MSc in Computer Science

at University of Milan

Statistical Methods for Machine Learning

course held by **Nicoló Cesa-Bianchi**

Email:
federico.bruzzone@studenti.unimi.it
andrea.longoni3@studenti.unimi.it
luca.bellani@studenti.unimi.it

Created by:
Federico Bruzzone
Andrea Longoni
Luca Bellani

Academic year of 2022/2023

Contents

1 Introduction **4**

2 The Nearest Neighbour algorithm **6**

3 Tree predictors **7**

4 Statistical Learning **8**

5 Risk Analysis for Tree Predictors **11**

6 Hyperparameter Tuning and Risk Estimates **12**

7 Consistency and Nonparametric Algorithms **13**

8 Risk analysis for nearest neighbor **15**

9 Linear Predictor **16**

10 Online Gradient Descent **18**

11 Kernel Functions **20**

12 Support Vector Machine **22**

13 Stability and risk control for SVM **24**

1 Introduction

- Write the formula for the square loss, the zero-one loss, and the logarithmic loss.

  Absolute loss: $\ell(y, \hat{y}) = |y - \hat{y}|$
  Square loss: $\ell(y, \hat{y}) = (y - \hat{y})^2$
  Zero-one loss: $\ell(y, \hat{y}) = \begin{cases} 0 & \text{if } y = \hat{y} \\ 1 & \text{if } y \neq \hat{y} \end{cases}$
  Logarithmic loss: $\ell(y, \hat{y}) = \begin{cases} \ln \frac{1}{\hat{y}} & \text{if } y = 1 \\ \ln \frac{1}{1-\hat{y}} & \text{if } y = 0 \end{cases}$

- What does a learning algorithm receive in input? And what does it produce in output?

  A **learning algorithm** receives a training set as an input and output a predictor.
  A **training set** is a set of examples $\mathcal{S} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ where $\boldsymbol{x}_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$ for $i = 1, \ldots, n$. Training and test set are often prepared together, through a single round of data collection and annotation.
  A **predictor** is a function $f : \mathcal{X} \to \mathcal{Y}$ mapping data points to labels.

- Write the mathematical formula defining the training error of a predictor h.

$$\ell_S(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, h(\boldsymbol{x}_i))$$

- Write the mathematical formula defining the ERM algorithm over a class $\mathcal{H}$ of predictors. Define the main quantities occurring in the formula.

  Let $\mathcal{F}$ be a given set of predictors and ℓ a loss function. The **ERM** (Empirical Risk Minimization) is a learning algorithm that outputs a predictor $\hat{f}$ that minimizes the training error.

$$\hat{f} \in \underset{f \in \mathcal{F}}{argmin} \left(\ell_S(f)\right)$$

  ERM obviously fails when no predictor in $\mathcal{F}$ has a low test error. This suggests the we should run ERM with a large $\mathcal{F}$, so that there is a good chance that a predictor with low test error exists in $\mathcal{F}$.

  In order not to fail ERM, the training set should contain at least $\log_2 |\mathcal{F}|$ distinct data points. Equivalently, $|\mathcal{F}|$ should be smaller than 2^m, where m is the training set size.

- Explain in words how overfitting and underfitting are defined in terms of behavior of an algorithm on training and test set.

  We may give specific names to the two ways of failing of ERM (i.e., returning a predictor with high test error) for a generic learning algorithm $\mathcal{A}$:

  - if $\mathcal{A}$ fails by returning predictors with high training error, then we say that $\mathcal{A}$ is **underfitting**,
  - if $\mathcal{A}$ fails by returning predictors with low training error, then we say that $\mathcal{A}$ is **overfitting**.

- Name and describe three reasons why labels may be noisy[1].

  Namely, when labels y are not deterministically associated with data points $\boldsymbol{x}$. Noise may occur for at least three (not mutually exclusive) reasons.

  1. **Human in the loop**: The labels are assigned by a human annotator who decides the "true" label for each data point. In this case, different annotators may have different opinions.
  2. **Epistemic uncertainty**: Each data point is represented by a feature vector $\boldsymbol{x}$ that does not contain enough information to uniquely determine the label.
  3. **Aleatoric uncertainty**: The feature vector $\boldsymbol{x}$ representing a data point is obtained through noisy measurements. The label associated with a given $\boldsymbol{x}$ is then stochastic because the same $\boldsymbol{x}$ could have been generated by different data points.

  Noisy labels cause overfitting because they may mislead the algorithm with regard to what is the "true" label for a given data point.

[1] Overfitting often arises when labels are noisy.

2 The Nearest Neighbour algorithm

- Is k-NN more likely to overfit when k is large or small?

  The learning algorithm suffers from high test error for small values of k (overfitting) and for large values of k (underfitting).

3 Tree predictors

- Write a short pseudo-code for building a tree classifier based on a training set.

  We now describe a generic method to construct a binary tree given a training set S.

  1. **Initialization**: Create T with only the root ℓ and let $S_\ell = S$. Let the label associated with the root be the most frequent label in S_ℓ.
  2. **Main loop**: Pick a leaf ℓ and replace it with an internal node v creating two children $\ell^{'}$ and $\ell^{''}$. Pick an attribute i and a test $f : \mathcal{X}_i \to \{1, 2\}$. Associate the test f with v and partition S_ℓ in the two subsets

  $$S_{\ell'} = \{(\boldsymbol{x}_t, y_t) \in S_\ell : f(x_{t,i}) = 1\} \text{ and } S_{\ell''} = \{(\boldsymbol{x}_t, y_t) \in S_\ell : f(x_{t,i}) = 2\}$$

  Let the labels associated with $S_{\ell'}$ and $S_{\ell''}$ be the most frequent labels in $S_{\ell'}$ and $S_{\ell''}$ respectively.

- What is the property of a splitting criterion ψ ensuring that the training error of a tree classifier does not increase after a split? Bonus points if you justify your answer with a proof.

  To answer this question is sufficient to observe that ψ (i.e, $\psi(x) = \min\{x, 1-x\}$) is a concave function.
  We can then apply Jensen's inequality, stating that $\psi(\alpha a + (1-\alpha)b) \geq \alpha\psi(a) + (1-\alpha)\psi(b)$ for all $a, b \in \mathbb{R}$ and $\alpha \in [0, 1]$.

  Hence, via Jensen's inequality, we can study how the training error changes when ℓ is replaces by two new leaves $\ell^{'}$ and $\ell^{''}$.

  $$\psi\left(\frac{N_\ell^+}{N_\ell}\right) N_\ell = \psi\left(\frac{N_{\ell'}^+}{N_{\ell'}}\frac{N_{\ell'}}{N_\ell} + \frac{N_{\ell''}^+}{N_{\ell''}}\frac{N_{\ell''}}{N_\ell}\right) N_\ell \geq \psi\left(\frac{N_{\ell'}^+}{N_{\ell'}}\right)\frac{N_{\ell'}}{N_\ell}N_\ell + \psi\left(\frac{N_{\ell''}^+}{N_{\ell''}}\right)\frac{N_{\ell''}}{N_\ell}N_\ell$$

  meaning that a split never increaseses the training error.

- Write the formula for at least two splitting criteria ψ used in practice to build tree classifiers.

  We use another type of splitting criterion ψ because the one described in the previous question has strictly negative second derivative. Define p, r, q where p is the parent node, r and q are the children nodes, and they are on the same side with respect to the α and $p = \alpha r + (1-\alpha)q$.
  In this case, $\psi(p) - \alpha\psi(r) + (1-\alpha)\psi(q) = 0$ beacuse ψ is a straight line.

  Some example of functions ψ used in practice are

  1. **Gini function**: $\psi(p) = 2p(1-p)$
  2. **Scaled entropy**: $\psi(p) = -\frac{p}{2}\log_2(p) - \frac{1-p}{2}\log_2(1-p)$
  3. $\psi(p) = \sqrt{p(1-p)}$

4 Statistical Learning

- Write the formula for the statistical risk of a predictor h with respect to a generic loss function and data distribution.

  The performance of a predictor $h : \mathcal{X} \to \mathcal{Y}$ with respect to $(\mathcal{D}, \ell)$ is evluated via the **statistical risk**, defined by:

  $$\ell_{\mathcal{D}}(h) = \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}[\ell(Y, h(\boldsymbol{X}))]$$

  This is the expected value of the loss function on a randon example $(\boldsymbol{X}, Y)$ drawn from $\mathcal{D}$.

- Write the formula for the Bayes optimal predictor for a generic loss function and data distribution.

  The best possible predictor $f^* : \mathcal{X} \to \mathcal{Y}$ given $\mathcal{D}$ is known as the **Bayes optimal predictor**, and is defined by:

  $$f^*(\boldsymbol{x}) = \operatorname*{argmin}_{\hat{y}\in\mathcal{Y}} \mathbb{E}[\ell(Y, \hat{y}) \mid \boldsymbol{X} = \boldsymbol{x}]$$

- Write the formula for Bayes optimal predictor and Bayes risk for the zero-one loss.

  On binary classification, where $\mathcal{Y} = \{-1, +1\}$. Let $\eta(\boldsymbol{x})$ be the probability of $Y = 1$ conditioned on $\mathbf{X} = \boldsymbol{x}$. We view $\eta(\boldsymbol{x}) = \mathbb{P}(Y = +1 \mid \mathbf{X} = \boldsymbol{x})$ as the value on $\boldsymbol{x}$ of a function $\eta : \mathcal{X} \to [0, 1]$.
  Let $\mathbb{I}\{A\} \in \{0, 1\}$ be the indicator function of the event A; that is, $\mathbb{I}\{A\} = 1$ if A occurs and $\mathbb{I}\{A\} = 0$ otherwise.
  The statistical risk with respect to the zero-one loss $\ell(y, \hat{y}) = \mathbb{I}\{y \neq \hat{y}\}$ is therefore defined by:

  $$\ell_{\mathcal{D}}(h) = \mathbb{E}[\mathbb{I}\{Y \neq h(\boldsymbol{X})\}] = \mathbb{P}(Y \neq h(\boldsymbol{X}))$$

  The Bayes optimal predictor $f^* : \mathcal{X} \to \{-1, +1\}$ for binary classification is derived as follows:

  $$\begin{aligned}
  f^*(\boldsymbol{x}) &= \operatorname*{argmin}_{\hat{y}\in\{-1,+1\}} \mathbb{E}[\ell(Y, \hat{y}) \mid \boldsymbol{X} = \boldsymbol{x}] \\
  &= \operatorname*{argmin}_{\hat{y}\in\{-1,+1\}} (\mathbb{P}(Y = +1 \mid \boldsymbol{X} = \boldsymbol{x})\mathbb{I}\{\hat{y} = -1\} + \mathbb{P}(Y = -1 \mid \boldsymbol{X} = \boldsymbol{x})\mathbb{I}\{\hat{y} = +1\}) \\
  &= \operatorname*{argmin}_{\hat{y}\in\{-1,+1\}} (\eta(\boldsymbol{x})\mathbb{I}\{\hat{y} = -1\} + (1 - \eta(\boldsymbol{x}))\mathbb{I}\{\hat{y} = +1\}) \\
  &= \begin{cases} -1 & \text{if } \eta(\boldsymbol{x}) < 1/2 \\ +1 & \text{if } \eta(\boldsymbol{x}) \geq 1/2 \end{cases}
  \end{aligned}$$

  Hence, the Bayes optimal classifier predicts the label whose probability is the highest when conditioned on the instance. Finally, it is easy to verify that the Bayes risk in this case is $\ell_{\mathcal{D}}(f^*) = \mathbb{E}[\min\{\eta(\mathbf{X}), 1 - \eta(\mathbf{X})\}]$

- Can the Bayes risk for the zero-one loss be zero? If yes, then explain how.

  In the case of the zero-one loss, where the penalty is 1 for each incorrect prediction and 0 for each correct prediction, a Bayes risk of zero implies that the error rate is zero.

  To achieve a Bayes risk of zero, two conditions must be met:

  1. **Perfectly separable data**: The classes in the data must be completely separable without any overlap. This means that there should be a decision boundary that can perfectly separate the instances of different classes. If such a decision boundary exists, it is possible to classify all instances correctly, resulting in an error rate of zero.

2. **Knowledge of the true underlying distribution**: To construct a decision rule that achieves a Bayes risk of zero, you would need to know the true distribution from which the data is generated. With this knowledge, you can design a decision rule that assigns each instance to its correct class with certainty, leading to a perfect classification and zero error rate.

- Write the formula for Bayes optimal predictor and Bayes risk for the square loss.

  We can compute the Bayes optimal predictor for the quadratic loss function $\ell(y, \hat{y}) = (y - \hat{y})^2$ when $\mathcal{Y} \equiv \mathbb{R}$

$$\begin{aligned} f^*(\boldsymbol{x}) &= \operatorname*{argmin}_{\hat{y} \in \mathbb{R}} \mathbb{E}[(Y - \hat{y})^2 \mid \boldsymbol{X} = \boldsymbol{x}] \\ &= \operatorname*{argmin}_{\hat{y} \in \mathbb{R}} \left(\mathbb{E}[Y^2 \mid \boldsymbol{X} = \boldsymbol{x}] + \hat{y}^2 - 2\hat{y}\mathbb{E}[Y \mid \boldsymbol{X} = \boldsymbol{x}] \right) \\ &= \mathbb{E}[Y \mid \boldsymbol{X} = \boldsymbol{x}] \end{aligned}$$

  The Bayes optimal prediction for the quadratic loss function is the expected value of the label conditioned on the instance.
  Substituting in the conditional risk formula $\ell_{\mathcal{D}}(h) = \mathbb{E}[(Y - f^*(\boldsymbol{X}))^2 \mid \boldsymbol{X} = \boldsymbol{x}]$ the Bayes optimal predictor $f^*(\boldsymbol{x}) = \mathbb{E}[Y \mid \boldsymbol{X} = \boldsymbol{x}]$ we obtain:

$$\mathbb{E}[(Y - f^*(\boldsymbol{X}))^2 \mid \boldsymbol{X} = \boldsymbol{x}] = \mathbb{E}[(Y - \mathbb{E}[Y \mid \boldsymbol{x}])^2 \mid \boldsymbol{X} = \boldsymbol{x}] = \mathrm{Var}[Y \mid \boldsymbol{X} = \boldsymbol{x}]$$

- Explain in mathematical terms the relationship between test error and statistical risk.

  It should be clear that, given an arbitrary predictor h, we cannot directly compute its risk $\ell_{\mathcal{D}}(h)$ with respect to D because D is typically unknown. We thus consider the problem of estimating the risk of a given predictor h. In order to compute this estimate, we can use the **test set** $S' = \{(x'_1, y'_1), \ldots, (x'_n, y'_n)\}$. We can then estimate $\ell_{\mathcal{D}}(h)$ with the **test error**, which is the average loss of h on the test set,

$$\ell_{S'}(h) = \frac{1}{n} \sum_{t=1}^{n} \ell(y'_t, h(\boldsymbol{x}'_t))$$

  Under the assumption that the test set is generated through independent draws from D, the test error corresponds to the **sample mean** of the risk. Indeed, for each $t = 1, \ldots, n$ the example (X'_t, Y'_t) is an independent draw from D. Therefore,

$$\mathbb{E}[\ell(Y'_t, h(\boldsymbol{X}'_t))] = \ell_{\mathcal{D}}(h) \quad \text{for all } t = 1, \ldots, n$$

  Note that the above equalities rely on the assumption that h does not depend on the test set. If it did, then the above equalities would not be necessarily true. This fact is important in the analysis of learning algorithms.

- State the Chernoff-Hoeffding bounds.

  Let $Z_1, \ldots, Z_n$ be indipendent and identically distributed random variables with expectation μ and such that $0 \leq Z_i \leq 1$ for all $i = 1, \ldots, n$. Then, for any $\varepsilon > 0$,

$$\mathbb{P}\left(\frac{1}{n} \sum_{t=1}^{n} Z_t > \mu + \varepsilon \right) \leq e^{-2n\varepsilon^2} \quad \textit{and} \quad \mathbb{P}\left(\frac{1}{n} \sum_{t=1}^{n} Z_t < \mu - \varepsilon \right) \leq e^{-2n\varepsilon^2}$$

  Using the Chernoff-Hoeffding bound with $Z_t = \ell(y_t, h(\boldsymbol{x}_t)) \in [0, 1]$ we can compute a confidence

interval for the risk as follows (where the test error is written as ℓ instead of $\ell_{S'}$):

$$\begin{aligned}\mathbb{P}\left(|\ell_{\mathcal{D}}(h) - \ell(h)| > \varepsilon\right) &= \mathbb{P}\left(\ell_{\mathcal{D}}(h) - \ell(h) > \varepsilon \ \cup \ \ell(h) - \ell_{\mathcal{D}}(h) > \varepsilon\right) \\ &= \mathbb{P}\left(\ell_{\mathcal{D}}(h) - \ell(h) > \varepsilon\right) + \mathbb{P}\left(\ell(h) - \ell_{\mathcal{D}}(h) > \varepsilon\right) \\ &\leq 2e^{-2n\varepsilon^2}\end{aligned}$$

- Write the bias-variance decomposition for a generic learning algorithm A and associate the resulting components to overfitting and underfitting.

  Fix a training set S and let $h_S = A(S)$. The following is called the **bias-variance decomposition**:

  $$\begin{aligned}\ell_D(h_S) = \ell_D(h_S) - \ell_D(h^*) \quad & \text{estimation/variance error (large when overfitting)} \\ + \ell_D(h^*) - \ell_D(f^*) \quad & \text{approximation/bias error (large when underfitting)} \\ + \ell_D(f^*) \quad & \text{Bayes error (unavoidable)}\end{aligned}$$

  where f^* is the Bayes optimal predictor for $(\mathcal{D}, \ell)$.

- Write the upper bound on the estimation error of ERM run on a finite class $\mathcal{H}$ of predictors.

  We now study the case $|\mathcal{H}| < \infty$, that is when the model space contains a finite number of predictors. Note that the event $\exists h \in \mathcal{H} : |\ell_{\mathcal{D}}(h) - \ell_S(h)| > \varepsilon/2$ is the union over $h \in \mathcal{H}$ of the events $|\ell_{\mathcal{D}}(h) - \ell_S(h)| > \varepsilon/2$. Therefore, by the union bound, we have that its probability is bounded by $|\mathcal{H}|$ times the probability of the event $|\ell_{\mathcal{D}}(h) - \ell_S(h)| > \varepsilon/2$ for a single predictor $h \in \mathcal{H}$ is $\leq |\mathcal{H}| 2e^{-m\varepsilon^2/2}$.

  In conclusion, we have that

  $$\mathbb{P}(\ell_D(h_S) - \ell_D(h^*) > \varepsilon) \leq 2|\mathcal{H}|e^{-m\varepsilon^2/2}$$

  Setting the right-hand side of equal to δ and solving for ε we obtain that:

  $$\ell_D(h_S) \leq \ell_D(h^*) + \sqrt{\frac{2}{m} \ln \frac{2|\mathcal{H}|}{\delta}}$$

5 Risk Analysis for Tree Predictors

- Write the upper bound on the estimation error of ERM run on a the class of complete binary tree predictors with at most N nodes on d binary features.

  Knowing that $|\mathcal{H}_N| \leq (2de)^N$ obtained via $\frac{N-1}{2}$-th Catalan number, we can write the upper bound on the estimation error of ERM run on a the class of complete binary tree predictors with at most N nodes on d binary features as follows:

  $$\ell_{\mathcal{D}}(\hat{h}) \leq \ell_{\mathcal{D}}(h^*) + \sqrt{\frac{2}{m}\left(N(1+\ln(2d)) + \ln\frac{2}{\delta}\right)}$$

  From that, we deduce that in this case a training set of size of order $N \ln d$ is enough to control the risk of $\hat{h} \in \mathcal{H}_N$.

- Write the bound on the difference between risk and training error for an arbitrary complete binary tree classifier h on d binary features in terms of its number N_h of nodes. Bonus points if you provide a short explanation on how this bound is obtained.

  $$\ell_D(h) \leq \ell_S(h) + \sqrt{\frac{1}{2m}\left(|\sigma(h)| + \ln\frac{2}{\delta}\right)}$$

  Hence, with Occam Razor given two predictor with the same training error the simpler (the shortest $|\sigma|$) is the best.

  We upper bound the risk of a tree predictor h by its training error plus a quantity ϵ_h that now depends on the size of the tree.
  We introduce a function $w : \mathcal{H} \to [0,1]$ and call $w(h)$ the weight of tree predictor h. We assume

  $$\sum_{h\in\mathcal{H}} w(h) \leq 1$$

  And now choosing

  $$\epsilon_h = \sqrt{\frac{1}{2m}\left(\ln\frac{1}{w(h)} + \ln\frac{2}{\delta}\right)}$$

  we get that

  $$\mathbb{P}\left(\exists h \in \mathcal{H} : |\ell_{\mathcal{D}}(h) - \ell_S(h)| > \epsilon_h\right) \leq \sum_{h\in\mathcal{H}} \delta w(h) \leq \delta$$

  A consequence of this analysis is that, with probabilty at least $1-\delta$ with respect to the training set random draw, we have

  $$\ell_D(h) \leq \ell_S(h) + \sqrt{\frac{1}{2m}\left(\ln\frac{1}{w(h)} + \ln\frac{2}{\delta}\right)}$$

  Now, using a theoretic techniques we can encode each tree predictor h as a binary string $\sigma(h)$ of length $|\sigma(h)| = \mathcal{O}(N_h \log d)$. Thanks to Kraft inequality, we can associate a weight $w(h) = 2^{-|\sigma(h)|}$ to each tree predictor h in order to get the bound shown at the beginning.

6 Hyperparameter Tuning and Risk Estimates

- Write the formula for the K-fold cross validation estimate. Explain the main quantities occurring in the formula.

  The K-fold cross validation estimate of $\mathbb{E}[\ell_D(A)]$ on S, denoted by $\ell_S^{CV}(A)$, is then computed as follows: we run A on each training part S_{-i} of the folds $i = 1, \ldots, K$ and obtain the predictors $h_i = A(S_{-i}) \ldots, h_K = A(S_{-K})$. We then compute the (rescaled) error on the testing part of each fold,

  $$\ell_{S_i}(h_i) = \frac{K}{m} \sum_{(\boldsymbol{x},y)\in S_i} \ell(h_i(\boldsymbol{x}), y)$$

  Finally, we compute the CV estimate by averaging these errors:

  $$\ell_S^{CV}(A) = \frac{1}{K} \sum_{i=1}^{K} \ell_{S_i}(h_i)$$

- Write the pseudo-code for computing the nested cross validation estimate.

Algorithm 1: K-fold nested cross-validation

Input: Dataset S

Split S into folds $S_1, \ldots, S_K$

for $i = 1, \ldots, K$ **do**

  Compute training part of i-th fold: $S_{-i} \equiv S \setminus S_i$

  Run CV on S_{-i} for each $\theta \in \Theta_0$ and find $\theta_i = \underset{\theta\in\Theta_0}{\operatorname{argmin}} \ell_{S_{-i}}^{CV}(A_\theta)$

  Re-train A_{θ_i} on S_{-i} : $h_i = A_{\theta_i}(S_{-i})$

  Compute error of i-th fold: $\epsilon_i = \ell_{S_i}(h_i)$

end

Output: $(\epsilon_1 + \cdots + \epsilon_K)/K$

7 Consistency and Nonparametric Algorithms

- Write the mathematical definition of consistency for an algorithm A.

  A learning algorithm A is consistent with respect to a loss function ℓ if for any data distribution D it holds that
  $$\lim_{m\to\infty} \mathbb{E}\left[\ell_{\mathcal{D}}(A(S))\right] = \ell_{\mathcal{D}}(f^*)$$
  where the expectation is with respect to the random draw of the training set S_m of size m from the distribution $\mathcal{D}$, and $\ell_{\mathcal{D}}(f)$ is the Bayes risk risk for $(\mathcal{D}, f)$.

- Write the statement of the no-free-lunch theorem.

  For any sequence $a_1, a_2, \ldots$ of positive numbers converging to zero and such that $\frac{1}{16} \geq a_1 \geq a_2 \geq \ldots$ and for any consistent learning algorithm A for binary classification with zero-one loss, there exists a data distribution $\mathcal{D}$ such that $\ell_{\mathcal{D}}(f^*) = 0$ and $\mathbb{E}[\ell_{\mathcal{D}}(A(S_m))] \geq a_m$ for all $m \geq 1$.

- Write the mathematical definition of nonparametric learning algorithm. Define the main quantities occurring in the formula.

  Given a learning algorithm A, let $\mathcal{H}_m$ be the set of predictors generated by A on training sets of size $m : h \in \mathcal{H}_m$ if and only if there exists a training set S_m of size m such that $A(S_m) = h$. We say that A is a nonparametric learning algorithm if A's approximation error vanishes as m grows to infinity. Formally,
  $$\lim_{m\to\infty} \min_{h\in\mathcal{H}_m} \ell_{\mathcal{D}}(h^*) = \ell_{\mathcal{D}}(f^*)$$

- Name one nonparametric learning algorithm and one parametric learning algorithm.

  Nonparametric: k-nearest neighbors and greedy decision tree classifiers
  Parametric: linear regression and logistic regression

- Write the mathematical conditions on k ensuring consistency for the k-NN algorithm.

  We let k be chosen as a function k_m of the training set size, then the algorithm becomes consistent provided $k_m \to \infty$ and $k_m = o(m)$.

- Write the formula for the Lipschitz condition in a binary classification problem. Define the main quantities occurring in the formula.

  In some cases, we may define consistency with respect to a restricted class of distributions $\mathcal{D}$. For example, in binary classification we may restrict to all distribution $\mathcal{D}$ such that $\eta(\boldsymbol{x}) = \mathbb{P}(Y = 1|\boldsymbol{X} = \boldsymbol{x})$ is Lipschitz function on $\mathcal{X}$. Formally, there exists $0 < c < \infty$ such that
  $$|\eta(\boldsymbol{x}) - \eta(\boldsymbol{x}')| \leq c\|\boldsymbol{x} - \boldsymbol{x}'\| \quad \text{for all } \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$$

- Write the rate at which the risk of a consistent learning algorithm for binary classification vanish as a function of the training set size m and the dimension d under Lipschitz assumptions.

  Under Lipschitz assumptions on η, the typical convergence rate to Bayes risk is $m^{\frac{-1}{d+1}}$.

- Explain the curse of dimensionality.

  The convergence rate $m^{\frac{-1}{d+1}}$ implies that to get ϵ-close to the Bayes risk, we need a training set size m of order $\epsilon^{-(d+1)}$. This exponential dependence on the number of features of the training set size is known as **curse of dimensionality** and refers to the difficulty of learning in a nonparametric setting when datapoints live in a high-dimensional space.

8 Risk analysis for nearest neighbor

- Write the bound on the risk of the 1-NN binary classifier under Lipschitz assumptions.

Recalling that the Bayes risk for the zero-one loss is $\ell_{\mathcal{D}}(f^*) = \mathbb{E}[\min\{\eta(\boldsymbol{X}), 1-\eta(\boldsymbol{X})\}]$ we have

$$\mathbb{E}[\ell_{\mathcal{D}}(h_S)] \leq 2\ell_{\mathcal{D}}(f^*) + c\,\mathbb{E}\left[||\boldsymbol{X} - \boldsymbol{X}_{\pi(S,\boldsymbol{X})}||\right]$$

We may write:

$$\begin{aligned}
\mathbb{E}\left[||\boldsymbol{X} - \boldsymbol{X}_{\pi(S,\boldsymbol{X})}||\right] &\leq \epsilon\sqrt{d} + (2\sqrt{d})\sum_{i=0}^{r} \epsilon^{-p_i m} p_i \\
&\leq \epsilon\sqrt{d} + (2\sqrt{d})\frac{r}{em} && \text{Concave } g(p) = e^{-pm}p \text{ is maximized for } p = \frac{1}{m} \\
&\leq \sqrt{d}\left(\epsilon + \frac{2}{em}\left(\frac{2}{\epsilon}\right)^d\right) && \text{for } 0 \leq \epsilon \leq 1 \\
&\leq \sqrt{d}\ 4m^{\frac{-1}{(d+1)}} && \epsilon = 2m^{\frac{-1}{(d+1)}} \text{ for curse of dimensionality}
\end{aligned}$$

Note that for $m \to \infty$, $\ell_D(f^*) \leq \mathbb{E}[\ell_{\mathcal{D}}(h_S)] \leq 2\ell_D(f^*)$.

9 Linear Predictor

- Can the ERM over linear classifiers be computed efficiently? Can it be approximated efficiently? Motivate your answers.

$$h_S = \operatorname*{argmin}_{h \in \mathcal{H}} \frac{1}{m} \sum_{t=1}^{m} \mathbb{I}\{y_t \boldsymbol{w}^\top \boldsymbol{x}_t \leq 0\} \quad (1)$$

Unfortunately, it is unlikely to find an efficient implementation of ERM for linear classifiers with zero-one loss because is NPO. In fact, the decision problem associated with finding h_S is NP-complete even when $x_t \in \{0,1\}^d$ for $t = 1, \ldots, m$.

*If $P \neq NP$, then for all $C > 0$ there are no polynomial time algorithms approximately solve every instance S of **Min Disagreement Optimization** with a number of missclassified examples bounded by $C \times OPT(S)$.*

This implies that, unless $P = NP$, there are no efficient algorithms that approximate the solution of (1) to within any constant factor. Here efficient means with running time polynomial in the input size md.
The ERM problem becomes easier when the training set is **linearly separable**.

- Write the system of linear inequalities stating the condition of linear separability for a training set in binary classfication.

The ERM problem becomes easier to solve when the training set is **linearly separable**. A training set $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)$ is linearly separable where there exists a linear classifier with zero training error. In other words, there exists a separating hyperplane $\boldsymbol{u} \in \mathbb{R}^d$ such that

$$\gamma(\boldsymbol{u}) \stackrel{\text{def}}{=} \min_{t=1,\ldots,m} y_t \boldsymbol{u}^\top \boldsymbol{x}_t > 0$$

The quantity $\gamma(\boldsymbol{u})$ is known as the **margin** of $\boldsymbol{u}$ on the training set. The scaled margin $\gamma(\boldsymbol{u})/\|\boldsymbol{u}\|$ measure the distance between the separating hyperplane and the closest training example.

We can write the condition of linear separability as a system of linear inequalities, and this solution can be found in polynomial time using an algorithm for linear programming.

$$\begin{cases} \boldsymbol{u}^\top \boldsymbol{x}_t > 0 & \text{if } t = 1 \\ \ldots & \ldots \\ \boldsymbol{u}^\top \boldsymbol{x}_t > 0 & \text{if } t = m \end{cases}$$

- Write the pseudo-code for the Perceptron algorithm.

Algorithm 2: The Perceptron algorithm

Input: Training set $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)$

```
w = (0, ..., 0)
while true do
    for i = 1, ..., m do                                   // (epoch)
        if y_i w^T x_i ≤ 0 then
            w ← w + y_i x_i                                // (update)
    end
    if no update in last epoch then
        break
end
```

Output: $\boldsymbol{w}$

- Write the statement of the Perceptron convergence theorem.

  Let $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)$ *be a linearly separable training set. Then the Perceptron algorithm returns a linear classifier with zero training error in a finite number of updates*

$$\left(\min_{\boldsymbol{u}:\gamma(\boldsymbol{y})\geq 1} \|\boldsymbol{u}\|^2\right) \left(\max_{t=1,\ldots,m} \|\boldsymbol{x}_t\|^2\right)$$

  The apparently stonger margin constraint $\gamma(\boldsymbol{u}) \geq 1$ is actually achievable by any separating hyperplane $\boldsymbol{u}$. Indeed, if $\gamma(\boldsymbol{u}) > 0$, $y_t \boldsymbol{u}^\top \boldsymbol{x}_t > \gamma(\boldsymbol{u})$ is equivalent to $y_t \boldsymbol{u}^\top \boldsymbol{x}_t > 1$ for $\boldsymbol{v} = \boldsymbol{u}/\gamma(\boldsymbol{u})$. Hence, $\gamma(\boldsymbol{u}) \geq 1$ can be achived by rescaling $\boldsymbol{u}$.

  After the proof of this theorem and finding the upper and lower bound we can bound the number of updates M:

$$M \leq \|\boldsymbol{u}\| \left(\max_{t=1,\ldots,M} \|\boldsymbol{x}_t\|\right) \sqrt{M}$$

- Write the closed-form formula (i.e., not the argmin definition) for the Ridge Regression predictor. Define the main quantities occurring in the formula.

  The closed-form formula the Ridge Regression coefficients (predictor) is:

$$\boldsymbol{w} = \boldsymbol{w}_{S,\alpha} = (S^\top S + \alpha I)^{-1} S^\top \boldsymbol{y}$$

  where S is the matrix of training examples of size $m \times d$ called **design matrix**, $\boldsymbol{y}$ is the vector of labels of size $m \times 1$, α is the regularization parameter or penalty factor that shrinks the regression coefficients towards zero and $\boldsymbol{w}$ is the vector of Ridge Regression coefficients of length d.

10 Online Gradient Descent

- Write the pseudo-code for the projected online gradient descent algorithm.

Algorithm 3: Projected Online Gradient Descent

Input: $\eta > 0$, $U > 0$

$\boldsymbol{w} = 0$

for $i = 1, 2, \ldots$ **do**

  $\boldsymbol{w}'_{t+1} = \boldsymbol{w}_t - \frac{\eta}{\sqrt{t}} \nabla \ell_t(\boldsymbol{w}_t)$

  $\boldsymbol{w}_{t+1} = \underset{\boldsymbol{w}:\|\boldsymbol{w}\| \leq U}{\operatorname{argmin}} \|\boldsymbol{w} - \boldsymbol{w}'_{t+1}\|$

end

- Write the upper bound on the regret of projected online gradient descent on convex functions. Define the main quantities occurring in the bound.

$$\underbrace{\frac{1}{T}\sum_{t=1}^{T} \ell_t(\boldsymbol{w}_t)}_{\ell_T(\boldsymbol{w})} \leq \underbrace{\min_{\boldsymbol{u}:\|\boldsymbol{u}\| \leq U} \frac{1}{T}\sum_{t=1}^{T} \ell_t(\boldsymbol{u})}_{\ell_T(\boldsymbol{u})} + \underbrace{UG\sqrt{\frac{8}{T}}}_{R_T(\boldsymbol{u}) = Regret}$$

  - $\ell_T(\boldsymbol{w})$ is the average loss of the algorithm.
  - $\ell_T(\boldsymbol{u}^*)$ is the average loss of the best fixed action.
  - $UG\sqrt{\frac{8}{T}}$ is the regret of the algorithm.

  U maximum radius of the euclidean sphere.
  G is the bound on the norm of the gradient of the loss function, i.e. $\|\nabla \ell_t(\boldsymbol{w}_t)\| \leq G$ for all $t \leq T$.
  T is the number of epochs.

- Write the upper bound on the regret of online gradient descent on σ-strongly convex functions. Define the main quantities occurring in the bound.

$$\underbrace{\frac{1}{T}\sum_{t=1}^{T} \ell_t(\boldsymbol{w}_t)}_{\ell_T(\boldsymbol{w})} \leq \underbrace{\min_{\boldsymbol{u}:\|\boldsymbol{u}\| \leq U} \frac{1}{T}\sum_{t=1}^{T} \ell_t(\boldsymbol{u})}_{\ell_T(\boldsymbol{u})} + \underbrace{\frac{G^2}{2\sigma}\frac{\ln(T+1)}{T}}_{R_T(\boldsymbol{u}) = Regret}$$

  - $\ell_T(\boldsymbol{w})$ is the average loss of the algorithm.
  - $\ell_T(\boldsymbol{u}^*)$ is the average loss of the best fixed action.
  - $\frac{G^2}{2\sigma}\frac{\ln(T+1)}{T}$ is the regret of the algorithm.

  σ is the strong convexity parameter and it must be > 0.
  G is the bound on the norm of the gradient of the loss function, i.e. $\max_t \|\nabla \ell_t(\boldsymbol{w}_t)\| \leq G$.
  $\ln(T+1)$ is the upper bound of harmonic series over T.
  T is the number of epochs.

- Write the formula for the hinge loss.

$$h_t(\boldsymbol{u}) = \max\{0, 1 - y_t \boldsymbol{u}^T \boldsymbol{x}_t\}$$

  The hinge loss is a convex function, but it is not σ-strongly convex and moreover it is not strictly convex.
  The hinge loss is not differentiable, but it is sub-differentiable.

- Write the mistake bound for the Perceptron run on an arbitrary data stream for binary classification. Define the main quantities occurring in the bound.

$$M \leq \sum_{t=1}^{T} h_t(\boldsymbol{u}) + (\|\boldsymbol{u}\|X)^2 + \|\boldsymbol{u}\|X\sqrt{\sum_{t=1}^{T} h_t(\boldsymbol{u})} \quad \text{for all } \boldsymbol{u} \in \mathbb{R}^d$$

This shows a bound on the number of mistakes made by the Perceptron algorithm on any data sequence of arbitrary length T, including those sequences that are not linearly separable.

$\boldsymbol{u}$ is not necessarily a separator, because we are not making any assumption on the stream.
M is the number of mistakes made by the Perceptron algorithm.
$h_t(\boldsymbol{u})$ is the hinge loss at time t for the weight vector $\boldsymbol{u}$. When the sequence is linearly seperable, then there exist $\boldsymbol{u} \in \mathbb{R}^d$ such that $y_t \boldsymbol{u}^T \boldsymbol{x}_t \geq 1$ for all $t \leq T$, which in turn implies that $h_t(\boldsymbol{u}) = 0$. Hence, the bound reduces to the Perceptron mistake theorem, i.e. $M \leq (\|\boldsymbol{u}\|X)^2$.
Concluding, X is the maximum norm of the data points, i.e. $\|\boldsymbol{x}_t\| \leq X$ for all $t \leq T$.

11 Kernel Functions

- Write the formula for the polynomial kernel of degree n.

  The polynomial kernel $K_n(\boldsymbol{x}, \boldsymbol{x}')$ for all $n \in \mathbb{N}$ using Newton's binomial theorem, we can explicitly compute the map $\boldsymbol{\phi}_n$ such that $K_n(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}_n(\boldsymbol{x})^\top \boldsymbol{\phi}_n(\boldsymbol{x}')$,

$$\begin{aligned} K_n(\boldsymbol{x}, \boldsymbol{x}') &= (\boldsymbol{x}^\top \boldsymbol{x}' + 1)^n \\ &= \sum_{k=0}^{n} \binom{n}{k} (\boldsymbol{x}^\top \boldsymbol{x}')^k \qquad \text{Newton's binomial theorem} \end{aligned}$$

- Write the formula for the Gaussian kernel with parameter γ.

  The Gaussian kernel $K_\gamma(\boldsymbol{x}, \boldsymbol{x}')$ for all $\gamma > 0$,

$$K_\gamma(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\frac{1}{2\gamma} \|\boldsymbol{x} - \boldsymbol{x}'\|^2) \quad \gamma > 0$$

- Write the pseudo-code for the kernel Perceptron algorithm.

Algorithm 4: Kernel Perceptron

```
S ← ∅
for t = 1, 2, ... do
    Get next example (x_t, y_t)
    Compute ŷ_t = sgn(Σ_{s∈S} y_s K(x_s, x_t))
    if ŷ_t ≠ y_t then
        S ← S ∪ {t}
    end
end
```

- Write the mathematical definition of the linear space $\mathcal{H}_K$ of functions induced by a kernel K.

  // Talking about the Hilbert space and complete inner vector space?
  The linear space $\mathcal{H}_K$ of functions induced by a kernel K is defined as

$$\mathcal{H}_K \equiv \left\{ \sum_{i=1}^{N} \alpha_i K(\boldsymbol{x}_i, \cdot) : \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathcal{X}, \alpha_1, \ldots, \alpha_N \in \mathbb{R}, N \in \mathbb{N} \right\}$$

- Let f be an element of the linear space $\mathcal{H}_K$ induced by a kernel K. Write $f(\boldsymbol{x})$ in terms of K.

  An element $f \in \mathcal{H}_K$ is a function $f : \mathcal{X} \to \mathbb{R}$ such that

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x})$$

  for some $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathcal{X}, \alpha_1, \ldots, \alpha_N \in \mathbb{R}$ and $N \in \mathbb{N}$

- Write the mistake bound of the Perceptron convergence theorem when the Perceptron is run with a kernel K. Define the main quantities occurring in the bound.

  Recall the bound on the number of mistakes provided by the Perceptron convergence theorem

  $$M \leq \|\boldsymbol{u}\|^2 \left(\max_t \|\boldsymbol{x}_t\|\right)^2$$

  which holds for any $\boldsymbol{u} \in \mathbb{R}^d$ such that $y_t \boldsymbol{u}^\top \boldsymbol{x}_t > 1$ for all $t = 1, \ldots, m$.

  In a generic *reproducing kernel Hilbert space* $\mathcal{H}_K$, the linear separator $\boldsymbol{u}$ is some $g \in \mathcal{H}_K$ such that $y_t g(\boldsymbol{x}_t) > 1$ for all $t = 1, \ldots, m$. The squared norm $\|\boldsymbol{x}_t\|^2 = \boldsymbol{x}_t^\top \boldsymbol{x}_t$ becomes $\|\phi_K(\boldsymbol{x})\|_K^2 = \langle K(\boldsymbol{x}, \cdot), K(\boldsymbol{x}, \cdot)\rangle_K = K(\boldsymbol{x}, \boldsymbol{x})$. Finally, the squared norm $\|\boldsymbol{u}\|^2$ is replaced by

  $$\|g\|_K^2 = \left\|\sum_{i=1}^N \alpha_i K(\boldsymbol{x}_i, \cdot)\right\|_K^2 = \left\langle \sum_{i=1}^N \alpha_i K(\boldsymbol{x}_i, \cdot), \sum_{j=1}^N \alpha_j K(\boldsymbol{x}_j, \cdot) \right\rangle_K = \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

  .

- Write the mistake bound for the kernel Perceptron run on an arbitrary data stream for binary classification. Define the main quantities occurring in the bound.

  Recall the bound on the number of mistakes provided by the for the Perceptron run on an arbitrary data stream for binary classification:

  $$M \leq \sum_{t=1}^T h_t(\boldsymbol{u}) + (\|\boldsymbol{u}\|X)^2 + \|\boldsymbol{u}\|X\sqrt{\sum_{t=1}^T h_t(\boldsymbol{u})} \quad \text{for all } \boldsymbol{u} \in \mathbb{R}^d$$

  In a generic *reproducing kernel Hilbert space* $\mathcal{H}_K$, the hyperplane $\boldsymbol{u}$ is some $g \in \mathcal{H}_K$ such that $y_t g(\boldsymbol{x}_t) > 1$ for all $t = 1, \ldots, m$. X was the maximum norm of the examples, which becomes $\max_t \|\phi_K(\boldsymbol{x}_t)\|_K = \max_t K(\boldsymbol{x}_t, \cdot)$.

- Write the closed-form formula (i.e., not the argmin definition) of the kernel version of the Ridge Regression predictor.

  Recall the closed-form formula for the Ridge Regression predictor

  $$\boldsymbol{w} = \left(\alpha I + S^\top S\right)^{-1} S^\top \boldsymbol{y}$$

  where S is the $m \times d$ matrix of the training examples and $\boldsymbol{y}$ is the vector of the labels.

  We can represent the ridge regression predictor in a generic *reproducing kernel Hilbert space* $\mathcal{H}_K$ as by

  $$\boldsymbol{y}^\top \left(\alpha I + \boldsymbol{K}\right)^{-1} \boldsymbol{k}(\cdot)$$

  where $\boldsymbol{K}$ is the $m \times m$ matrix with entries $\boldsymbol{K}_{i,j} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and $\boldsymbol{k}(\cdot)$ is the vector $(K(\boldsymbol{x}_1, \cdot), \ldots, K(\boldsymbol{x}_m, \cdot))$ of functions $K(\boldsymbol{x}_t, \cdot) = \langle \phi_K(\boldsymbol{x}_t)\rangle$. Similarly to the non-kernel case, where the prediction on $\boldsymbol{x}$ is $\boldsymbol{w}^\top \boldsymbol{x}$, the prediction on $\phi_K(\boldsymbol{x})$ is $\langle g, \phi_K(\boldsymbol{x})\rangle_K$ which evaluates to $g(\boldsymbol{x}) = \boldsymbol{y}^\top \left(\alpha I + \boldsymbol{K}\right)^{-1} \boldsymbol{k}(\boldsymbol{x})$ where $\boldsymbol{k}(\boldsymbol{x}) = (K(\boldsymbol{x}_1, \boldsymbol{x}), \ldots, K(\boldsymbol{x}_m, \boldsymbol{x}))$.

12 Support Vector Machine

- Write the convex optimization problem with linear constraints that defines the SVM hyperplane in the linearly separable case.

  Given a linearly separable training set $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\} \in \mathbb{R}^d \times \{-1, 1\}$, SVM outputs the linear classifier corresponding to the unique solution $\boldsymbol{w}^*$ of the following convex optimization problem with linear constraints:

  $$\begin{aligned} &\min_{\boldsymbol{w} \in \mathbb{R}^d} \frac{1}{2}\|\boldsymbol{w}\|^2 \\ &\text{s.t. } y_t \boldsymbol{w}^\top \boldsymbol{x}_t \geq 1 \text{ for } t = 1, \ldots, m \end{aligned}$$

  Geometrically, $\boldsymbol{w}^*$ corresponds to the **maximum margin separating hyperplane**. For every linearly separable set $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\} \in \mathbb{R}^d \times \{-1, 1\}$, the maximum margin is defined as

  $$\gamma^* = \max_{\boldsymbol{u}: \|\boldsymbol{u}\|=1} \min_{t=1,\ldots,m} y_t \boldsymbol{u}^\top \boldsymbol{x}_t$$

  and the vector $\boldsymbol{u}^*$ achieving the maximum maring is the maximum margin separator.

- Write the unconstrained optimization problem whose solution defines the SVM hyperplane when the training set is not necessarily linearly separable.

  If we consider the case of a non-linearly separable training set, we should analyze how the SVM objective changes. Consider the following formulataion

  $$\begin{aligned} \min_{(\boldsymbol{w}, \boldsymbol{\xi}) \in \mathbb{R}^d + m} \quad & \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{1}{m}\sum_{t=1}^{m} \xi_t & \\ \text{s.t.} \quad & y_t \boldsymbol{w}^\top \boldsymbol{x}_t \geq 1 - \xi_t & t = 1, \ldots, m \\ & \xi_t \geq 0 \text{ for} & t = 1, \ldots, m \end{aligned}$$

  The components of $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_m)$ are called **slack variables** and measure how each margin constraint is violated by a potential solution $\boldsymbol{w}$. Finally, a regularization parameter $\lambda > 0$ is introduced to balance the two terms.

  We now consider the constraints involving the slack variables ξ_t. In oreder to minimize each ξ_t we can set

  $$\xi_t = \begin{cases} 1 - y_t \boldsymbol{w}^\top \boldsymbol{x}_t & \text{if } y_t \boldsymbol{w}^\top \boldsymbol{x}_t < 1 \\ 0 & \text{otherwise} \end{cases}$$

  Now, fix $\boldsymbol{w} \in \mathbb{R}^d$, we can see $\xi_t = \left[1 - y_t \boldsymbol{w}^\top \boldsymbol{x}_t\right]_+$ which is the hinge loss $h_t(\boldsymbol{w})$.

  The SVM problem can be rewritten as

  $$\min_{\boldsymbol{w} \in \mathbb{R}^d} \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{1}{m}\sum_{t=1}^{m} h_t(\boldsymbol{w})$$

  .

- Write the bound on the expected value of the SVM objective function achieved by Pegasos. Provide also a bound on the expected squared norm of the loss gradient.

$$\mathbb{E}\left[F(\bar{\boldsymbol{w}})\right] \leq F(\boldsymbol{w}^*) + \frac{\mathbb{E}\left[G^2\right]}{2\lambda T}(\ln T + 1)$$

where:

  - $F(\boldsymbol{w}) = \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{1}{m}\sum_{t=1}^{m} h_t(\boldsymbol{w})$ is the SVM objective function, where $h_t(\boldsymbol{w}) = \left[1 - y_t \boldsymbol{w}^\top \boldsymbol{x}_t\right]_+$ is the hinge loss.
  - $\bar{\boldsymbol{w}} = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{w}_t$ is the average of the iterates
  - $\boldsymbol{w}^* = \underset{\boldsymbol{w}\in\mathbb{R}^d}{\operatorname{argmin}}(F(\boldsymbol{w}))$ is the optimal solution of the SVM problem
  - $G = \max_{t=1,\ldots,T} \|\nabla \ell_{s_t}(\boldsymbol{w}_t)\|$ is the norm of the loss gradient
  - $\lambda > 0$ is the regularization parameter
  - T is the number of iterations

  We can bound G bounding the norm of the loss gradient in the following way: $\|\nabla \ell_t(\boldsymbol{w}_t)\| \leq X + \lambda\|\boldsymbol{w}_t\| \leq 2X$ where $X = \max_{t=1,\ldots,m} \|\boldsymbol{x}_t\|$ is the maximum norm of the training examples.

$$\mathbb{E}\left[F(\bar{\boldsymbol{w}})\right] \leq F(\boldsymbol{w}^*) + \frac{2X^2}{\lambda T}(\ln T + 1)$$

13 Stability and risk control for SVM

- Write the definition of ε-stability for a learning algorithm

  Assume that $S = S_m$ is a sample of m examples $\boldsymbol{Z}_t = (\boldsymbol{X}_t, Y_t)$ independently drawn from a distribution $\mathcal{D}$ on $\mathcal{X} \times \mathcal{Y}$, and that A is a learning algorithm. We use $S^{(t)}$ to denote S where the t-th example $(\boldsymbol{X}_t, Y_t)$ is replaced by $\boldsymbol{Z}'_t = (\boldsymbol{X}'_t, Y'_t)$, also drawn from $\mathcal{D}$ independently of S.

  Let $A(S)$ and $A(S^{(t)})$ denote the output of A on S and $S^{(t)}$ respectively, we say that A is ε-stable for a training set of size m if, for each $t = 1, \ldots, m$

  $$\mathbb{E}\left[\ell(h_{S^{(t)}}, \boldsymbol{Z}_t) - \ell(h_S, \boldsymbol{Z}_t)\right] \leq \varepsilon$$

- Write the value of ε for which SVM is known to be stable. The value depends on the radius X of the ball where the training datapoints live, the training set size m, and the regularization coefficient λ.

  Kwnowing that:
  Let ℓ be a loss function such that $\ell(\cdot, \boldsymbol{z})$ is convex, differentiable and Lipschitz with constant $L > 0$. Then the learning algorithm A such that

  $$A(S) = \underset{\boldsymbol{w} \in \mathbb{R}^d}{\operatorname{argmin}} \left(\ell_S(\boldsymbol{w}) + \frac{\lambda}{2} \|\boldsymbol{w}\|^2 \right)$$

  for all training sets S of size m is $\frac{(2L)^2}{\lambda m}$-stable for every $\lambda > 0$.

  The value of ε for which SVM is known to be stable is $\frac{(4L^2)}{\lambda m}$, where $L = X$.

- Write the mathematical conditions on the regularization coefficient λ ensuring consistency for the SVM algorithm wih Gaussian kernel.

  Let λ be chosen as a function λ_m on the training set of size m, then the SVM algorithm with Gaussian kernel is consistent if, for $m \to \infty$, λ_m must satisfy the following conditions: $\lambda_m = o(1)$ and $\lambda_m = \omega(m^{-1/2})$.