text{Loss}(x, y, \boldsymbol{w})$,
For each $(x, y) \in \mathcal{D}_t$ :
    $\boldsymbol{w} = successor(\boldsymbol{w}, x, y)$
```

There are reports that in practice, doing one pass over the training instances with SGD, often performs comparably to taking ten passes over the data with GD.

(Artificial) Neural Networks

- An (artificial) neural network is a directed graph, most often acyclic.
- The sources (no incoming arcs) are inputs and the targets (no outgoing arcs) are outputs. The internal nodes represent "hidden" features.

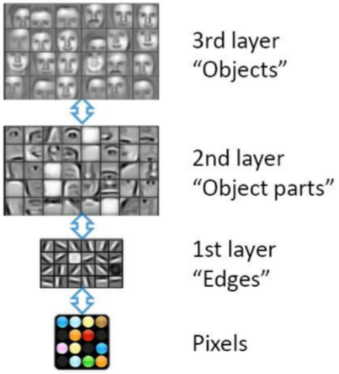

Figure: From Percy Liang's AI note