MATH 565: Lecture 7 (02/03/2026)

Today: * optimization in general v/s in ML
* cross validation
* SGD details

Optimization in general v/s Optimization in ML

general

1. objective function need not be separable

2. Optimize as much as possible

3. Worst performance on outside/test-data; as the model is overfitted to training data

4. Work with the original objective function.

ML

1. loss functions are usually additively separable, e.g.,
$$J(\bar{w}) = \frac{1}{2}\|D\bar{w}-\bar{y}\|^2 = \sum_{i=1}^{n}(x_i^T\bar{w}-y_i)^2$$

2. Avoid overfitting — stop when "performance" on test-data is high enough (or use cross validation)

3. better performance on test-data; because of stochasticity (from SGD), regularization, etc.

4. Loss function J usually modified by adding regularization term(s)
e.g., Tikhonov regularization
$$J(\bar{w}) = \frac{1}{2}\|D\bar{w}-\bar{y}\|^2 + \frac{1}{2}\|\bar{w}\|^2$$

Hyperparameter Tuning

e.g. SVM

$$J_{L_1\text{-SVM}} = C\sum_{i=1}^{n}\xi_i + \frac{\lambda}{2}\|\vec{w}\|^2$$

variables are ξ_i, w_i; but also called as parameters in some settings.

hyperparameters $(C>0, \lambda>0)$

We use grid search to identify best C, λ values.

e.g., $C = \{0.01, 0.1, 1, 10, 100, 1000\}$ → 6 values

$\lambda = \{0.001, 0.01, 0.1, 1, 10\}$ → 5 values

For each (C_i, λ_j) choice, solve the optimization problem $\min_{\vec{w}} J$.

The only requirements are that $C>0$ and $\lambda>0$. Hence, we naturally do not want to try too many values. Here is a typical strategy.

* Start with a coarse grid, identify one pair of (C_i, λ_j) values that "perform best"
* do a finer grid search around this (C_i, λ_j) pair.

e.g., Let $C=10$, $\lambda=0.1$ be identified in the coarse grid search. Then, we explore $C=\{6.5, 7, \ldots 10, \ldots 13.5\}$ and $\lambda = \{0.05, 0.06, .., 0.1, \ldots, 0.15\}$, for instance, and identify a (possibly) new optimal pair.

Can use k-fold cross validation (see next page...)

Note: We could also do sampling of C, λ values rather than run through grids of candidate values.

k-fold Cross Validation (CV)

$k=10$ is a widely used option

Let $R[n]$ = random permutation of $\{1,\dots,n\}$.
$= \{r_1,\dots,r_n\}$, where each $r_i \in \{1,\dots,n\}$.

Form k folds (subsets) $\{F_i\}_{i=1}^{k}$ of $R[n]$, each with $p \approx \lfloor \frac{n}{k} \rfloor$ points.

e.g., $F_i = \{ r_{(i-1)p+1}, \dots, r_{ip} \}$, $i = 1, \dots, k-1$, and

$F_k = \{ r_{(k-1)p+1}, \dots, r_n \}$. → the last fold may have a slightly different # entries.

e.g., $n = 256$, $k = 10$ ⇒ $p = 26$, but last fold has 22 pts.

for $i = 1 .. k$

$T_i = R[n] \setminus F_i$ (training set for iteration i)

$\bar{w}_i = \text{argmin } J_i \left[\text{using } (D_{T_i}, \bar{y}_{T_i}) \right]$ → minimize loss function on the training set

prediction → $\hat{\bar{y}}_i \leftarrow (D_{F_i}, \bar{w}_i)$ use $\bar{w}_i$ to predict for instances in F_i

end

Evaluate $(\hat{\bar{y}}, \bar{y})$

$\hat{\bar{y}}$: vector of predicted values
$\bar{y}$: true values.

For instance, with $|C|=6$, $|\lambda|=5$ values, we get 30 $\hat{\bar{y}}$ predictions. Pick the (C_i, λ_j) pair that gives the least error.

We can use SGD inside each iteration (for solving $\min_{\bar{w}} J_i$).

Note that depending on how many hyperparameters are there and on the # folds (k), we may end up solving large numbers of (smaller) optimization problems here ($\min J_i$)! or huge!

Details of Stochastic Gradient Descent (SGD)

Recall: linear regression with Tikhonov regularization:

$$J = \frac{1}{2}\|D\bar{w}-\bar{y}\|^2 + \frac{\lambda}{2}\|\bar{w}\|^2 = \frac{1}{2}\sum_{i=1}^{n}(\bar{w}^T\bar{x}_i - y_i)^2 + \frac{\lambda}{2}\|\bar{w}\|^2$$

$$\Rightarrow \nabla J = \underbrace{D^TD\bar{w} - D^T\bar{y} + \lambda\bar{w}}$$

GD: $\bar{w} \leftarrow \bar{w} - \alpha\nabla J$

$$\Rightarrow \bar{w} \leftarrow \bar{w}(1-\alpha\lambda) - \alpha D^T(\underbrace{D\bar{w}-\bar{y}}_{\bar{e}\text{ error}})$$

We compute the error vector $\bar{e}$ first, and then $D^T\bar{e}$ (and hence avoid computing D^TD). → computationally expensive to evaluate matrix-matrix product; matrix-vector products are more efficient

Rewriting the GD step:

$$\bar{w} \leftarrow \bar{w}(1-\alpha\lambda) - \alpha\sum_{i=1}^{n}\bar{x}_i\underbrace{(\bar{w}^T\bar{x}_i - y_i)}_{e_i}$$

We get the SGD update by not using all n points:

$$\bar{w} \leftarrow \bar{w}(1-\alpha\lambda) - \alpha\sum_{i\in S}\bar{x}_i\underbrace{(\bar{w}^T\bar{x}_i - y_i)}_{e_i}$$

If setting $y = \bar{w}^T\bar{x}+b$ (or $\bar{w}^T\bar{x}+w_0$), the SGD updates are made in a similar fashion: → b: "bias"

$$\bar{w} \leftarrow \bar{w}(1-\alpha\lambda) - \alpha\sum_{i\in S}\bar{x}_i(\bar{w}^T\bar{x}_i + b - y_i)$$

$$b \leftarrow b(1-\alpha\lambda) - \alpha\sum_{i\in S}(\bar{w}^T\bar{x}_i + b - y_i)$$

SGD for binary classification

We have $y_i = \pm 1$ now.

Recall: $J = \frac{1}{2}\sum_{i=1}^{n} y_i^2 (\bar{x}_i^T \bar{w} - y_i)^2 + \frac{\lambda}{2}\|\bar{w}\|^2$

$y_i^2 = 1$

$$= \frac{1}{2}\sum_{i=1}^{n} (y_i^2 - y_i(\bar{w}^T \bar{x}_i))^2 + \frac{\lambda}{2}\|\bar{w}\|^2$$

$$J = \frac{1}{2}\sum_{i=1}^{n} (1 - y_i(\bar{w}^T \bar{x}_i))^2 + \frac{\lambda}{2}\|\bar{w}\|^2$$

SGD: $\bar{w} \leftarrow \bar{w} - \alpha \nabla J$

$$\bar{w} \leftarrow \bar{w}(1 - \alpha\lambda) + \alpha \sum_{\cancel{i=1}\ i \in S}^{n} y_i \bar{x}_i \underbrace{(1 - y_i(\bar{w}^T \bar{x}_i))}_{\text{error}}$$

Recall that this loss function penalizes truly well-separated instances.

Better loss functions: hinge loss and L_2-SVM loss:

$$J_{H\text{-SVM}} = \sum_{i=1}^{n} \max\{0, 1 - y_i(\bar{w}^T \bar{x}_i)\} + \frac{\lambda}{2}\|\bar{w}\|^2 \quad \text{(hinge)}$$

$$J_{L_2\text{-SVM}} = \frac{1}{2}\sum_{i=1}^{n} \max\{0, (1 - y_i(\bar{w}^T \bar{x}_i))\}^2 + \frac{\lambda}{2}\|\bar{w}\|^2 \quad (L_2\text{-SVM})$$

Here are the gradients of these loss functions:

$$\nabla J_{H\text{-SVM}} = -y_i \bar{x}_i \, \delta([1 - y_i(\bar{w}^T \bar{x}_i)] > 0) + \lambda \bar{w}$$

(indicator)

$$\nabla J_{L_2\text{-SVM}} = -y_i \bar{x}_i \max\{0, 1 - y_i(\bar{w}^T \bar{x}_i)\} + \lambda \bar{w}$$