Machine learning

Lecture 4: Linear classifiers and regression models

Aleksei Platonov
DingTalkID: aplatonov

13 April 2020
HDU

- The intuition behind linear models.
- Gradient descent numerical optimization algorithm.
- Linear classifiers and regression models.
- Tuning training process and model structure.

The simplest models

Linear functions for regression task

Machine learning task reminding:

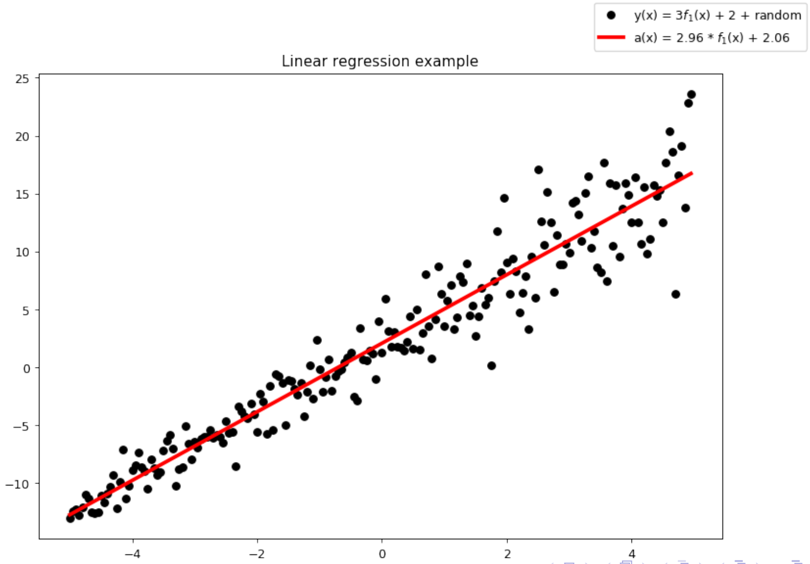
- We have training set $X^l = (x_i, y_i)_{i=1}^l$.
- We need to deduce a function $y_i = y(x_i)$ and generate an algorithm $y_i \approxeq a(x_i)$
- We already know two ML approaches: kNN and logical rules.

In linear models we try to find pure functional relationship between objects and answers.
And linear regression models are the simplest models of regression:

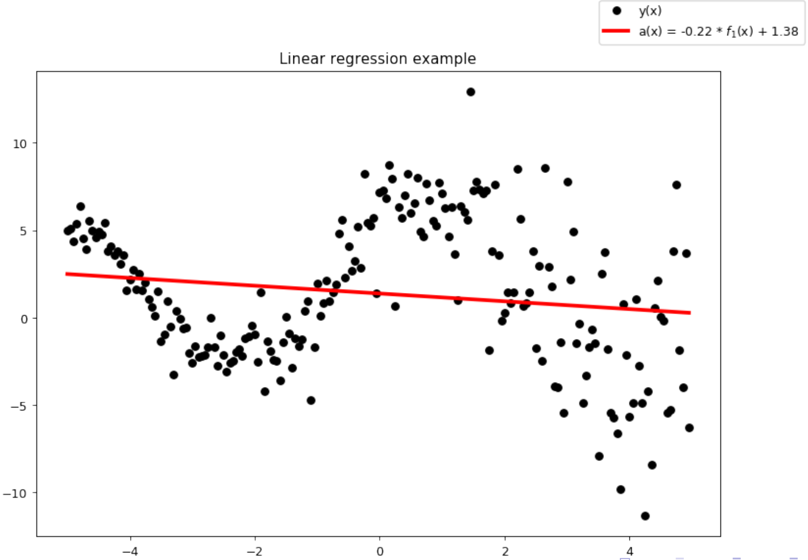
$$a(x, w) = \sum_{j=1}^{n} w_j f_j(x) - w_0,$$

where w_0 is a bias of linear function, $f_1(x), \ldots, f_n(x)$ - feature functions, $w_1, \ldots, w_n$ - features' weights

What does it look like?



What does it look like?



Separating hyperplane for classification

Machine learning task reminding:

- We have training set $X^l = (x_i, y_i)_{i=1}^l$.
- We need to deduce a function $y_i = y(x_i)$ and generate an algorithm $y_i \approxeq a(x_i)$

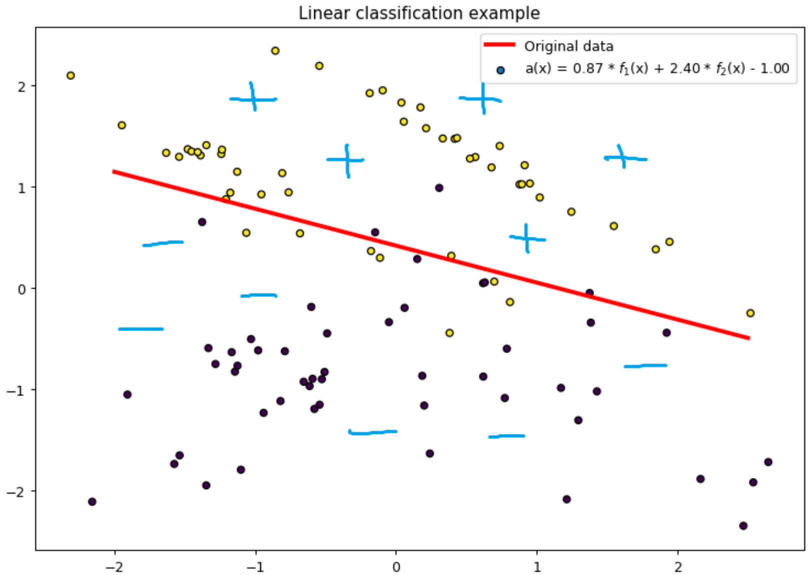
$$a(x, w) = sign(\sum_{j=1}^{n} w_j f_j(x) - w_0),$$

where $sign(x) = 1$ if $x > 0$ else $sign(x) = -1$
We can simplify formula by adding fake feature $f_0(x) = -1$, then we can construct weights vector $w = [w_0, w_1, \ldots, w_n]$:

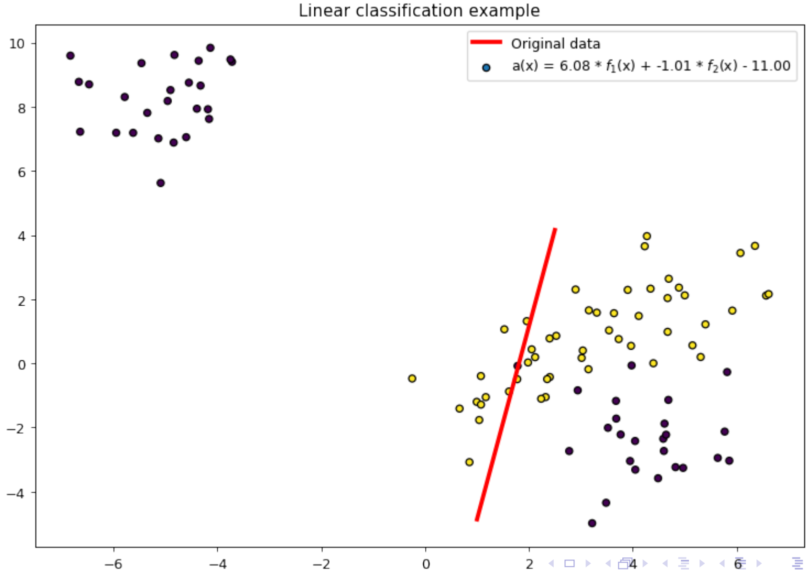
$$a(x, w) = sign(\langle w, x \rangle),$$

where $\langle u, v \rangle$ means dot product of two vectors.

What does it look like?



What does it look like?



Optimization task for linear classifier

- As for kNN we can define the margin of classifier. If $Y = -1, +1$:

$$M(x_i) = y_i f(x_i, w) = y_i \cdot \langle w, x_i \rangle,$$

so, if classifier returned a right answer the margin will be > 0 and < 0 if classifier was wrong.

- If $M(x_i) \gg 0$ then it is prototype-like object. The algorithm has high confidence level in this example.
- If $M(x_i) \ll 0$ then it could be outlier.
- If $|M(x_i)| < \delta_0$ where $delta_0$ is value close to zero, then the algorithm has low confidence level, it could be an error.

Optimization task for linear classifier

So, we can introduce an optimization problem explicitly:

$$Q(w) = \sum_{i=1}^{l} [M(x_i) < 0] \leq \widetilde{Q}(w) = \sum_{i=1}^{l} \mathcal{L}(M_i(w)) \to \min_w$$

What is $\mathcal{L}$?

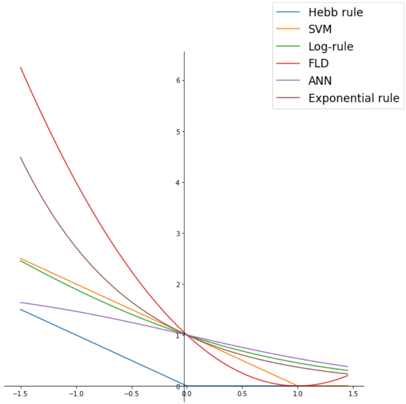
- $\mathcal{L}$ is a positive definite and nonincreasing function.
- $\mathcal{L}$ should be proportional to error function.
- $\mathcal{L}$ should be differentiable.

So, $\mathcal{L}$ is a loss function.

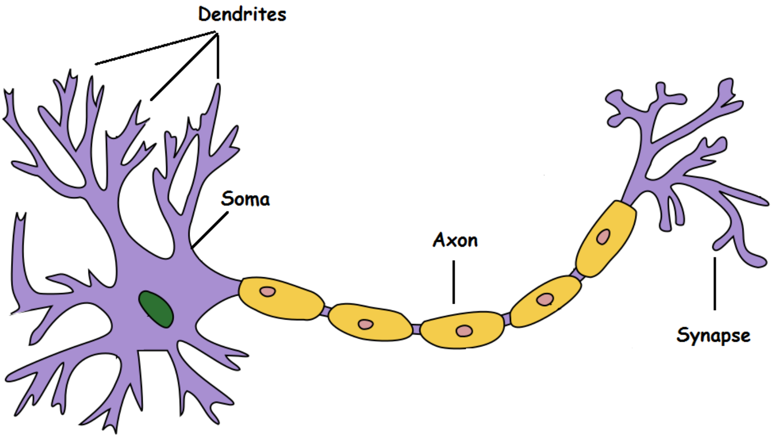
Approximations for loss function

What's the form of loss function?

- Hebb rule: $H(M) =$ if $M < 0$ then $-M$ else 0
- SVN rule: $SVM(M) =$ if $M < 1$ then $1 - M$ else 0
- Log-rule: $L(M) = log_2(1 + e^{-M})$
- FLD: $FLD(M) = (1 - M)^2$
- ANN: $ANN(M) = \frac{2}{1+e^M}$
- Exponential rule: $E(M) = e^{-M}$



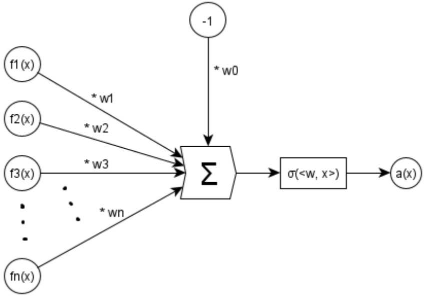
Biological neuron



[1] https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

The perceptron model

- The perceptron model is a mathematical model of a brain's neuron.



- What does it look like? Like the linear model!

$$a(x, w) = \sum_{j=1}^{n} w_j f_j(x) - w_0,$$

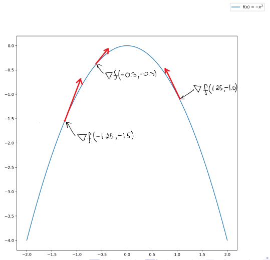
Gradient Descend: numerical optimization method

- The gradient of a function is a vector of partial derivatives. Definition:

$$\nabla f(x) = \left[\frac{\partial f(x)}{\partial x_i}\right]_{i=1}^{n}$$

- For example:
  1. if $f(x) = x_1^2 + x_2^3 + x_1 x_2$ then $\nabla f(x) = \left[2x_1 + x_2, 3x_2^2 + x_1\right]$
  2. if $f(x) = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$ then $\nabla f(x) = [w_1, w_2, \ldots, w_n]$

The gradient of function matches with the vector of maximal increasing of function.



Gradient Descend: numerical optimization method

- If the gradient == 'vector of function increasing' $\to$ then the anti-gradient $-\nabla f(x)$ == 'vector of function decreasing'.
- It gives us a numerical minimization method:

$$w^{(t+1)} = w^{(t)} - \eta \nabla_w f(w, x)$$

- Remind we have a task:

$$\widetilde{Q}(w, X^l) = \sum_{i=1}^{l} \mathcal{L}(M_i(w)) \to \min_w$$

- So we can use the gradient descent:

$$w^{(t+1)} = w^{(t)} - \eta \nabla_w \widetilde{Q}(w, X^l)$$

Stochastic Gradient Descent

But we have a problem: If the training set is too big, then the compute of one gradient step is really expensive because $\widetilde{Q}(x)$ sums errors in all objects in X^l.

The idea of stochastic gradient descent:

- The gradient of function is a linear operator:

$$\nabla(f_1(x) + f_2(x)) = \nabla f_1(x) + \nabla f_2(x)$$

- We can compute $\nabla \widetilde{Q}(x)$ partially because:

$$\nabla \widetilde{Q}(x) = \nabla \sum_{i=1}^{l} \mathcal{L}(M_i(x)) =$$

$$= \nabla \sum_{i=1}^{k_1} \mathcal{L}(M_i(x)) + \dots + \nabla \sum_{i=k_{m-1}}^{k_m} \mathcal{L}(M_i(x))$$

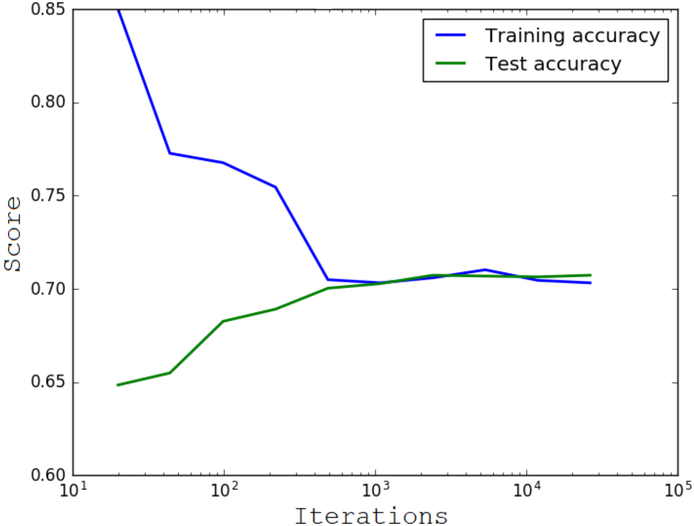
Stochastic Gradient Descent

So, we can approximate real gradient using just part of data:

Algorithm 1 Stochastic gradient descent algorithm

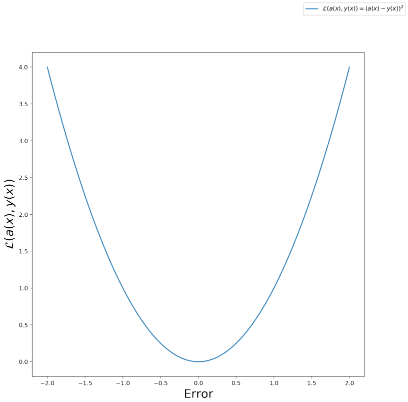
procedure $\mathrm{SGD}(X^l, \eta, \lambda, \delta)$
    $w_0 \leftarrow \textit{generate\_initial\_w}_0()$
    $\widetilde{Q}^{(-1)} \leftarrow \inf$
    $\widetilde{Q}^{(0)} \leftarrow \widetilde{Q}(w_0, X^l)$
    $t \leftarrow 0$
    while $|\widetilde{Q}^{(t)} - \widetilde{Q}^{(t-1)}| < \delta$ **do**
        $(x_i, y_i) \leftarrow \textit{select\_random\_object}(X^l)$
        $\epsilon_i \leftarrow \mathcal{L}(< x_i, w_t > \cdot y_i)$
        $w_{t+1} \leftarrow w_t - \eta \nabla \mathcal{L}(\langle x_i, w_t \rangle \cdot y_i)$
        $\widetilde{Q}^{(t+1)} \leftarrow (1-\lambda)\widetilde{Q}^{(t)} + \lambda \epsilon_i$
    return w_t

Learning curves



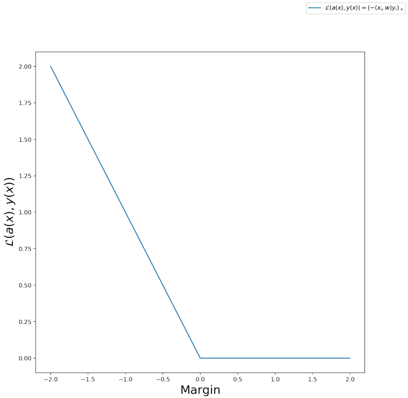
Special case: ADALINE (Linear regression model)

- Regression task: $X \subseteq \mathbb{R}^{n+1}, y \subseteq \mathbb{R}$ ($n+1$ for w_0 element)
- Loss function for regression: $\mathcal{L}(a, x, y) = (a(x) - y(x))^2$
- Gradient step, the delta-rule: $w^{(t+1)} = w^{(t)} - \eta(\langle w, x_i \rangle - y_i)x_i$



Special case: Hebb rule

- Classification task: $X \subseteq \mathbb{R}^{n+1}, y = -1, 1$ ($n+1$ for w_0 element)
- Loss function for classification: $\mathcal{L}(a, x, y) = (-\langle x_i, w \rangle y_i)_+$
- Gradient step:
  if $-\langle x_i, w \rangle y_i < 0$
  then $w^{(t+1)} = w^{(t)} = \eta x_i y_i$
  else $w^{(t+1)} = w^{(t)}$



Novikov's theorem

If we try to solve classification task with $Y = -1, 1$ and training set is linearly separable then:

- SGD for Hebb rule will train a very precise w_i (without errors).
- It will work for any η and w_0.
- It will be completed after the finite number of iterations.
- It's independent of a selection ordering of objects from the dataset.

w_0 initialization

- $w_j = 0$
- $w_j = random(-\frac{1}{2n}, \frac{1}{2n})$, where n - the number of features
- $w_j = \frac{\langle y, f_j \rangle}{\langle y, f_j \rangle}$, where f_j - feature vector, $f_j = (f_j(x_i))_{i=1}^{l}$
  Good for regression
- Find initial w_0 by pre-training.
- Generate several random vectors w_0 and select the best.

Order of objects from training set for SGD

- The X^l dataset shuffling: need to alternate objects from different classes.
- Get objects with high loss function value:
  - The larger $-M_i(x)$, the greater probability of getting x_i
  - The larger $M_i(x)$, the less probability of getting x_i
- Don't use objects with very high $M_i(x)$ at all ($M_i(x) \gg 0$).
- Don't use objects with very small $M_i(x)$ at all ($M_i(x) \ll 0$).

Selection of learning rate value

- For convex $\widetilde{Q}(w)$ function we can use $\eta_t \to 0$, i.e. $\eta = \frac{1}{t}$
  But $\widetilde{Q}(w)$ is not usually convex
- Steepest descent method:

  $$Q(w - \eta\nabla(W)) \to \min_{\eta},$$

  for example: the method of interval bisection, Newton's method.
- Random probes of η values and choosing the best value.

Logistic regression as classification

- Classification function of special form:

$$f(w,x) = \sigma(w,x) = \frac{1}{1+e^{-y\langle w,x\rangle}}$$

- Sigmoid $\sigma(z)$ approximates function $sign(z)$ and it has the special form of Q:

$$Q(w) = \sum_{i=1}^{l} ln(1+e^{-y_i\langle w,x_i\rangle})$$

- And the gradient will be:

$$\widetilde{Q}(w) = -\sum_{i=1}^{l} \sigma(-y_i\langle w,x_i\rangle)y_i x_i$$

- Logistic regression has a great property: $P(y|x) = \sigma(y\langle w,x\rangle)$

Multiclassification

- Our already built classifier solves just binary classification task. We cannot create hyperplane for separation more than two classes.
- But we have solution. One-vs-Rest classifier:

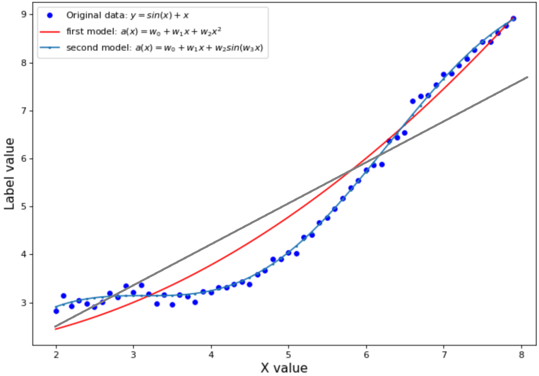
Algorithm 2 One vs. All training

1: **procedure** $\mathrm{OVA\_SGD}(X^l, \eta, \lambda, \delta)$
2: $\quad$ *models* $\leftarrow$ []
3: $\quad$ **for** $y \in Y$ **do**
4: $\quad\quad$ $X^{l'} \leftarrow \{(x_i, \text{ if } y_i = y \text{ then } 1 \text{ else } -1) : (x_i, y_i) \in X^l\}$
5: $\quad\quad$ models.append(SGD($X^{l'}, \eta, \lambda, \delta$))
6: $\quad$ **return** new OveVsAll(models)

- And predict for object u will be: $y_u = arg \max_{a,y} P_a(y|u)$, where $a \in ova.models$

Non-linear regression from linear model

- We have another problem: our model is linear.
- But we can transform linear regression (or classifier) to non-linear by adding new feature functions:



Reasons of overfitting

1. Too many features and few objects in training dataset.
2. Linear dependence of feature columns:
   - Let's we have a classifier: $a(x, w) = sign(\langle w, x \rangle)$
   - There is a linear dependence between features, so: $\exists v \in \mathbb{R} : \langle v, x \rangle$.
   - Then $\forall \gamma \in \mathbb{R} \Rightarrow a(x, w) = a(x, w + \gamma v)$.
   - We can increase the weights' vector endlessly.

How can we detect overfitting?

- Too big values of weights in vector w.
- Unstable of $a(x, w)$ answers.
- $Q(X^l) \ll Q(X^k)$, where $Q(X^k)$ is a test set.

Solution: Weight vector decay

Let's add a penalty for the big values of weights' vector:

$$Q_\tau(w, X^l) = Q(w, X^l) + \frac{\tau}{2}||w||^2 \to \min_w$$

Then the gradient will be:

$$\nabla Q_\tau(w, X^l) = \nabla Q(w, X^l) + \tau w \to \min_w$$

And the gradient step:

$$w^{(t+1)} = w^{(t)}(1 - \eta\tau) - \eta\nabla Q(w)$$

How an we choose τ?

- Leave-one-out method as minimization algorithm.
- Generate a set of random values of τ and choose the best.

Conclusions

Today we learned:

- A simplest approach in machine learning - linear model for classification and regression.
- Gradient descent and stochastic gradient descent method of numeric optimization.
- How to transform linear model to non-linear and how to adapt binary classifier to multiclassification task.
- The reasons of overfitting in the linear models and method to avoiding of it.