CSE517A Machine Learning	Spring 2020
Lecture 1: Structural Risk Minimization	
Instructor: Marion Neumann	*Scribe: Jingyu Xin*
Reading: FCML Ch1 (Linear Modeling); ESL 3.4.3, 10.6	

Learning Objective

Understand that many machine learning algorithms solve the *structural risk minimization problem*, which is essentially minimizing a combination of *loss function* and *model complexity penalty.*

Application

Build a **spam filter** that works well on the *training data* and generalizes well to unseen *test data.*

In fact, this will be our first implementation project for the course. Take some time to answer the following warm-up questions:
(1) *How does our data look like?* (2) *What are the features?* (3) *What is the prediction task?* (4) *How well do you think a linear classifier will perform?* (5) *How do you measure the performance of a (linear) classifier?*

1 Introduction

1.1 Machine Learning Problem

Assume we have a dataset

$$D = \{(\mathbf{x}_i, y_i)\}_{i=1,\ldots,n}, \tag{1}$$

our goal is to learn

$$y = h(\mathbf{x}) \tag{2}$$

such that:

$$\begin{aligned} &h(\mathbf{x}_i) = y_i, \forall i = 1, \ldots, n \text{ and} \\ &h(\mathbf{x}^*) = y^* \text{ for unseen test data} \end{aligned}$$

Note that we can write $y = h(f(\mathbf{x}))$ with $f : \mathbf{x} \to \mathbb{R}$ and we get classification models with class labels $\{-1, +1\}$ by choosing the sign function $h(a) = sign(a)$ and regression models by using the identity function $h(a) = I(a)$.

Question: How do we find such a function h?
Answer: Optimize some performance measure of the model (aka the function/hypothesis h).

$\Rightarrow$ minimize expected risk:

$$\min_h R[h] = \min_h \int \underbrace{l(h(\mathbf{x}), \mathbf{y})}_{e.g.\text{squared loss}} \, \mathrm{d}p(\mathbf{x}, y), \tag{3}$$

where $p(\mathbf{x}, y)$ is the joint probability of the data, which is unknown.

$\Rightarrow$ use empirical risk instead:

$$\min_h R_{emp}[h] = \min_h \frac{1}{n} \sum_{i=1}^{n} l(h(\mathbf{x}_i), y_i) \tag{4}$$

Empirical risk minimization minimizes the training error. However, this tends to *overfit* to the training data, and typically a simpler model is preferred (*Occam's razor*).

1.2 Structural Risk Minimization

The goal of structural risk minimization is to balance fitting the training data against model complexity. Training means then to learn the model parameters by solving the following optimization problem:

$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = \min_{\mathbf{w}} \underbrace{\frac{1}{n} \sum_{i=1}^{n} l(h_{\mathbf{w}}(\mathbf{x}_i), y_i)}_{\text{training loss}} + \lambda \underbrace{r(\mathbf{w})}_{\text{regularizer}} \tag{5}$$

where the objective function $\mathcal{L}(\mathbf{w})$ combines a loss function penalizing a high training error and a regularizer penalizing the model complexity. λ is a model *hyperparameter* that controls the trade-off between the terms. We are interested in choosing λ to minimize the *true risk* (test error). As the true risk is unknown we may resort to *cross-validation* to learn λ.[1] The science behind finding an ideal loss function is known as Empirical Risk Minimization (ERM). Extending the objective function to incorporate a regularizer leads to Structural Risk Minimization (SRM).

This provides us with a unified view on many machine learning methods. By plugging in different (surrogate) loss functions and regularizers, we obtain different machine learning models. Remember for example the **unconstrained SVM formulation**:

$$\min_{\mathbf{w}} \; C \underbrace{\sum_{i=1}^{n} \max[1 - y_i (\underbrace{\mathbf{w}^T \mathbf{x}_i + b}_{=f_{\mathbf{w}}(\mathbf{x}_i)}), 0]}_{\text{hinge-loss}} + \underbrace{||\mathbf{w}||_2^2}_{l_2-\text{regularizer}} \tag{6}$$

SVM uses the hinge loss as error measure and the l_2-regularizer to penalize complex solutions; $\lambda = \frac{1}{C}$.

2 Loss Functions

2.1 Commonly Used Binary Classification Loss Functions

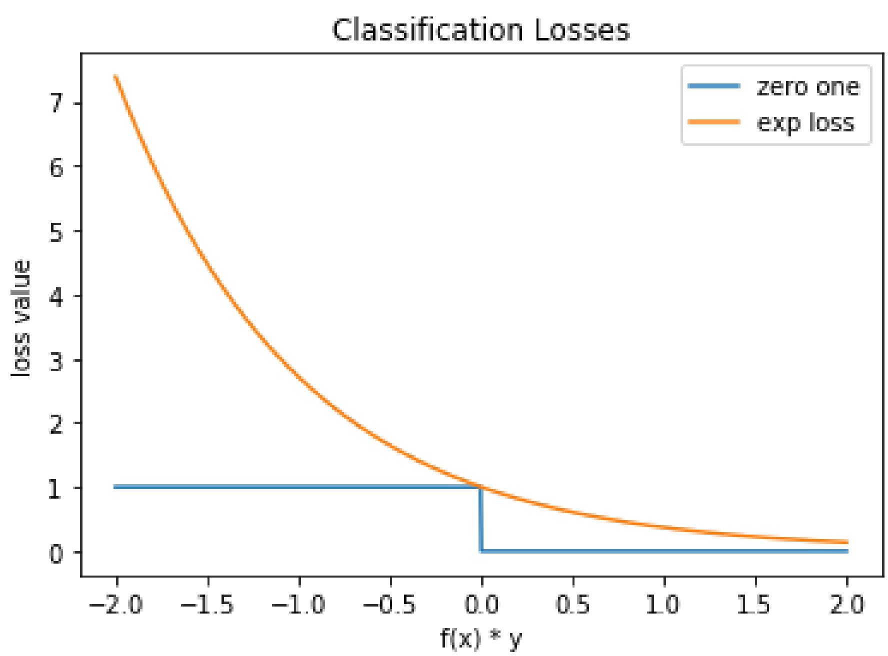
For binary classification we naturally want to minimize the zero-one loss $l(h_{\mathbf{w}}(\mathbf{x}), y)$ also called true classification error. However, due to its non-continuity it is impractical to optimize and we resort to using various *surrogate* loss functions $l(f_{\mathbf{w}}(\mathbf{x}), y)$. Table 1 summarizes the most commonly used classification losses.

[1] In Bayesian machine learning, we truly incorporate the choice of λ into the learning objective.

Table 1: loss functions for classification $y \in \{-1, +1\}$

Loss $l(f_{\mathbf{w}}(\mathbf{x}_i), y_i)$	**Usage**	**Comments**
Zero-one Loss: $\delta(h_{\mathbf{w}}(\mathbf{x}_i) \neq y_i)$	true classification loss	Non-continuous and thus impractical to optimize.
Hinge-Loss: $\max[1 - f_{\mathbf{w}}(\mathbf{x}_i) * y_i, 0]^p$	• standard SVM ($p = 1$) • (differentiable) squared hinge loss SVM ($p = 2$)	When used for standard SVM, the loss function denotes margin length between linear separator and its closest point in either class. Only differentiable everywhere with $p = 2$.
Log-Loss: $\log(1 + e^{-f_{\mathbf{w}}(\mathbf{x}_i) y_i})$	logistic regression	One of the most popular loss functions in machine learning, since its outputs are very well-tuned.
Exponential Loss: $e^{-f_{\mathbf{w}}(\mathbf{x}_i) y_i}$	AdaBoost	This function is very aggressive and thus sensitive to label noise. The loss of a misprediction increases exponentially with the value of $-f_{\mathbf{w}}(\mathbf{x}_i) y_i$.

What do all these loss functions look like? The Illustration below shows the zero-one and exponential losses, where the input/x-axis is the "correctness" of the prediction $f_{\mathbf{w}}(\mathbf{x}_i) y_i$.



Exercise 2.1. Add the *hinge loss* and *log loss functions* to this plot.

Additional Notes on Classification Loss Functions

1. Zero-one loss is zero when the prediction is correct, and one when incorrect.
2. As $z \to \infty$, log-loss, exp-loss, and hinge loss become increasingly parallel.
3. The exponential loss and the hinge loss are both upper bounds of the zero-one loss. (For the exponential loss, this is an important aspect in Adaboost.)

Exercise 2.2. In what scenario would you want to use the Huber loss instead of squared or absolute loss?

2.2 Commonly Used Regression Loss Functions

Unsurprisingly, regression models (where the predictions are reals) also have their own loss functions summarized in Table **??**. Note that for regression the error is quantified as $h_{\mathbf{w}}(\mathbf{x}) = f_{\mathbf{w}}(\mathbf{x})$.

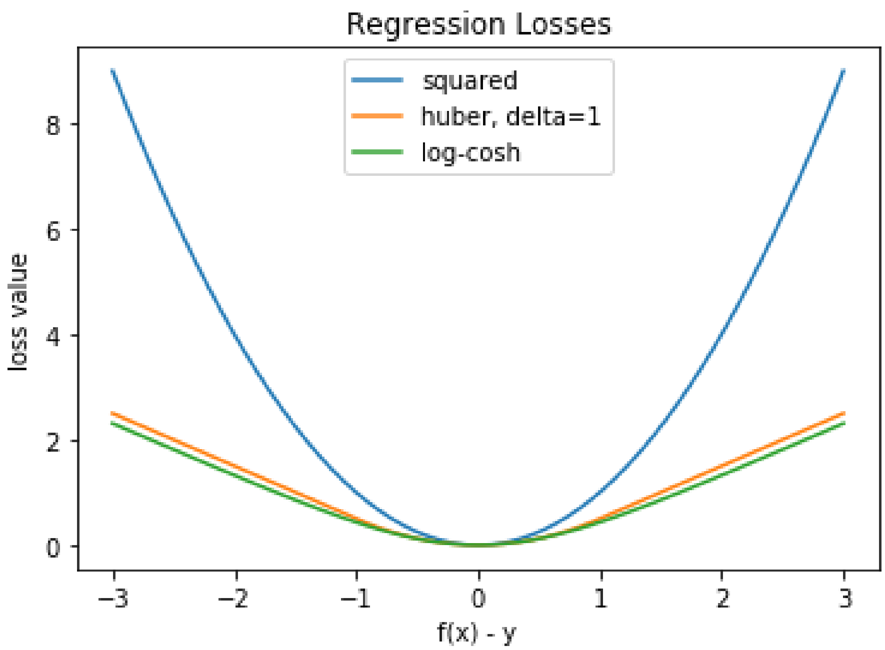
Table 2: Loss Functions With Regression, $y \in \mathbb{R}$

Loss $l(f_{\mathbf{w}}(\mathbf{x}_i), y_i)$	**Comments**
Squared Loss: $$(f_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2$$	• most popular regression loss function • $\mathbf{w}^*$ will be related to the **mean** observations in D[2] • ADVANTAGE: differentiable everywhere • DISADVANTAGE: tries to accommodate every sample $\to$ sensitive to outliers/noise • also known as Ordinary Least Squares (OLS)
Absolute Loss: $$\lvert f_{\mathbf{w}}(\mathbf{x}_i) - y_i \rvert$$	• also a very popular loss function • $\mathbf{w}^*$will be related to the **median** observations in D[3] • ADVANTAGE: less sensitive to noise • DISADVANTAGE: not differentiable at 0
Huber Loss: $$\begin{cases} \frac{1}{2} z_i^2 & \text{if } \lvert z_i \rvert < \delta \\ \delta(\lvert z_i \rvert - \frac{\delta}{2}) & \text{otherwise} \end{cases}$$ where $z_i = f_{\mathbf{w}}(\mathbf{x}_i) - y_i$	• also known as Smooth Absolute Loss • Once-differentiable • ADVANTAGE: "Best of Both Worlds" of squared and absolute loss • Takes on behavior of squared loss when loss is small, and absolute loss when loss is large.

[2] For the same input location and multiple observations, the minimal squared loss is achieved for the mean of the observations.
[3] For the same input location and multiple observations, the minimal absolute loss is achieved for the median of the observations.

Log-Cosh Loss: $$\log(\cosh(f_{\mathbf{w}}(\mathbf{x}_i) - y_i)),$$ where $\cosh(x) = \frac{e^x+e^{-x}}{2}$	• ADVANTAGE: similar to Huber loss, but twice differentiable everywhere.
ε-Insensitive Loss $$\begin{cases} 0 & \text{if } \lvert z_i \rvert < \varepsilon \\ \lvert z_i \rvert - \varepsilon & \text{otherwise} \end{cases}$$ where $z_i = f_{\mathbf{w}}(\mathbf{x}_i) - y_i$	• used in SVM regression • yields sparse solution (cf. *support vectors*) • ε regulates the sensitivity of the loss and hence the number of support vectors used in the SVM

What do all these loss functions look like? The Illustration below shows the squared loss, huber loss with $\delta = 1$, and log-cosh loss, where the input/x-axis is the "correctness" of the prediction $z = f_{\mathbf{w}}(\mathbf{x}_i) - y_i$.



Exercise 2.3. Add the functions for the *absolute loss* and the *huber loss* with $\delta = 2$ to this plot.

Exercise 2.4. In the same or a new plot, graph the functions for the ε-*insensitive loss* for $\varepsilon = 1$ and $\varepsilon = 0.1$.

3 Regularizers

Similar to the SVM primal derivation, we can establish the following equivalency:

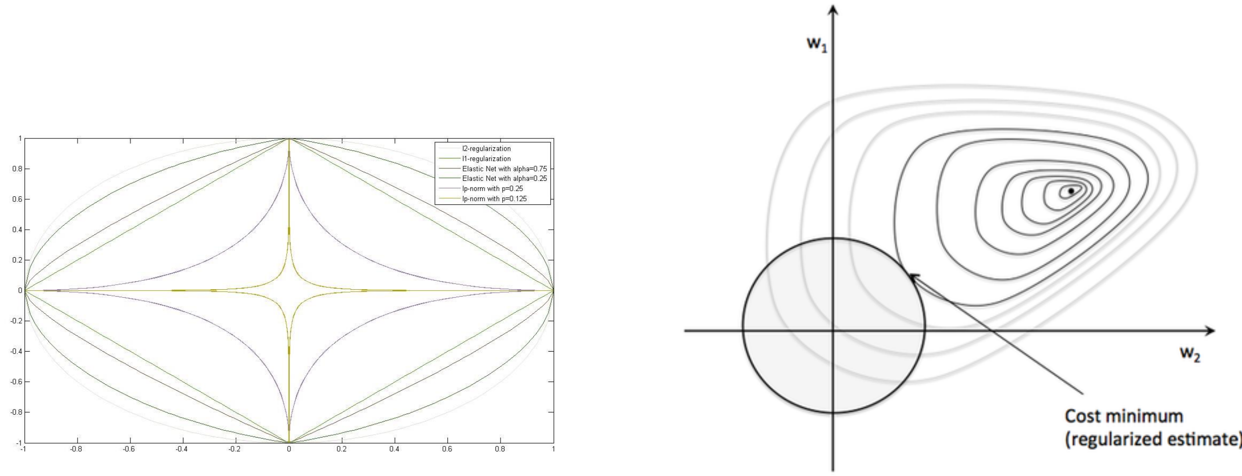
$$\min_{\mathbf{w},b} \sum_{i=1}^{n} l(\mathbf{w}^T\mathbf{x}_i + b, y_i) + \lambda r(\mathbf{w}) \quad \Longleftrightarrow \quad \begin{array}{c} \min_{\mathbf{w},b} \sum_{i=1}^{n} l(\mathbf{w}^T\mathbf{x}_i + b, y_i) \\ \text{subject to: } r(\mathbf{w}) \leq B \end{array} \tag{7}$$

For each $\lambda \geq 0$, there exists $B \geq 0$ such that the two formulations in Eq. (**??**) are equivalent, and vice versa. In previous sections, the l_2-regularizer has been introduced as the component in SVM that reflects the complexity of solutions. Besides the l_2-regularizer, other types of useful regularizers and their properties are listed in Table **??**.

Table 3: Types of regularizers

Regularizers $r(\mathbf{w})$	**Properties**
l_2-regularizer: $r(\mathbf{w}) = \mathbf{w}^T\mathbf{w} = (\|\|\mathbf{w}\|\|_2)^2$	• ADVANTAGE: strictly convex • ADVANTAGE: differentiable • DISADVANTAGE: uses weights on all features, i.e. relies on all features to some degree (ideally we would like to avoid this) - these are known as <u>dense solutions</u>.
l_1-regularizer: $r(\mathbf{w}) = \|\|\mathbf{w}\|\|_1$	• convex (but not strictly) • DISADVANTAGE: not differentiable at 0 (the point which minimization is intended to bring us to • Effect: <u>sparse</u> (i.e. not <u>dense</u>) solutions
Elastic Net: $\alpha\|\|\mathbf{w}\|\|_1 + (1-\alpha)\|\|\mathbf{w}\|\|_2^2, \alpha \in [0,1)$	• ADVANTAGE: strictly convex (i.e. unique solution) • DISADVANTAGE: non-differentiable
lp-Norm, often $0 < p \leq 1$: $\|\|\mathbf{w}\|\|_p = (\sum_{i=1}^{d} \|\mathbf{w}\|^p)^{\frac{1}{p}}$	• DISADVANTAGE: non-convex • ADVANTAGE: very sparse solutions • initialization dependent • DISADVANTAGE: not differentiable

Figure **??** shows plots of some common used regularizers. Note that regularizers are functions of $\mathbf{w}$ and not $\mathbf{x}_i$. Figure **??** shows the effect that adding a regularizer to the loss function minimization has on the optimization problem and its solution.



(a) Common regularizers.

(b) Contours of a loss function, constraint region for the l_2-regularizer, and optimal solution.

Figure 1: Illustrations for a two dimensional feature space ($d = 2$).

Exercise 3.1. Regularizers

(a) Add the constraint region for the l_1 and elastic net regularizers to the plot in Figure **??**.

(b) The l_p-regularizer yields sparse solutions for $\mathbf{w}$. Explain why.

(c) What is the advantage of using the elastic net regularizer compared to l_1 regularization? What is it's advantage compared to l_2 regularization? Briefly explain one advantage each.

4 Famous SRM models

This section includes several special cases of SRM – all preforming structural risk minimization, such as Ordinary Least Squares, Ridge Regression, Lasso, and Logistic Regression. Table 5 provides information on their loss functions, regularizers, as well as solutions.

Notation: Here we use $X \in \mathbb{R}^{d \times n}$, where d is the number of feature dimensions and n is the number of training data points[4]. **Caution:** this is the **transpose** of what they use in FCML.

Table 4: Special Cases of SRM

Loss and Regularizer	**Properties**	**Solution**
Ordinary Least Squares: $$\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} (\mathbf{w}^T \mathbf{x}_i - y_i)^2$$	• squared loss • no regularization	• $\mathbf{w} = (XX^T)^{-1} X y^T$ • $X = [\mathbf{x_1}, ..., \mathbf{x_n}]$ • $\mathbf{y} = [y_1, ..., y_n]$

[4] I personally prefer this notation since you can quickly determine whether we are using *inner* or *outer* products. E.g. the inner product in matrix notation will then look like $X^T X$, which is more intuitive as it aligns with the inner product of vectors $\mathbf{x}^T \mathbf{x}$. However, both notations are found in the literature. So, always be careful about the definition of X.

Ridge Regression: $\min_{\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n}(\mathbf{w}^T\mathbf{x}_i - y_i)^2 + \lambda\|\|\mathbf{w}\|\|_2^2$	• squared loss • l_2-regularization	• $\mathbf{w} = (XX^T + \lambda I)^{-1}Xy^T$
Lasso: $\min_{\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n}(\mathbf{w}^T\mathbf{x}_i - y_i)^2 + \lambda\|\|\mathbf{w}\|\|_1$	• + sparsity inducing (good for feature selection) • + convex • - not strictly convex (no unique solution) • - not differentiable (at 0)	• Solve with (sub)-gradient descent or LARS (least angle regression)
Logistic Regression: $\min_{\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n}\log(1 + e^{-y_i(\mathbf{w}^T\mathbf{x}_i + b)})$	• often also l_1 or l_2 regularized	• Solve with gradient descent or Newton's method • $P(y = +1 \mid \mathbf{x}) = \frac{1}{1+e^{-y(\mathbf{w}^T\mathbf{x}+b)}}$
SVM: cf. Equation (6)	• hinge-loss • l_2-regularization	• solve dual quatradic program (QP)

5 Summary

SRM tries to find the best model by explicitly adding a weighted complexity penalty (regularization term) to the loss function optimization. **List of concepts and terms to understand from this lecture**:

- ERM
- SRM
- *overfitting*
- *underfitting*
- *Occam's razor*
- *training loss*, *testing loss*
- *surrogate loss*
- *regularizer*
- *cross-validation*

Exercise 5.1. Using *your own words*, summarize each of the above concepts in 1-2 sentences.[a]

[a] *Yes, **say it out loud** or **write it down**, it'll help you retain the knowledge!*

With respect to our **spam filter application**, we are now able to come up with objective functions, aka learning models, that will (hopefully) work well on the training data and are able to generalize well to unseen test data.

The next step is to actually build the spam filter, which means we will need to *solve* the SRM problem (Eq. **??**) for various combinations of loss functions and regularizers. In the next lecture, we will cover a variety of optimization techniques to help us with that.