Machine Learning
Exercises

Christian Rossi

Academic Year 2023-2024

Abstract

The course topics are:

- Introduction: basic concepts.
- Learning theory:
  - Bias/variance tradeoff. Union and Chernoff/Hoeffding bounds.
  - VC dimension. Worst case (online) learning.
  - Practical advice on how to use learning algorithms.
- Supervised learning:
  - Supervised learning setup. LMS.
  - Logistic regression. Perceptron. Exponential family.
  - Kernel methods: Radial Basis Networks, Gaussian Processes, and Support Vector Machines.
  - Model selection and feature selection.
  - Ensemble methods: Bagging, boosting.
  - Evaluating and debugging learning algorithms.
- Reinforcement learning and control:
  - MDPs. Bellman equations.
  - Value iteration and policy iteration.
  - TD, SARSA, Q-learning.
  - Value function approximation.
  - Policy search. Reinforce. POMDPs.
  - Multi-Armed Bandit.

Contents

CHAPTER 1

Linear regression

1.1 Exercise one

Given the relationship:

$$S = f(TV, R, N)$$

where S is the amount of sales revenue, TV, R and N are the amount of money spent on advertisements on TV programs, radio and newspapers, respectively, explain what are the:

1. Response.
2. Independent variables.
3. Features.
4. Model.

Which kind of problem do you think it is trying to solve?

1.1.1 Solution

In the proposed relationship we have:

1. The response (or target or output) is the amount of sales S.
2. The independent variables (or input) are TV, R and N
3. The features (or input) are TV, R and N.
4. The model is identified by the function $f(\cdot)$.

Since the amount of sales S is a continuous and ordered variable, we are trying to solve a regression problem (supervised learning).

1.2 Exercise two

Why is linear regression important to understand? Select all that apply and justify your choice:

1. The linear model is often correct.
2. Linear regression is extensible and can be used to capture nonlinear effects.
3. Simple methods can outperform more complex ones if the data are noisy.
4. Understanding simpler methods sheds light on more complex ones.
5. A fast way of solving them is available.

1.2.1 Solution

1. False: it rarely happens that the problem we are modeling has linear characteristics.
2. True: it is true that this model is easy to interpret and can be extended to also consider nonlinear relationships among variables, e.g., using basis functions.
3. True: since we are able only to minimize the discrepancy between the considered function and the real one, and we can not reduce the variance introduced by noise, the use of linear model might be a better choice with respect to more complex ones since they are usually prone to overfitting, i.e., they try to model also the noise of the considered process.
4. True: they are easy to interpret and might give suggestions on more sophisticated techniques which can be used to tackle specific problems.
5. True/False: for some loss functions we have a closed form solution for linear model (LS method), thus we can guarantee that we are able to find the parameters minimizing the loss function effectively. That is not always true and depends also on the loss function we want to minimize.

1.3 Exercise three

Consider a linear regression with input x, target y and optimal parameter θ^*.

1. What happens if we consider as input variables x and $2x$?
2. What we expect on the uncertainty about the parameters we get by considering as input variables x and $2x$?
3. Provide a technique to solve the problem.
4. What happens if we consider as input variables x and x^2?

Motivate your answers.

1.3.1 Solution

The original formulation is:

$$y = \theta^* x$$

1. In this scenario, the formulation simplifies to:

   $$y = \theta_1 x + \theta_2 2x$$

   As the two variables are dependent, it can alternatively be expressed as:

   $$y = \underbrace{(\theta_1 + 2\theta_2)}_{\theta^*} x$$

   Thus, yielding the original formulation:

   $$y = \theta^* x$$

   Moreover, for computing the closed-form optimization, the formula employed is:

   $$\omega = \left(x^T x\right)^{-1} x t$$

   However, in this particular case, the matrix x takes the form:

   $$x = \begin{bmatrix} x_1 & 2x_1 \\ x_2 & 2x_2 \\ \vdots & \vdots \\ x_n & 2x_n \end{bmatrix}$$

   Hence, $x^T x$ becomes singular, rendering the previous formula inapplicable.

   In general, if x lacks full rank, then $x^T x$ becomes singular, and its inverse cannot be computed.

2. The parameter we get have a high variance, since we have an infinite number of couples of parameters minimizing the loss of the samples in the considered problem. Indeed, if the parameters of the two inputs are w_1 and w_2 we would have that the true relationship would be:

   $$y = \theta_1 x + \theta_2 2x = (\theta_1 + 2\theta_2) x$$

   Which can be satisfied by an infinite number of solutions.

3. In this case, the use of ridge regression is able to partially cope with the influence of using highly linearly correlated features. Another viable option is to remove the variables which are linearly dependent, for instance by checking if they have correlation equal to 1 or -1.

4. In this case we do not have a badly conditioned matrix since x and x^2 are not linearly dependent and the corresponding design matrix would not be ill-conditioned. As a result, we can find a closed form optimization.

1.4 Exercise four

After performing Ridge regression on a dataset with $\lambda = 10^{-5}$ we get one of the following one set of eigenvalues for the matrix $\left(\Phi^T\Phi + \lambda I\right)$:

1. $\Lambda = \{0.00000000178, 0.014, 12\}$
2. $\Lambda = \{0.0000178, -0.014, 991\}$
3. $\Lambda = \{0.0000178, 0.014, 991\}$
4. $\Lambda = \{0.0000178, 0.0000178, 991\}$

Explain whether these sets are plausible solutions or not.

1.4.1 Solution

Since the matrix $\left(\Phi^T\Phi + \lambda I\right)$ is definite positive and its eigenvalues should all be greater than $\lambda = 10^{-5}$, we have:

1. Not plausible: one eigenvalue is smaller than 10^{-5}.
2. Not plausible: one eigenvalue is negative.
3. Plausible: all positive and greater than 10^{-5}.
4. Plausible: all positive and greater than 10^{-5}.

CHAPTER 2

Classification

2.1 Exercise one

Classify the following variable as quantitative or qualitative:

1. Height.
2. Age.
3. Speed.
4. Color.

Provide a technique for transforming qualitative data into quantitative format without imposing additional organization on the data.

Solution

The classification is as follows:

1. Quantitative variable: heights can be ordered and typically belong to a bounded continuous set.
2. Quantitative variable: values are ordered natural numbers.
3. Quantitative variable: takes real number values.
4. Qualitative variable: since there's no inherent order among colors, one-hot encoding can be employed without imposing additional structure on the data.

For a set of all possible colors, $\mathcal{C} = \{c_1, \ldots, c_p\}$ one-hot encoding creates a binary variable $z_i \in \{0, 1\}$ for each color c_i. This variable equals one when the color is c_i. Thus, color c_i is represented by a binary vector $z_i = \begin{bmatrix} z_1 & \cdots & z_n \end{bmatrix}^T$, where $z_i = 1$ and all other $z_j = 0$ for $j \neq i$. This method introduces p new variables without further structuring the data. Notably, any two vectors $zi \neq z_j$ are equally distant under reasonable metrics like Euclidean distance.

It's important to note that assigning a quantitative variable i to each color would introduce additional structure to the data, which should generally be avoided. While this approach requires only one variable instead of p, it imposes an ordering among the colors. Moreover, it results in color c_i being closer to color $c_i + 1$ than to color $c_i + 2$ in terms of Euclidean distance.

2.2 Exercise two

Consider a dataset comprising workers' attributes such as the number of hours spent working (x_1), the number of completed projects (x_2), and whether they received a bonus (t). After applying logistic regression, we obtain the following coefficients: $w_0 = -6$, $w_1 = 0.05$, and $w_2 = 1$.

1. Determine the likelihood of a worker receiving a bonus given that they worked for 40 hours and completed 3.5 projects.
2. Calculate the number of hours a worker needs to work to have a 50% chance of receiving a bonus.
3. Discuss whether values of z in $\sigma(z)$ lower than -6 are meaningful in this context, and provide reasoning.

Solution

1. The logistic model yields the probability of receiving a bonus as its output, expressed by:

$$\mathrm{P}(t = 1|\mathbf{x}) = \sigma(w_0 + w_1 x_1 + w_2 x_2)$$

   Given $x_1 = 40$ and $x_2 = 3.5$:

$$\mathrm{P}(t = 1|\mathbf{x}) = \sigma(-6 + 0.05 \cdot 40 + 1 \cdot 3.5) = \sigma(-0.5) = 0.3775$$

2. To ascertain the probability of receiving a bonus with a confidence level $\alpha\%$, we need to invert the sigmoidal function. However, in this instance, a 50% chance corresponds to the sigmoidal argument being zero. Hence:

$$w_0 + w_1\hat{x} + w_2 x_2 = 0 \rightarrow -6 + 0.05\hat{x} + 3.5 = 0 \rightarrow \hat{x} = 50$$

3. Considering that all the variables under consideration are positive definite, it is reasonable to regard predictions with values greater than -6 as meaningful.

2.3 Exercise three

Consider a binary classifier trained on a dataset comprising $N = 100$ samples.

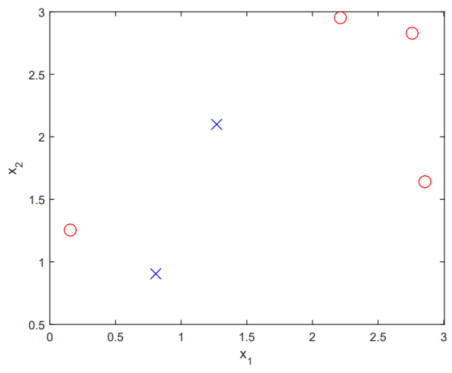
1. Given a precision of 0.25 and an F1 score of 0.4, compute the recall.
2. Additionally, with an accuracy of 0.85, calculate the complete confusion matrix.
3. Under what circumstances is accuracy not a dependable metric for evaluating the model's quality?

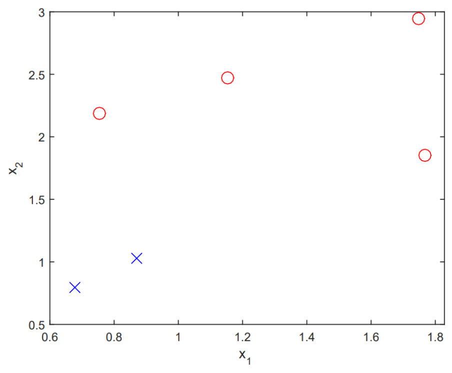
Solution

1. We have:
$$\text{F1} = \frac{2 \cdot \text{Pre} \cdot \text{Rec}}{\text{Pre} + \text{Rec}} = 0.4$$
$$\text{Pre} = \frac{TP}{TP + FP} = 0.25$$
We seek to find:
$$\text{Rec} = \frac{TP}{TP + FN}$$
Substituting, we obtain:
$$\frac{2 \cdot 0.25 \cdot \text{Rec}}{0.25 + \text{Rec}} = 0.4 \to \text{Rec} = 1$$
2. Given:
$$\text{F1} = \frac{2 \cdot \text{Pre} \cdot \text{Rec}}{\text{Pre} + \text{Rec}} = 0.4$$
$$\text{Pre} = \frac{TP}{TP + FP} = 0.25$$
$$\text{Rec} = \frac{TP}{TP + FN} = 1$$
$$\text{Acc} = \frac{TP + TN}{N} = 0.85$$
We infer that $FN = 0$ since the recall is unity. Then, from the other formulas:
$$\begin{cases} \text{TP} + \text{TN} = 85 \\ \text{FN} = 0 \\ \text{TP} = 0.25(\text{TP} + \text{FP}) \\ \text{TP} + \text{TN} + \text{FN} + \text{FP} = 100 \end{cases} \to \begin{cases} \text{TP} = 5 \\ \text{FN} = 0 \\ \text{FP} = 15 \\ \text{TN} = 80 \end{cases}$$
3. Accuracy is not a reliable indicator of the model's quality primarily under two conditions: when the dataset is imbalanced, and when the consequences of misclassifying positive-class samples differ from misclassifying negative-class samples.

2.4 Exercise four

Consider the provided datasets:





Now, let's analyze whether the learning procedure terminates and the number of steps required for convergence using the online stochastic gradient descent algorithm to train a perceptron.

Solution

The perceptron learning algorithm converges if there exists a linear separation hyperplane. In such a scenario, the classification error can be reduced to zero. If no linear separation exists, the optimization process doesn't halt. The convergence rate isn't assured since it relies on the initial parameterization and the sequence of points used for training. Nonetheless, convergence does occur within a finite number of steps.

In the first dataset (left), convergence isn't guaranteed. However, in the second dataset (right), the online stochastic gradient descent will ultimately converge within a finite number of steps.

Considering that the loss function for logistic regression is convex, the online learning process converges to the global optimum asymptotically, irrespective of the dataset provided.

CHAPTER 3

Bias-variance tradeoff

3.1 Exercise one

As you're fitting a linear model to your dataset, you contemplate transitioning to a quadratic model, incorporating quadratic features $\varphi(x) = \begin{bmatrix} 1 & x & x^2 \end{bmatrix}$. Considering this change, which of the following statements is most likely true:

1. Employing the quadratic model will reduce your reducible error.
2. Employing the quadratic model will decrease the bias in your model.
3. Employing the quadratic model will reduce the variance in your model.
4. Employing the quadratic model will decrease your reducible error.

Provide motivations to your answers.

Solution

1. False: changing the model doesn't directly impact the irreducible error, as it's inherent to the problem itself and can't be mitigated by model choice.
2. True: expanding the model to include quadratic features increases its flexibility, making it better able to capture complex relationships in the data. Therefore, it's likely to reduce bias or, at the very least, not increase it.
3. False: introducing more complex features typically leads to increased variance as the model becomes more sensitive to fluctuations in the training data.
4. Partially true: whether using the quadratic model decreases the reducible error depends on the balance between bias reduction and variance increase. If the quadratic model effectively reduces bias without excessively inflating variance, it can lead to a more accurate model and decrease reducible error. However, if the increase in variance outweighs the reduction in bias, the overall error may increase. Therefore, the statement could be true or false depending on the specific circumstances.

3.2 Exercise two

We determine the regression coefficients in a linear regression model by minimizing ridge regression for a specific value of λ. For each of the following, elucidate the trend of the elements as we increment λ from 0 (e.g., remains constant, increases, decreases, increases and then decreases):

1. The training RSS.
2. The test RSS.
3. The variance.
4. The squared bias.
5. The irreducible error.

Solution

1. Increases: as λ increases, simpler models are favored, leading to a decrease in flexibility and an inability to fit the training data precisely. Consequently, the training RSS will steadily increase.
2. Decreases and then increases. Initially, as λ increases, the test RSS improves due to a reduction in overfitting on the training data. However, beyond a certain point, overly simplistic models fail to capture the true underlying patterns, causing the test RSS to increase.
3. Decreases: increasing λ forces the use of simpler models, which inherently reduces the variability of the fits across different datasets.
4. Increases: with higher λ values, simpler models are employed, likely resulting in larger squared bias as these models may fail to capture the true underlying relationships adequately.
5. Remains constant: increasing λ does not affect the irreducible error since it is independent of the model's complexity and solely depends on the inherent noise in the data.

3.3 Exercise three

What methods would you employ to assess the efficacy of various models given the following scenarios:

1. Limited dataset size with straightforward models.
2. Limited dataset size with intricate models.
3. Extensive dataset with basic models.
4. Extensive dataset with access to parallel computing capabilities for training.

Justify you choices.

Solution

1. Leave-one-out cross-validation (LOO): when dealing with a small dataset and simple models, LOO is a viable option as it doesn't pose significant computational complexity. This method offers a nearly unbiased estimation of the test error.

2. Akaike information criterion (AIC) with Adjustment Techniques: with a smaller dataset, training might lead to overfitting, rendering traditional methods ineffective. AIC, with its adjustment techniques, can help mitigate overfitting concerns. However, for complex models, LOO may still be impractical due to computational constraints.

3. Cross-validation (CV): cross-validation is suitable for obtaining stable estimates to select the best model, particularly when Leave-One-Out is infeasible due to computational complexity. It balances the need for reliable estimates with computational efficiency.

4. Parallelized leave-one-out (LOO): in scenarios where parallel computing resources are available, and a large dataset is being utilized, parallelizing LOO can significantly reduce computation time. By concurrently training multiple models, the time required for LOO can be reduced by a factor equal to the number of parallel processes running simultaneously.

CHAPTER 4

Model selection

4.1 Exercise one

Consider the following statement regarding PCA and tell if they are true or false. Provide motivation for your answers.

1. Even if all the input features are on very similar scales, we should still perform mean normalization (so that each feature has zero mean) before running PCA.
2. Given only scores t_i and the loadings W, there is no way to reconstruct any reasonable approximation to x_i.
3. Given input data $x_i \in \mathbb{R}^d$, it makes sense to run PCA only with values of k that satisfy $k \leq d$.
4. PCA is susceptible to local optima, thus trying multiple random initializations may help.

Solution

1. True: since the principal components are identifying the directions where the most of the variance of the data is present, where the directions is defined as a vector with tail in the origin, we should remove the mean values for each component in order to identify correctly these directions.
2. False: by applying again the loadings matrix W to the scores t_i, thanks to the orthogonality property of W, we are able to reconstruct perfectly the original mean normalized vectors. If we want to reconstruct the original vectors we should also store the mean values for each dimension.
3. True: running it with $k = d$ is possible but usually not helpful and $k > d$ does not make sense.
4. False: there is no source of randomization and no initialization point in the algorithm to perform PCA.

4.2 Exercise two

State whether the following claims about Bagging and Boosting are true or false, motivating your answers:

1. Since Boosting and Bagging are ensemble methods, they can be both parallelized.
2. Bagging should be applied with weak learners.
3. The central idea of Boosting consists in using bootstrapping.
4. It is not a good idea to use Boosting with a deep neural network as a base learner.

Solution

1. False, only Bagging can be parallelized, since training is done on different datasets, while Boosting is sequential by nature.
2. False, weak learners are good candidate for Boosting, since they have low variance. Typically, one uses instead bagging when more complex and unstable learners are needed, to reduce their variance.
3. False, bootstrapping is used in bagging, whose name derived indeed from boosting aggregation.
4. True, it is not a good idea to do that, since deep neural networks are very complex predictor, which can have large variance. Therefore, you may not succeed in lowering bias without increasing variance. Moreover, since you need to train the network multiple times, the procedure may require a lot of time.

4.3 Exercise three

Answer to the following questions about the bias-variance decomposition, model selection, and related topics. Motivate your answers.

1. If your linear regression model underfits the training data (i.e., the model is not complex enough to explain the data), would you apply PCA to compute a more suitable feature space for your model?
2. If solving a regression problem, the design matrix $X^T X$, is singular, would you apply PCA to solve this issue?
3. Assuming a classifier fits very well the training data but underperforms on the validation set, would you apply Bagging or Boosting to improve it?
4. Assuming that you trained a classifier with a K-fold cross-validation and it consistently has poor performances both on training and on validation folds, would you apply Bagging or Boosting to improve it?
5. You applied ridge regression to train a linear model using a rather large regularization coefficient, would you think that bagging would improve your model?

6. You have been asked to implement a feature selection process on a system with very limited computational resources. Would you opt for a filter approach or for a wrapper approach?

7. You have been asked to implement a feature selection process to improve as much as possible the performance of your model. Would you opt for a filter approach or for a wrapper approach?

8. You need to train a linear regression model using as input the readings of several sensors. Assuming that you know that some of these sensors might be faulty (i.e., resulting in meaningless readings), which linear regression approach would you use to train your model?

9. A linear regression model, computed using ordinary least squares, has a validation error that is much larger than training error. Assuming that you do not want to change neither the input features nor the kind of model, what would you do to improve it?

10. If you have to choose among a few models knowing only the training error (assuming you cannot retrain them or evaluate them on a different dataset), what would you do?

Solution

1. No, because if linear regression does not fit training data (underfitting), the features computed with PCA will not solve the problem as they are linear combinations of input variables.

2. Yes, applying PCA and selecting the top K components we will allow to avoid co-linearity in the resulting feature space.

3. Bagging, because it might reduce the variance of the model. In contrast, boosting allows to reduce bias without increasing (significantly) the variance (however, in this example we have a low bias and high variance).

4. Boosting, because it can successfully reduce bias of a stable learner without increasing the variance which seems to be the problem of the learner in this case.

5. No, because ridge regression with large regularization coefficient will be very stable (limited variance) and this would not allow to exploit significantly bagging.

6. Filter, because a wrapper approach involves solving an optimization problem that requires training several models. In contrast, filter approaches only require to compute statistics on the features.

7. Wrapper, because filter approaches assume features are independent and might not find the best subset.

8. Lasso, because it implicitly performs a feature selection that will get rid of the faulty sensors.

9. Regularization can help me improve the performance by reducing the variance. In particular ridge regression would be the most obvious choice. Another solution, if viable, is to increase the number of samples used for training.

10. Adjusted complexity matrix can be used to correct the training error taking into account also the model complexity.

CHAPTER 5

Kernel methods

5.1 Exercise one

Tell if the following functions are valid kernels. Motivate your answers. Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$:

1. $k_1(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y} + \mathbf{x}^T\mathbf{1} + \mathbf{y}^T\mathbf{1} + \mathbf{d}$, where $\mathbf{1} \in \mathbb{R}^d$ is the vector of all ones.
2. $k_2(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y} - \|\mathbf{x}\|^2$.
3. $k_3(\mathbf{x}, \mathbf{y}) = k_1(\cos(\mathbf{x}), \cos(\mathbf{y}))^3$, where the $\cos(\cdot)$ function is applied element-wise.
4. $k_4(\mathbf{x}, \mathbf{y}) = e^{(k_2(\mathbf{x},\mathbf{y})+k_2(\mathbf{x},\mathbf{y}))}$.

Solution

1. True: by definition, selecting $\phi(\mathbf{x}) = \mathbf{x} + \mathbf{1}$, we have $k_1(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^T\phi(\mathbf{y})$.
2. False: $k_2(\mathbf{x}, \mathbf{y})$ is not symmetric.
3. True: since we are considering the same transformation $\cos(\cdot)$ applied to both arguments, then $(\cdot)^3$ is a polynomial transformation with non-negative coefficients, and k_1 is a kernel.
4. True: with simple computations, we obtain:
$$k_4(\mathbf{x}, \mathbf{y}) = e^{-k_2(\mathbf{x},\mathbf{y})+k_2(\mathbf{y},\mathbf{x})} = e^{2\mathbf{x}^T\mathbf{y}-\|\mathbf{x}\|^2-\|\mathbf{y}\|^2} = e^{-\|\mathbf{x}-\mathbf{y}\|^2}$$
That is the Gaussian kernel with $\sigma = \frac{1}{2}$.

5.2 Exercise two

Comment on the following statements about Gaussian Processes (GP). Assume to have a dataset generated from a GP $\mathcal{D} = (x_i, y_i)_{i=1}^N$. Motivate your answers.

1. GPs are parametric methods.
2. The computation of the estimates of the variance of the GP $\hat{\sigma}^2(x)$ corresponding to the input x provided by $\mathcal{D}$ does not require the knowledge of the samples output $(y_1, \ldots, y_N)$.

3. In the neighborhood of the input points $(x_1, \dots, x_N)$, we observed the variance of the GP gets smaller and smaller as we collect more samples.

4. The complexity of the computation of the estimates of the mean $\hat{\mu}(x)$ and variance $\hat{\sigma}^2(x)$ scales as N^3, i.e., cubically with the number of samples N.

Solution

1. False: they require to store the gram matrix whose dimension depends on the number of samples.

2. True: it requires only the gram matrix and the computation of the kernel on the new point.

3. True: the uncertainty we have around the sampled points decreases as we get more and more samples.

4. True: indeed, it requires the inversion of the gram matrix which has N^3 computational cost.

5.3 Exercise three

Consider the linear two-class SVM classifier defined by the parameters $w = \begin{bmatrix}2 & 1\end{bmatrix}$, $b = 1$. Answer the following questions providing adequate motivations.

1. Is the point $x_1 = \begin{bmatrix}-2 & 4\end{bmatrix}$ a support vector?

2. Give an example of a point which is on the boundary of the SVM.

3. How the point $x_2 = \begin{bmatrix}3 & -1\end{bmatrix}$ is classified according to the trained SVM?

4. Assume to collect a new sample $x_3 = \begin{bmatrix}-1 & 2\end{bmatrix}$ in the negative class, do we need to retrain the SVM?

Solution

1. A point is a support vector if $\left|w^T x + b\right| \leq 1$, thus:

$$\left|w^T x + b\right| = |-4 + 4 + 1| = 1$$

   Meaning that x_1 is a support vector.

2. A point on the boundary has to satisfy $w^T x + b = 0$ thus by considering $x_{11} = 0$:

$$2 \cdot 0 + 1 \cdot x_{22} + 1 = 0 \rightarrow x_{22} = -1$$

   Thus $x = \begin{bmatrix}0 & -1\end{bmatrix}$ is on the boundary.

3. A point is classified either in the positive class or in the negative one if $w^T x + b$ is positive or negative, respectively, thus:

$$w^T x_2 + b = 2 \cdot 3 - 1 \cdot 1 + 1 = 6$$

   Which means that the point it classified in the positive class.

4. The point is misclassified by the current model, as it is

$$w^T x_3 + b = -2 + 2 + 1 = 1$$

Which means that x_3 would be a support vector, thus we need to retrain the model.

CHAPTER 6

Learning theory

6.1 Exercise one

1. Show that the VC dimension of an axis aligned rectangle is 4.
2. Show that the VC dimension of a linear classifier in 2D is 3.
3. Show that the VC dimension of a triangle in the plane is at least 7.
4. Show that the VC dimension of a 2D stump, i.e., use either a single horizontal or a single vertical line in 2D to separate points in a plane, is 3.

Solution

1. Consider 4 points. It is possible to show by enumeration that all the possible labeling are shattered by the rectangle. Consider 5 points. Consider the set of points with maximum and minimum x coordinate and maximum and minimum y coordinates. If all the points are on the rectangle, we consider the labeling which assign alternate labels to the points if you follow the rectangle perimeter. Otherwise, there are at most 4 points in this set. If we label them $+$ and label $-$ the other, it is not possible to shatter this labeling.
2. Let us call the class of all the linear classifier in 2D $\mathcal{H}$. The proof consists in two steps:
   (a) $VC(\mathcal{H}) \geq 3$ and $VC(\mathcal{H}) \leq 3$. We need to show that it exists a set of 3 points that can be shattered by the considered hypothesis space $\mathcal{H}$. By considering a set of three non-aligned points, it is possible to show by enumeration that it is possible to shatter them with a linear classifier. Thus, $VC(\mathcal{H}) \geq 3$.
   (b) We need to show that it does not exists a set of 4 points which can be shattered by a linear classifier. There are different cases:
      i. Four aligned points: if we alternate instances coming from the positive and negative classes we cannot shatter them.
      ii. Three aligned points and a fourth on an arbitrary position: if we alternate instances coming from the positive and negative classes for the three aligned points, we cannot shatter them.

iii. Four points on a convex hull: if we label the points on the two diagonals with opposite classes, we cannot shatter them.
iv. Three points on a convex hull (triangle) and one inside the hull: if we label the three points on the triangle with a label and the last one with the other class, we cannot shatter them.

Since there does not exists a configuration where we can shatter the points, we have that $VC(\mathcal{H}) \leq 3$.

3. If we consider a set of points on a circle it is possible to show by enumeration that a triangle ia able to shatter all of them.

4. Since the decision stumps in 2D are a model which is less flexible than linear boundaries, which have $VC = 3$, they should have $VC(\mathcal{H}) \leq 3$. The proof that $VC(\mathcal{H}) = 3$ is by enumeration:

CHAPTER 7

Markov Decision Processes

APPENDIX A

Linear regression

A.1 Introduction

Let's examine the Iris dataset, which comprises the following features for each sample:

1. Sepal length.
2. Sepal width.
3. Petal length.
4. Petal width.
5. Species (including Iris setosa, Iris virginica, and Iris versicolor).

This dataset consists of a total of $N = 150$ samples, with each species contributing 50 samples.

Using linear regression, we can extract valuable insights from the data. From the data, we can infer relationships between variables and make predictions based on them. We can provide forecasts for various quantities using newly observed data. Specifically, we can predict the petal width of a particular type of Iris setosa by leveraging the relationship with petal length through linear regression. In this scenario, the target variable is continuous ($t_n \in \mathbb{R}$), indicating a regression problem.

A.2 Linear regression

Preliminary operations The preliminary operations on data are:

- Loading data: load the dataset into memory.
- Inspecting data: examine the dataset to understand its structure and contents.
- Selecting interesting data: identify and choose the relevant features or variables from the dataset.
- Preprocessing: prepare the data for further analysis by performing various preprocessing steps such as:

- Shuffling the data to randomize the order of samples (`shuffle ()`).
- Removing inconsistent data points that may contain errors or inconsistencies.
- Removing outliers to ensure data quality.
- Normalizing or standardizing the data to bring all features to a similar scale.
- Filling missing data, for example, by replacing `NaN` values with appropriate values.

These preprocessing steps help ensure that the data is clean, consistent, and ready for analysis.

Data normalization We can normalize a series of samples $\{s_1, \dots, s_N\}$ to a sample s using:

- Z-score:
$$\frac{s - \bar{s}}{S}$$
Here, $\bar{s} = \frac{1}{N} \sum_{n=1}^{N} s_n$ and $S^2 = \frac{1}{N-1} \sum_{n=1}^{N} (s_n - \bar{s})^2$.
- Min-max feature scaling:
$$\frac{s - s_{\min}}{s_{\max} - s_{\min}}$$
Here, $s_{\max} = \max_{n \in \{1,\dots,N\}} s_n$ and $s_{\min} = \min_{n \in \{1,\dots,N\}} s_n$.

A.2.1 Functions definition

To proceed with our analysis, we must define the following elements:

- Hypothesis space: we consider linear models represented by:
$$\hat{t} = y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j x_j = \mathbf{w}^T \mathbf{x}$$
Here, $\mathbf{w} = \left[w_0, w_1, \dots, w_{M-1}\right]^T$ and $\mathbf{x} = \left[x_0, x_1, \dots, x_{M-1}\right]^T$.
- Loss function: the loss function is defined as the residual sum of squares over the N samples $\{(\mathbf{x}_n, t_n)\}_{n=1}^{N}$:
$$\text{RSS}(\mathbf{w}) = \sum_{n=1}^{N} (y(\mathbf{x}_n, \mathbf{w}) - t_n)^2$$
- Optimization method: various optimization methods can be employed, including closed form solutions, gradient descent, and others, to minimize the loss function and find the optimal values for the model parameters.

A.2.2 Result evaluation

Evaluation of the linear regression results involves several metrics:

- Residual sum of squares (RSS) or sum of squared errors (SSE):
$$\text{RSS}(\mathbf{w}) = \sum_{n=1}^{N} \left(\hat{t}_n - t_n\right)^2$$
Here, $\hat{t}_n = y(\mathbf{x}_n, \mathbf{w})$.

- Mean square error (MSE):
$$\text{MSE} = \frac{\text{RSS}(\mathbf{w})}{N}$$

- Root mean square error (RMSE):
$$\text{RMSE} = \sqrt{\frac{\text{RSS}(\mathbf{w})}{N}}$$

- Coefficient of determination (R^2):
$$R^2 = 1 - \frac{\text{RSS}(\mathbf{w})}{\text{TSS}}$$

  Here $\text{TSS} = \sum_{n=1}^{N} (\bar{t} - t_n)^2$ is the total sum of squares and $\bar{t} = \frac{1}{N}\sum_{n=1}^{N} t_n$.

- Degrees of freedom (dfe):
$$\text{dfe} = N - M$$

- Adjusted coefficient of determination (R^2_{adj}):
$$R^2_{adj} = 1 - \left(1 - R^2\right)\frac{N-1}{\text{dfe}}$$

A.2.3 Python implementation

We have several solutions available from different libraries:

- Utilizing `sklearn` with `LinearRegression ()`.
- Utilizing `statsmodels` with `OLS`.
- Implementing it manually as $\mathbf{w}^* = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{t}$.

With the `sklearn` option:

- Initialize a linear model (`LinearRegression ()`).
- Fit the model to the data (`fit ()`).
- Analyze the results.

A.3 Statistical tests on coefficients

Let's consider the scenario where t_n satisfies $t_n = \mathbf{w}\mathbf{x}_n + \epsilon_n$, where ϵ_n s an independent and identically distributed white zero-mean noise with variance σ^2. In this case, the exact distribution of the statistic is given by:
$$\frac{\hat{w}_j - w_j}{\hat{\sigma}\sqrt{v_j}} \sim t_{N-M}$$

Here:

- w_j represents the true parameters.

- $\hat{w}_j$ denotes the estimated parameter with N samples.
- v_j stands for the j-th diagonal element of the matrix $\left(\mathbf{X}^T\mathbf{X}\right)^{-1}$.
- t_{N-M} is the Student's t-distribution with dfe $= N - M$ degrees of freedom.
- $\hat{\sigma}^2$ is the unbiased estimate for the target variance:

$$\hat{\sigma}^2 = \frac{\text{RSS}(\hat{\mathbf{w}})}{N-M}$$

Based on this, we can conduct the following tests:

- Test on single coefficients $j \in \{0, \dots, M-1\}$: we compare $H_0 : w_j = 0$ against $H_1 : w_j \neq 0$. Then, we apply:

$$t_{stat} = \frac{\hat{w}_j - w_j}{\hat{\sigma}\sqrt{v_j}} \sim t_{N-M}$$

  Here, t_{N-M} follows the Student's t-distribution with $N - M$ degrees of freedom.

- Test on the overall significance of the model: we compare $H_0 : w_1 = \cdots = w_{M-1} = 0$ against $H_1 : \exists j \in \{1, \dots, M-1\}$ such that $w_j \neq 0$. Then, we use:

$$F_{stat} = \frac{N-M}{M-1}\frac{\text{TSS} - \text{RSS}(\hat{\mathbf{w}})}{\text{RSS}(\hat{\mathbf{w}})} \sim F_{M-1,N-M}$$

  Here, $F_{M-1,N-M}$ represents the Fisher-Snedecor distribution with parameters $M-1$ and $N-M$. This test compares the full linear model $y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j x_j$ with $N-M$ degrees of freedom against the constant model $y(\mathbf{x}, w_0) = w_0$ with $N-1$ degrees of freedom.

APPENDIX B

Classification

B.1 Introduction

Let's examine the Iris dataset, which comprises the following features for each sample:

1. Sepal length.
2. Sepal width.
3. Petal length.
4. Petal width.
5. Species (including Iris setosa, Iris virginica, and Iris versicolor).

This dataset consists of a total of $N = 150$ samples, with each species contributing 50 samples.

Using petal and sepal length and width variables, it is possible to predict the species of Iris (target). In this context, the target variables are discrete and non-metric, indicating a classification problem. The targets are referred to as classes.

To tackle the problem of predicting the species of Iris based on petal and sepal measurements, several approaches can be considered:

- *Discriminant function approach*: in this method, the model functions as a mapping of inputs to classes, expressed as $f(x) = C_k \in \{C_1, \dots, C_K\}$. The process involves fitting the model to the available data.
- *Probabilistic discriminative approach*: here, the model represents a conditional probability distribution, $\mathrm{P}(C_k|\mathbf{x}) \in [0, 1]$. The approach entails fitting the model to the data to establish probabilities.
- *Probabilistic generative approach*: the model incorporates the likelihood $\mathrm{P}(\mathbf{x}|C_k) \in [0, 1]$ and the prior $\mathrm{P}(C_k) \in [0, 1]$ The process involves fitting the model to the available data. Inference is made using the posterior probability with the Bayes rule. New samples can be generated from the joint distribution $\mathrm{P}(C_k|\mathbf{x}) = \mathrm{P}(\mathbf{x}|C_k)\mathrm{P}(C_k)$.

The classification problem can be addressed through various methods, including linear classification methods like the perceptron and logistic regression, alongside algorithms like naïve Bayes and K-nearest neighbor.

B.1.1 Preliminary operations

As usual before solving the problem we need to perform some preliminary operations:

- Load the data.
- Consistency checks.
- Select and normalize the input.
- Shuffle the data (`shuffle ()`).
- Generate the output ($t_n \in \{0, 1\}$ or $t_n \in \{-1, 1\}$).
- Explore the selected data (`scatter`).

B.2 Perceptron

The perceptron operates as a discriminant function approach since it directly assigns elements into classes without providing probabilities.

Functions definition To proceed with our analysis, we must define the following elements:

- Hypothesis space: we consider linear models represented by:

$$y(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x}) = \text{sign}(w_0 + x_1w_1 + x_2w_2)$$

  Here,

$$\text{sign}(z) = \begin{cases} -1 & \text{if } z < 0 \\ +1 & \text{otherwise} \end{cases}$$

- Loss function: distance of misclassified points in $\{(\mathbf{x}_n, t_n)\}_{n=1}^N$ with $t_n \in \{-1, 1\}$:

$$L_P(\mathbf{w}) = -\sum_{n\in\mathcal{M}} \mathbf{w}^T\mathbf{x}_n t_n$$

  Here, $\mathcal{M} = \{n \in \{1, \dots, N\} : t_n \neq y(\mathbf{x}_n)\}$

- Optimization method: online gradient descent

B.2.1 Result evaluation

This method converges when the data are linearly separable. To visualize the separating hyperplane or decision boundary (line) we need to plot:

$$\text{sign}(\mathbf{x}^T\mathbf{x}) = 0 \to \text{sign}(w_0 + x_1w_1 + x_2w_2) = 0 \to x_2 = -\frac{w_1x_1 + w_0}{w_2}$$

To evaluate the performance of a classifier, we can to compute the confusion matrix which tells us the number of points which have been correctly classified and those which have been misclassified

		Actual class	
		1	0
Predicted class	1	True positive	False positive
	0	False negative	True negative

From this table we can compute the following measures:

- Accuracy: fraction of the samples correctly classified in the dataset:
$$\text{Acc} = \frac{TP + TN}{N}$$
- Precision: fraction of samples correctly classified in the positive class among the ones classified in the positive class:
$$\text{Pre} = \frac{TP}{TP + FP}$$
- Recall: fraction of samples correctly classified in the positive class among the ones belonging to the positive class:
$$\text{Rec} = \frac{TP}{TP + FN}$$
- F1 score: harmonic mean of the precision and recall:
$$\text{F1} = \frac{2 \cdot \text{Pre} \cdot \text{Rec}}{\text{Pre} + \text{Rec}}$$

The higher these figures of merits the better the algorithm is performing. These performance measures are not symmetric, but depend on the class we selected as positive. Depending on the application one might switch the classes to have measures which better evaluate the predictive power of the classifier.

B.2.2 Python implementation

Implementation in Python can be accomplished either through the `sklearn` library utilizing the `Perceptron` module, or by manually coding the algorithm.

B.3 Logistic regression

Logistic regression functions as a probabilistic discriminative approach by directly assigning probabilities to elements belonging to certain sets.

B.3.1 Functions definition

To proceed with our analysis, we must define the following elements:

- Hypothesis space:
$$y(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x}) = \sigma(w_0 + x_1 w_1 + x_2 w_2)$$
Here, $\sigma = \frac{1}{1+e^z}$.

- Loss function: distance of misclassified points in $\{(\mathbf{x}_n, t_n)\}_{n=1}^{N}$ with $t_n \in \{-1, 1\}$:

$$L_P(\mathbf{w}) = \mathrm{P}(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \sum_{n=1}^{N} t_n \ln y(\mathbf{x}_n) + (1 - t_n) \ln(1 - y(\mathbf{x}_n))$$

- Optimization method: online gradient descent.

Result evaluation To visualize the separating hyperplane or decision boundary (line) we need to plot:

$$\sigma(\mathbf{x}^T\mathbf{x}) = \frac{1}{2} \to \sigma(w_0 + x_1 w_1 + x_2 w_2) = \frac{1}{2} \to x_2 = -\frac{w_1 x_1 + w_0}{w_2}$$

B.3.2 Python implementation

Implementation in Python can be accomplished through the `sklearn` library utilizing the `LogisticRegression` module.

B.3.3 Logit

Given the function:

$$\mathit{logit}(y) = \log\left(\frac{y}{1-y}\right)$$

We can utilize it with the output of logistic regression:

$$\mathrm{logit}(y(\mathbf{x})) = \mathbf{w}^T\mathbf{x} = w_0 + x_1 w_1 + x_2 w_2$$

This exhibits a similar characterization to linear regression. Consequently, we can conduct hypothesis testing to ascertain the significance of the parameters.

B.4 Naïve Bayes

The naïve assumption posits that within the class C_k, each input is conditionally independent of one another. In this scenario the decision function, given a prior $\mathrm{P}(C_k)$, maximize the Maximum A Posteriori (MAP) probability:

$$y(\mathbf{x}) = \operatorname*{argmax}_k \mathrm{P}(C_k) \prod_{i=1}^{M} \mathrm{P}(x_j|C_k)$$

B.4.1 Functions definition

So with this approach we have:

- Hypothesis space:

$$y(\mathbf{x}) = \operatorname*{argmax}_k \mathrm{P}(C_k) \prod_{i=1}^{M} \mathrm{P}(x_j|C_k)$$

  Here, $\sigma = \frac{1}{1+e^z}$.

- Loss function: log likelihood for fitting both the priors $\mathrm{P}(C_k)$ and the likelihoods $\mathrm{P}(x_j|C_k)$.
- Optimization method: maximum likelihood estimation (MLE).

In our classification problem, we opt for the following:

- Prior: $\mathrm{P}(C_k)$ multinomial distribution with parameters $(p_1, \dots, p_k)$.
- Likelihood: $\mathrm{P}(x_j|C_k)$ follows a normal distribution $\mathcal{N}(\mu_{jk}, \sigma^2_{jk})$, where each feature x_j and each class C_k are represented.

We may select different distributions for the features based on the input.

B.4.2 Generative method

Due to the generative capabilities of the naïve Bayes classifier, we can generate datasets that resemble the original one using the following steps:

1. Select a class $C_{\hat{k}}$ according to the prior multinomial distribution with parameters:
$$\hat{\mathrm{P}}(C_1), \dots, \hat{\mathrm{P}}(C_k)$$
2. For each feature j, draw a sample x_j from $\mathcal{N}(\hat{\mu}_{j\hat{k}}, \hat{\sigma}^2_{j\hat{k}})$.
3. Repeat these steps whenever a new sample is desired.

B.4.3 Python implementation

We can implement naïve Bayes in Python using two approaches:

- Utilizing pre-implemented `sklearn GaussianNB`, with a multinomial distribution as prior and Gaussian distributions as likelihood.
- Implementing it by hand:
  - Estimate the prior: $\hat{\mathrm{P}}(C_k) = \frac{\sum_{i=1}^{N} I\{\mathbf{x}_n \in C_k\}}{N}$
  - Estimate the Maximum Likelihood Estimation (MLE) parameters: $\mathrm{P}(x_j|C_k) = \mathcal{N}(x_j; \hat{\mu}_{jk}, \hat{\sigma}^2_{jk})$, where $\hat{\mu}_{jk}$ and $\hat{\sigma}^2_{jk}$ are computed by maximizing the likelihood.
  - Compute $\mathrm{P}(C_k) \prod_{j=1}^{M} \mathrm{P}(x_j|C_k)$ for each class C_k and choose the maximum one.

Notice that naïve Bayes isn't strictly a Bayesian method, as the priors are estimated from data and not updated using likelihoods.

B.5 K-nearest neighbor

The k-nearest neighbor is a discriminative function approach.

1-nearest neighbor The concept revolves around leveraging nearby points to predict the target of a new point. Given a dataset $\{\mathbf{x}_n, t_n\}_{i=1}^N$ and a new data point $\mathbf{x}_q$, we predict the target as:

$$i_q \in \underset{n\in\{1,\dots,N\}}{\operatorname{argmin}} \|\mathbf{x}_q - \mathbf{x}_n\|_2$$

Resulting in $\hat{t}_q = t_{i_q}$. This approach seamlessly caters to both regression and classification tasks without explicit training. The training process is essentially querying the dataset.

Key design choices include the selection of the distance function and the number of neighbors. If $k > 1$, the targets can be combined as follows:

- For classification, predict the mode class (with tie-breaking rules):

$$\hat{t}_q \in \underset{C_k}{\operatorname{argmax}} |\{i \in \mathcal{N}_k(\mathbf{x}_q) : t_i = C_k\}|$$

- For regression, predict the average target:

$$\hat{t}_q = \frac{1}{k} \sum_{i\in\mathcal{N}_k(\mathbf{x}_q)} t_i$$

- Other approaches include providing a probability distribution instead of a class, using weights proportional to the inverse of the distance.

The choice of k introduces varying degrees of regularization, ranging from strong to mild regularization.

B.6 Summary

	Parametric	**Frequentist**	**Category**
Perceptron	✓	✓	Discriminative function
Logistic regression	✓	✓	Probabilistic discriminative
Naïve Bayes	✓	✓	Probabilistic generative
K-Nearest neighbor	✗	✗	Discriminative function

B.7 Multiple classes

If we're dealing with multiple classes, we can utilize the same function by providing a target with more than two labels. This approach involves training k different models, each distinguishing one class from the rest. In this scenario, the parameter vector becomes a matrix W.

Although we can still visualize the separating surfaces, it becomes slightly more challenging compared to the binary classification case. However, extending these methods to handle multiple classes doesn't require any changes.

APPENDIX C

Bias-variance tradeoff

C.1 Introduction

To comprehensively analyze the variance and bias of a model, understanding the data generation process is essential. Consider the following data generation process:

$$t = f(x) + \varepsilon \qquad f(x) = 1 + \frac{1}{2}x + \frac{1}{10}x^2$$

Here are the specifics:

- Inputs x are uniformly distributed in the interval $[0, 5]$.
- The noise ε follows a distribution $\mathrm{P}(t|x)$ with $\mathbb{E}[\varepsilon|x] = 0$ and $\mathrm{Var}[\varepsilon|x] = \sigma^2 = 0.7^2$

For the learning problem, assuming we don't know the true model, we consider two alternative models:

- Linear model $\mathcal{H}_1 : y(x) = a + bx$.
- Quadratic model $\mathcal{H}_2 : y(x) = a + bx + cx^2$.

Both models can be interpreted as linear models: $y(x) = \mathbf{w}^T\boldsymbol{\phi}(x)$ with the following feature mappings:

- For $\mathcal{H}_1 : \boldsymbol{\phi}(x) = (1, x)^T$ and weights $\mathbf{w} = (a, b)^T$.
- For $\mathcal{H}_2 : \boldsymbol{\phi}(x) = (1, x, x^2)^T$ and weights $\mathbf{w} = (a, b, c)^T$.

C.2 Population risk minimization

Probability known We begin by assuming that $\mathrm{P}(x, t)$ is known a priori. We define:

- Hypothesis space: $y(x) \in \mathcal{H}$.
- Loss function: squared loss function $(t - y(x))^2$.

- Population risk minimization (PRM):

$$y^* \in \operatorname*{argmin}_{y\in\mathcal{H}} \mathbb{E}_{t,x}\left[(t-y(x))^2\right] = \int \mathrm{P}(x,t)(t-y(x))^2\,dxdt \overset{t=f(x)+\varepsilon}{=} \int \mathrm{P}(x)(f(x)-y(x))^2\,dx$$

If the true model is known, we can compute the optimal model for the two hypothesis spaces:

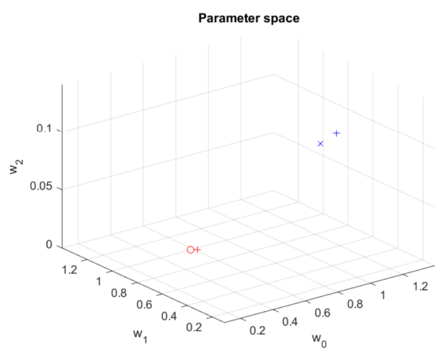
$$\begin{cases}\mathcal{H}_1 : y^* \in \operatorname{argmin}_{(a,b)\in\mathbb{R}^2} \int_0^5 \frac{1}{5}(f(x)-a-bx)^2\,dx = \left(\frac{7}{12},1\right)^T \\ \mathcal{H}_2 : y^* \in \operatorname{argmin}_{(a,b,c)\in\mathbb{R}^3} \int_0^5 \frac{1}{5}(f(x)-a-bx-cx^2)^2\,dx = \left(1,\frac{1}{2},\frac{1}{10}\right)^T\end{cases}$$

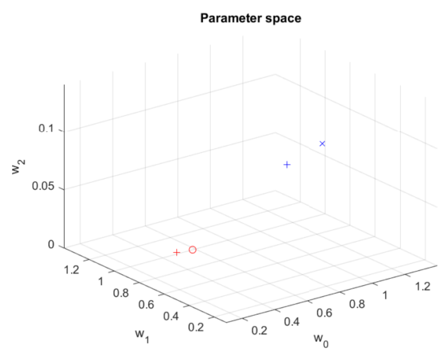
Probability unknown Now, let's assume that $\mathrm{P}(x,t)$ is not known a priori, but we possess a training dataset $\mathcal{D} = \{(x_n,t_n)\}_{n=1}^N$ of independent identically distributed random variables from P. We define:

- Hypothesis space: $y(x) \in \mathcal{H}$.
- Loss function: squared loss function $(t-y(x))^2$.
- Empirical risk minimization (ERM):

$$\hat{y} \in \operatorname*{argmin}_{y\in\mathcal{H}} \frac{1}{N}\sum_{n=1}^{N}(t_n - y(x_n))^2$$

Here, $\hat{y}$ is a random variable depending on the dataset $\mathcal{D}$.

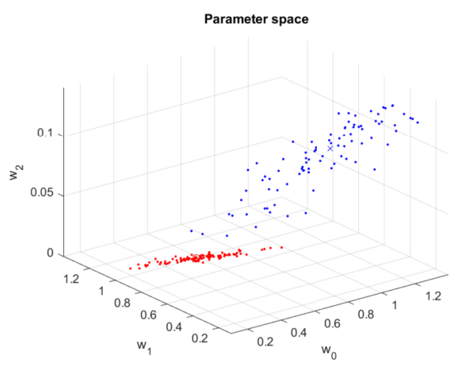


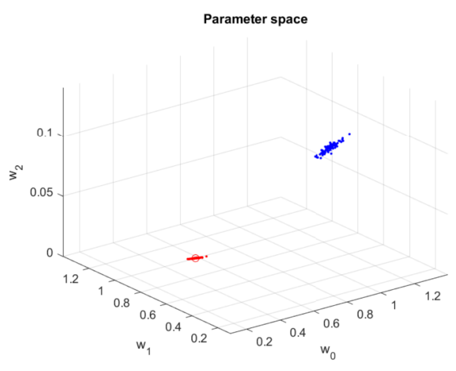


In the Probabilistic Risk Minimization (PRM) framework, the blue $\times$ symbolizes the top-performing model within $\mathcal{H}_2$, while the red $\circ$ denotes the best model within $\mathcal{H}_1$.

For the Empirical Risk Minimization (ERM) perspective, the $+$ signifies the optimal parameters discovered for two instances of the dataset $\mathcal{D}$, each comprising $N = 1000$ samples.

If we conduct ERM iteratively across multiple trials, generating a hundred independent datasets, with varying sample sizes ($N = 100$ on the left and $N = 10000$ on the right).





C.2.1 Error

The error can be expressed as:

$$\mathbb{E}_{\mathcal{D},t}\left[(t-\hat{y}(x))^2\right] = \sigma^2 + \text{Var}_{\mathcal{D}}\left[\hat{y}(x)\right] + \mathbb{E}_{\mathcal{D}}[f(x)-\hat{y}(x)]^2$$

Here:

- $\mathbb{E}\mathcal{D}, t\left[(t-\hat{y}(x))^2\right]$ represents the expected error, computed with respect to the training dataset $\mathcal{D}$ and the target t.
- σ^2 denotes the irreducible error.
- $\text{Var}\mathcal{D}\left[\hat{y}(x)\right]$ stands for the variance, which diminishes with an increase in the number of samples $N = |\mathcal{D}|$.
- $\mathbb{E}_{\mathcal{D}}[f(x)-\hat{y}(x)]^2$ represents the bias, influenced by the hypothesis space $\mathcal{H}$.

C.3 Bias-variance tradeoff

In practical scenarios, the true model is often unknown, requiring us to select the most appropriate model from a set of options. Let's examine the potential solutions for a regression problem:

- Hypothesis space: $y(x, \mathbf{w}) = f(x, \mathbf{w}) = \sum_{k=0}^{o} x^k w_k$.
- Loss function: $\frac{1}{N}\sum_{(x,t)\in\mathcal{D}}\left(y(x_n, \mathbf{w}) - t_n\right)^2$ on a dataset $\mathcal{D}$.
- Optimization method: Least Square (LS).

The order o and other parameters, chosen before training, are commonly referred to as hyperparameters.

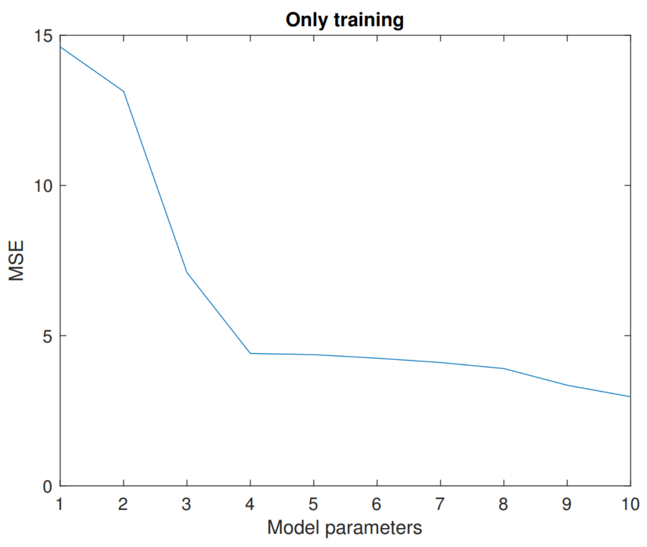


Figure C.1: Training error

The error decreases monotonically because the quality of a fixed model $\mathbf{w}$ is represented by the expected Mean Squared Error (MSE):

$$\text{MSE}(\mathbf{w}) := \mathbb{E}_{\mathbf{x},t}\left[(y(\mathbf{x},\mathbf{w}) - t)^2\right]$$

We train on $\mathcal{D}_{train}$ with $N = |\mathcal{D}_{train}|$ by minimizing the empirical MSE:

$$\hat{\mathbf{w}} \in \underset{\mathbf{w}\in\mathbb{R}^{o+1}}{\text{argmin}}\, \hat{\text{MSE}}_{train}(\hat{\mathbf{w}}) := \frac{1}{N}\sum_{(\mathbf{x},t)\in\mathcal{D}_{train}} (y(\mathbf{x},\mathbf{w}) - t)^2$$

Here:

- $\hat{\mathbf{w}}$ is statistically dependent on $\mathcal{D}_{train}$.
- $\hat{\text{MSE}}_{train}(\hat{\mathbf{w}})$ isn't a reliable estimate of MSE_{train} and can't be used for evaluating the performance of $y(\cdot,\mathbf{w})$ or for model selection.

To address this, we can divide the dataset into two parts:

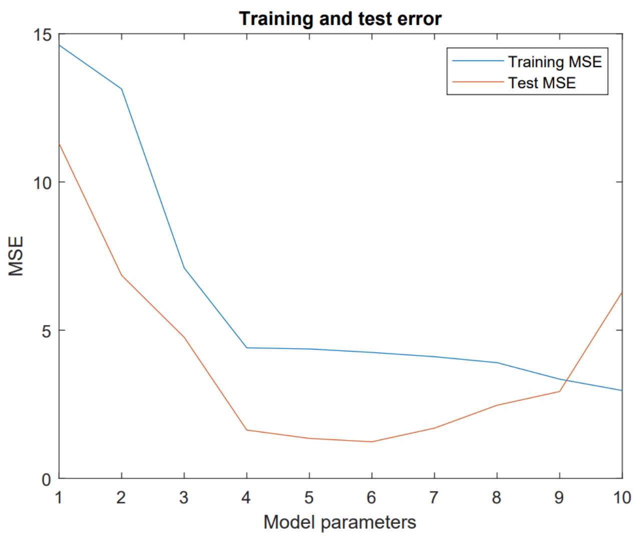


Figure C.2: Training error with two sets

Alternatively, we can employ a test set to evaluate the results, dividing the dataset as follows:

- Training set $\mathcal{D}_{train}$: data used for learning model parameters.
- Validation set $\mathcal{D}_{vali}$: data used for model selection.
- Test set $\mathcal{D}_{test}$: data used for evaluating model performance.

Typically, a split of 50%-25%-25% is used for the three sets.

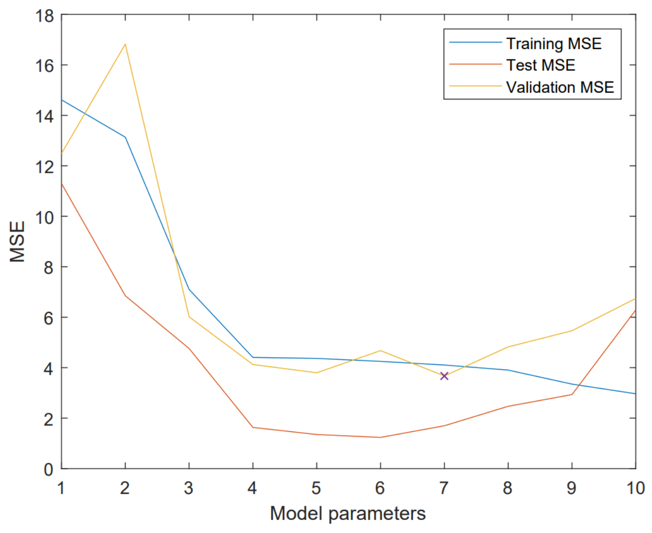


Figure C.3: Training error with three sets

APPENDIX D

Model selection

D.1 Introduction

When faced with persistent poor performance of a model despite exhaustive parameter tuning and cross-validation, two opposing strategies emerge: simplification or augmentation of complexity. To simplify a model, various techniques can be employed:

- *Feature selection*: this process entails selecting a subset of significant features while discarding irrelevant or redundant ones. Techniques such as filter methods, embedded methods, or wrapper methods like backward step-wise selection can be utilized.
- *Feature extraction*: by transforming features into another space, often of lower dimensionality, techniques such as Principal Component Analysis (PCA) or t-SNE can reduce data complexity while retaining essential characteristics.
- *Regularization* (shrinkage): techniques like Lasso and Ridge regularization introduce penalties for complex models within the loss function, discouraging overfitting and promoting simpler models.

D.2 Model selection

D.2.1 Filter method

In the absence of a predefined hypothesis space for models, the filter method for feature selection involves the following steps:

1. For each feature $j \in \{1, \ldots, M\}$, compute the Pearson correlation coefficient between x_k and the target variable y:

$$\hat{\rho}(x_j, y) = \frac{\sum_{n=1}^{N}(x_{j,n} - \bar{x}_j)(y_n - \bar{y})}{\sqrt{\sum_{n=1}^{N}(x_{j,n} - \bar{x}_j)^2}\sqrt{\sum_{n=1}^{N}(y_n - \bar{y})^2}}$$

Where:

$$\bar{x}_j = \frac{1}{N}\sum_{n=1}^{N} x_{j,n} \qquad \bar{y} = \frac{1}{N}\sum_{n=1}^{N} y_j$$

2. Select the features with higher Pearson correlation coefficient, which captures only linear relationships between features and the target variable.

For addressing non-linear relationships, alternative approaches such as mutual information can be employed.

D.2.2 Wrapper method

In the wrapper method for feature selection, where a hypothesis space of models $\mathcal{H}$ is provided as input, the following steps are undertaken:

1. For each k number of features $k \in \{1, \dots, M\}$ learn all possible $\binom{M}{k}$ models within $\mathcal{H}$ with k inputs.

2. Select the model with the smallest loss.

Choose the number of features M that provides the selected model with the smallest loss.

Application to Iris dataset Consider a classification problem where the goal is to discriminate between Virginica and Non-Virginica iris species. We select a performance index: validation accuracy on 20% of the data. Initially, we train a logistic regression model on the full dataset $(x_1, x_2, x_3, x_4)^T$ and observe the accuracies after removing individual features:

- Model with $(x_1, x_2, x_3)^T$: accuracy 1.
- Model with $(x_1, x_3, x_4)^T$: accuracy 1.
- Model with $(x_1, x_2, x_4)^T$: accuracy 1.
- Model with $(x_2, x_3, x_4)^T$: accuracy 1.

Removing a single feature doesn't affect the model's performance. Let's randomly remove x_4 and then another feature to check the error:

- Model with $(x_1, x_3)^T$: accuracy 0.96.
- Model with $(x_1, x_2)^T$: accuracy 0.96.
- Model with $(x_2, x_3)^T$: accuracy 1.

The model with features $(x_2, x_3)^T$ exhibits superior performance.

D.2.3 Principal Component Analysis

Principal Component Analysis (PCA) is an unsupervised dimensionality reduction technique used to extract low-dimensional features from a dataset.

It performs a linear transformation of the original data matrix $\mathbf{X}$, such that the largest variance is captured by the first transformed feature, the second largest variance by the second transformed feature, and so forth.

Finally, to reduce dimensionality, only a subset of the extracted features is retained.

Procedure The steps are:

1. Translate the original data matrix $\mathbf{X}$ to $\tilde{\mathbf{X}}$ such that they have zero mean.
2. Compute the covariance matrix of $\tilde{\mathbf{X}}$: $\mathbf{C} = \tilde{\mathbf{X}}^T\tilde{\mathbf{X}}$.
3. The eigenvectors of $\mathbf{C}$ are the principal components. The computation of the eigenvectors can be done with Singular Value Decomposition (SVD).
4. Given a sample vector $\tilde{\mathbf{x}}$, its transformed version $\mathbf{t}$ can be computed using:

   $$\mathbf{T} = \tilde{\mathbf{X}}\mathbf{W}$$

   Here:

   - Loadings: $\mathbf{W} = (\mathbf{e}_1|\mathbf{e}_2|\dots|\mathbf{e}_M)$ matrix of the principal components.
   - Scores: $\mathbf{W}$ transformation of the input dataset $\tilde{\mathbf{X}}$.
   - Variance: $(\lambda_1, \dots, \lambda_M)^T$ vector of the variance of principal components.
5. There are several methods to determine how many features to choose:

   - Keep all the principal components until we have a cumulative variance of 90%-95%:

     $$\text{cumulative variance with } k \text{ components} = \frac{\sum_{j=1}^{k} \lambda_i}{\sum_{j=1}^{M} \lambda_i}$$

   - Keep all the principal components which have more than 5% of variance (discard only those with low variance).
   - Find the elbow in the cumulative variance.

Purposes

- *Feature extraction*: reduce the dimensionality of the dataset by selecting only the number of principal components that retain information about the problem.
- *Compression*: retain the first k principal components and obtain $\mathbf{T}_k = \tilde{\mathbf{X}}\mathbf{W}_k$. The linear transformation $\mathbf{W}_k$ minimizes the reconstruction error:

  $$\min_{\mathbf{W}_k \in \mathbb{R}^{M\times k}} \left\| \mathbf{T}\mathbf{W}_k^T - \tilde{\mathbf{X}} \right\|_2^2$$

- *Data visualization*: reduce the dimensionality of the input dataset to 2 or 3 dimensions to facilitate visualization of the data.

D.2.4 Regularization

Regularization techniques such as ridge, lasso, and elastic net are well-established procedures used to mitigate overfitting in linear regression models. While originally developed for linear regression, these methods can also be extended to other machine learning algorithms.

For classification tasks, specific regularization methods tailored to the nature of the problem are employed.

D.3 Ensemble methods

Ensembling methods offer another avenue for enhancing predictive performance by combining multiple models. Two widely adopted ensembling techniques are bagging and boosting.

D.3.1 Bagging

The goal of bagging is to achieve a reduction in variance without significantly increasing bias. This is accomplished by training multiple learners, possibly in parallel:

1. Generate multiple datasets by applying random sampling with replacement (bootstrapping).

2. Train a model on each dataset.

To make predictions for new samples, apply all the trained models and combine their outputs using techniques such as majority voting (for classification) or averaging.

Bagging is generally effective in reducing variance, although the sampled datasets are not independent. It is particularly beneficial for unstable learners, which are models that exhibit significant changes with even small variations in the dataset, typically characterized by low bias and high variance in regression tasks.

D.3.2 Boosting

The goal of boosting is to achieve low bias by utilizing simple (weak) learners while simultaneously minimizing variance. This is accomplished by sequentially training weak learners:

1. Initially, assign equal weights to all samples in the training set.

2. Train a weak learner on the weighted training set.

3. Compute the error of the trained model on the weighted training set.

4. Increase the weights of samples misclassified by the model.

5. Repeat steps 2-4 until some predefined stopping criterion is met.

The ensemble of models learned through this process can be applied to new samples by computing the weighted prediction of each model, where more accurate models are given higher weights.

D.3.3 Comparison

Bagging:

- Reduces variance.

- Not suitable for stable learners.

- Applicable with noisy data.

- Generally provides assistance, though the impact may be modest.

- Executes in parallel.

Boosting:

- Reduces bias (typically without overfitting).
- Compatible with stable learners.
- May encounter issues with noisy data.
- Not always beneficial, but it can significantly improve performance.
- Operates sequentially.

APPENDIX E

Kernel methods

E.1 Introduction

In the case that the model you are considering is not performing well even after tuning its parameters properly through cross-validation, two opposite options emerge: simplify the model or increase its complexity. In the second scenario, one might approach the problem in a more complex space by introducing handcrafted features or examining the problem within the kernel space.

E.1.1 Kernel construction

To leverage kernel substitution, it's essential to have valid kernel functions:

1. Choose a feature space mapping $\phi(\mathbf{x})$ and use it to find the corresponding kernel:

   $$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}') = \sum_{i=1}^{M} \phi_i(\mathbf{x})\phi_i(\mathbf{x}')$$

   Here, $\phi(\mathbf{x})$ represents basis functions such as polynomials. For each i, we select $\phi_i(\mathbf{x}) = \mathbf{x}^i$ in the one-dimensional case.

2. Without explicitly constructing the function $\phi(\mathbf{x})$, a necessary and sufficient condition for a function $k(\mathbf{x}, \mathbf{x}')$ to be a kernel is the Gram matrix $\mathbf{K}$, where its elements are given by $k(\mathbf{x}_n, \mathbf{x}_m)$, to be positive semi-definite for all possible choices of the set $\{\mathbf{x}_n\}$ Positive semi-definite does not imply that the matrix's elements are non-negative; rather, it means $\mathbf{y}^T\mathbf{K}\mathbf{y} \geq 0$ for non-zero vectors $\mathbf{y}$ with real entries. In other words, for any real numbers $\{\mathbf{x}_n\}$ such that $\sum_n \sum_m K_{n,m y_n y_m \geq 0}$. New kernels can be constructed from simpler kernels as building blocks, thanks to Mercer's theorem.

E.2 Gaussian processes

A Gaussian process is defined as a probability distribution over functions $y(\mathbf{x}_i)$ where the values of $y(\mathbf{x}_i)$ evaluated at any set of points $\mathbf{x}_1, \ldots, \mathbf{x}_N$ jointly have a Gaussian distribution. This distribution is completely specified by the mean and the covariance:

- Usually, we do not have any prior information about the mean of $y(\mathbf{x})$, so we take it to be zero.
- The covariance is given by the kernel function:

$$\mathrm{Cov}[y(\mathbf{x}_i), y(\mathbf{x}_j)|\mathbf{x}_i, \mathbf{x}_j] = \mathbb{E}[y(\mathbf{x}_i)y(\mathbf{x}_j)|\mathbf{x}_i, \mathbf{x}_j] = K(\mathbf{x}_i, \mathbf{x}_j)$$

With this formulation, Gaussian Processes (GPs) are kernel methods that can be applied to solve regression problems.

The target is represented as $\mathbf{t} = y(\mathbf{x}) + \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ is a noise term independent of the point $\mathbf{x}$. The conditional distribution of the targets $\mathbf{t}_N = (t_1, \dots, t_N)^T$ of size N is given by:

$$p(\mathbf{t}_N|\mathbf{y}_N) = \mathcal{N}(\mathbf{t}_N|\mathbf{y}_N, \sigma^2\mathbf{I}_N)$$

Here, $\mathbf{y}_N = (y(\mathbf{x}_1), \dots, y(\mathbf{x}_N))^T$. The prior distribution of $p(\mathbf{y}_N) = \mathcal{N}(0, \mathbf{K}_N)$, where:

$$\mathbf{K}_N = \begin{bmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & \cdots & K(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ K(\mathbf{x}_N, \mathbf{x}_1) & \cdots & K(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}$$

Hence, the marginal distribution of the target is:

$$p(\mathbf{t}_N) = \int p(\mathbf{t}_N|\mathbf{y}_N)p(\mathbf{y})\, d\mathbf{y}_N = \mathcal{N}(\mathbf{t}_N|\mathbf{0}, \mathbf{C}_N)$$

Here, $\mathbf{C}_N = \mathbf{K}_N + \sigma^2\mathbf{I}_N$. Our objective is to predict the target $\mathbf{t}_{N+1}$ corresponding to a specific unseen input $\mathbf{x}_{N+1}$. From the definition, we have: $p(\mathbf{t}_{N+1}) = \mathcal{N}(\mathbf{t}_{N+1}|\mathbf{0}, \mathbf{C}_{N+1})$, Where:

$$\mathbf{C}_{N+1} = \begin{bmatrix} \mathbf{C}_N & \mathbf{k} \\ \mathbf{k}^T & c \end{bmatrix} \quad \mathbf{k} = (K(\mathbf{x}_1, \mathbf{x}_{N+1}), \dots, K(\mathbf{x}_N, \mathbf{x}_{N+1}))^T \quad c = K(\mathbf{x}_{N+1}, \mathbf{x}_{N+1}) + \sigma^2$$

We need to compute $p(\mathbf{t}_{N+1}|\mathbf{t}_N, \mathbf{x}_1, \dots, \mathbf{x}_N) = \mathcal{N}(m(\mathbf{x}_{N+1}), \sigma^2(\mathbf{x}_{N+1}))$, where:

- Mean: $m(\mathbf{x}_{N+1}) = \mathbf{k}^T\mathbf{C}_N^{-1}\mathbf{t}$
- Variance: $\sigma^2(\mathbf{x}_{N+1}) = c - \mathbf{k}^T\mathbf{C}_N^{-1}\mathbf{t}$

Python To model the relationship between petal length and width as a Gaussian Process (GP), follow these steps:

1. Load the data and normalize them.
2. Select the values of:
   - Noise variance $\sigma^2 = \mathrm{Var}[\varepsilon] = 0.2$
   - Constant $k = 1$
   - Length-scale $l = 0.8$

$$K(\mathbf{x}_i, \mathbf{x}_j) = ke^{-\frac{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|_2^2}{2l^2}}$$

3. Initialize a GP regression model (`GaussianProcessRegressor`).
4. Predict new values.

Hyperparameters Although Gaussian Processes (GPs) are non-parametric methods, the noise variance σ^2 and the parameters of the kernel have to be estimated or set. This can be done by:

- Utilizing a priori information on the problem being analyzed.
- Maximizing their log-likelihood on an independent dataset.
- Potentially refining them as new data are collected.

However, it's important to note a caveat: often, these hyperparameters are estimated using the same data used for prediction. This practice is not advisable in machine learning, as it is equivalent to overfitting.

E.3 Support Vector Machines

Support Vector Machines are a flexible and theoretically supported method initially applied to classification. Over the years, they have been extended to handle regression, clustering, and anomaly detection problems.

The fundamental idea behind SVM is to find the hyperplane that maximizes the margins (distance between the boundary and the points).

The hypothesis space of SVM is defined as $y(\mathbf{x}) = f(\mathbf{x}, \mathbf{w}) = \text{sign}(\mathbf{w}^T\mathbf{x} + b)$. The loss function is computed over the dataset $\mathcal{D} = \{(\mathbf{x}_n, t_n)\}_{n=1}^N$ with $t_n \in \{-1, 1\}$. It aims to minimize:

$$\frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{n=1}^{N}\zeta_i$$

with respect to $\mathbf{w}, \zeta_1, \dots, \zeta_N$ subject to:

$$\begin{cases} t_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 - \zeta_i & \forall n \in \{1, \dots, N\} \\ \zeta_i \geq 0 \quad \forall \in \{1, \dots, N\} \end{cases}$$

where $C > 0$ is a hyperparameter. The optimization method typically used for SVM is sequential quadratic optimization.

Python To train a linear classification Support Vector Machine:

1. Define the SVM: `SVM_model.svm.SVC(kernel='linear')`.
2. Train the SVM: `SVM_model.fit(input, target)`.

We are interested in determining:

- Boundary $\mathbf{w}^T\mathbf{x}_n + b = 0$.
- Margins $\mathbf{w}^T\mathbf{x}_n + b = \pm 1$.
- Support vectors (`SVM_model.support_vectors_`).

The use of kernels in SVMs is almost native, turning it into a non-parametric method:

- Hypothesis space: $y(\mathbf{x}) = f(\mathbf{x}, \mathbf{w}) = \text{sign}\left(\sum_{n=1}^{N}\alpha_n t_n K(\mathbf{x}_n, \mathbf{x}) + b\right)$.

- Loss measure: loss function in the dual formulation.
- Optimization method: quadratic optimization.

In Python:

- Define the SVM: `SVM_model = svm.SVC()`.
- Train the SVM: `SVM_model.fit(input, target)`.

With kernels, we no longer have an explicit formula for the boundary and the margins.

APPENDIX F

Learning theory

F.1 Introduction

Evaluating a model involves various techniques, including:

- Validation.
- Cross-validation/Leave-One-Out (LOO).
- Adjustment techniques.

F.1.1 Problem definition

Consider a supervised learning problem with:

- Input space $\mathcal{X}$.
- Output space $\mathcal{Y}$.
- Unknown joint probability $\Pr(\mathbf{x}, t)$ on $\mathcal{X} \times \mathcal{Y}$.
- Loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$.
- Hypothesis space $\mathcal{H} \subset \{h : \mathcal{X} \to \mathcal{Y}\}$.

We can minimize the loss function using:

- Population risk minimization: where we know $\Pr(\mathbf{x}, t)$ and minimize the true loss $\mathcal{L}$:

$$h^* \in \operatorname*{argmin}_{h \in \mathcal{H}} \mathcal{L}(h) = \mathbb{E}_{t,\mathbf{x}}[\ell(h(\mathbf{x}, t))]$$

- Empirical risk minimization: where we have a training dataset $\mathcal{D}_{train} = (\mathbf{x}n, t_n)^N n = 1$, independent and identically distributed random variables from Pr, and minimize the training loss $\hat{\mathcal{L}}$:

$$\hat{h} \in \operatorname*{argmin}_{h \in \mathcal{H}} \hat{\mathcal{L}}(h) = \frac{1}{N} \sum_{n=1}^{N} \ell(h(\mathbf{x}_n), t_n)$$

We need to evaluate the true loss of the empirical risk minimizer:

$$\mathcal{L}(\hat{h}) = \mathbb{E}_{t,\mathbf{x}}[\ell(\hat{h}(\mathbf{x}), t)|\hat{h}]$$

The true loss $\mathcal{L}(\hat{h})$ cannot be computed exactly without knowing $\Pr(\mathbf{x}, t)$. The training loss $\hat{\mathcal{L}}(\hat{h})$ is a negatively biased estimator for the true loss $\mathcal{L}(\hat{h})$. Thus, we seek a Probably Approximately Correct (PAC) result:

$$\mathcal{L}(\hat{h}) \leq \epsilon \text{ with probability } 1 - \delta$$

The value of ϵ is computed from data. We may encounter two cases:

- We have a test dataset $\mathcal{D}_{test}$.
- We only have the training dataset $\mathcal{D}_{train}$.

F.2 Test set

We possess a test set $\mathcal{D}_{test} = (\mathbf{x}_j, t_j)_{j=1}^{J}$ comprising independent and identically distributed random variable samples from Pr, completely separate from the training dataset $\mathcal{D}_{train}$. For any arbitrary hypothesis $h \in \mathcal{H}$, we can assess the test loss:

$$\tilde{\mathcal{L}}(h) = \frac{1}{J}\sum_{j=1}^{J} \ell(h(\mathbf{x}_j), t_j)$$

The empirical risk minimizer $\hat{h}$ remains independent of $\mathcal{D}_{test}$ (while dependent on $\mathcal{D}_{train}$): consequently, the test loss $\tilde{L}(\hat{h})$ stands as an unbiased estimator for the true loss $L(\hat{h})$.

Hoeffding inequality bound Suppose $X_1, \ldots, X_t$ are independent and identically distributed random variables with support in $[0, L]$, all having the identical mean $\mathbb{E}[X_i] =: X$, and let denote the sample mean as:

$$\bar{X}_t = \frac{\sum_{i=1}^{t} X_i}{t}$$

Then:

$$\Pr(X \leq \bar{X}_t + u) \geq 1 - e^{-\frac{2tu^2}{L^2}}$$

This implies an upper bound can be constructed with at least $1 - \delta$ confidence by setting $\delta = e^{-\frac{2tu^2}{L^2}}$. Consequently, the bound becomes:

$$X \leq \bar{X}_t + u = \bar{X}_t + L\sqrt{\frac{\log\left(\frac{1}{\delta}\right)}{2n}}$$

A crucial observation: all losses $\{\ell(\hat{h}(\mathbf{x}_j), t_j)\}_{j=1}^{J}$ are independent and identically distributed random variables conditioned to $\hat{h}$. $\tilde{\mathcal{L}}(\hat{h})$ can be interpreted as a sample mean of independent and identically distributed random variable samples, estimating the true mean $\mathcal{L}(\hat{h})$.

Under the assumption of bounded loss $\ell(y, y') \in [0, L]$, Hoeffding's inequality can be applied:

$$\mathcal{L}(\hat{h}) \leq \tilde{\mathcal{L}}(\hat{h}) + L\sqrt{\frac{\log\left(\frac{1}{\delta}\right)}{2J}}$$

With probability $1-\delta$. The larger the test set (J), the more precise the estimate $\tilde{\mathcal{L}}(\hat{h})$ becomes. Notably, there's no dependence on the complexity of the hypothesis space $\mathcal{H}$.

F.3 Training set

We can exclusively utilize the same training set $\mathcal{D}_{test} = (\mathbf{x}_n, t_n)_{n=1}^N$, comprising independent and identically distributed random variable samples from Pr, which was used to learn the empirical risk minimizer $\hat{h}$. For any arbitrary hypothesis $h \in \mathcal{H}$, we can compute the training loss:

$$\tilde{\mathcal{L}}(h) = \frac{1}{N}\sum_{n=1}^{N} \ell(h(\mathbf{x}_n), t_n)$$

The empirical risk minimizer $\hat{h}$ is reliant on $\mathcal{D}_{train}$ ($\hat{h}$ is derived from the same $\mathcal{D}_{train}$): the training loss $\hat{\mathcal{L}}(\hat{h})$ serves as a negatively biased estimator for the true loss $\mathcal{L}(\hat{h})$, thus:

$$\mathbb{E}[\hat{\mathcal{L}}(\hat{h})|\hat{h}] \leq \mathbb{E}[\hat{\mathcal{L}}(h^*)|\hat{h}] \leq \mathcal{L}(h^*) \leq \mathcal{L}(\hat{h})$$

A crucial observation: all losses $\{\ell(\hat{h}(\mathbf{x}_n), t_n)\}_{n=1}^N$ are not independent and identically distributed random variables conditioned to $\hat{h}$. Thus, Hoeffding's inequality cannot be applied.

A Statistical Learning Theory approach (Vapnik) provides:

$$\mathcal{L}(\hat{h}) = \hat{\mathcal{L}}(\hat{h}) + \mathcal{L}(\hat{h}) - \hat{\mathcal{L}}(\hat{h}) \leq \hat{\mathcal{L}}(\hat{h}) + \sup_{h\in\mathcal{H}} \left|\mathcal{L}(h) - \hat{\mathcal{L}}(h)\right|$$

Now, the problem is to provide bounds on $\sup_{h\in\mathcal{H}} \left|\mathcal{L}(h) - \hat{\mathcal{L}}(h)\right|$, termed uniform bounds. They depend on:

- The size of the training set N.
- The complexity of the hypothesis space $\mathcal{H}$.

For binary classification and $\mathcal{L}$ equals accuracy, considering:

- Finite hypothesis space ($|\mathcal{H}| < +\infty$) and consistent learning ($\hat{\mathcal{L}}(\hat{h}) = 0$ always):

  $$\mathcal{L}(\hat{h}) \leq \frac{\log|\mathcal{H}| + \log\left(\frac{1}{\delta}\right)}{N}$$

  With probability $1 - \delta$.

- Finite hypothesis space ($|\mathcal{H}| < +\infty$) and agnostic learning ($\hat{\mathcal{L}}(\hat{h}) > 0$ possibly):

  $$\mathcal{L}(\hat{h}) \leq \hat{\mathcal{L}}(\hat{h}) + \sqrt{\frac{\log|\mathcal{H}| + \log\left(\frac{1}{\delta}\right)}{2N}}$$

  With probability $1 - \delta$.

- Infinite hypothesis space ($|\mathcal{H}| = +\infty$) and agnostic learning ($\hat{\mathcal{L}}(\hat{h}) > 0$ possibly):

  $$\mathcal{L}(\hat{h}) \leq \hat{\mathcal{L}}(\hat{h}) + \sqrt{\frac{VC(\mathcal{H})\log\left(\frac{2eN}{VC(\mathcal{H})}\right) + \log\left(\frac{4}{\delta}\right)}{N}}$$

  With probability $1 - \delta$.

APPENDIX G

Markov Decision Processes

G.1 Introduction

Our objective is to capture the dynamics of a process and explore the ability to make choices among various actions in each scenario. We address two distinct but interconnected challenges:

- *Prediction*: given a predefined behavior (policy) for each scenario, our aim is to forecast the anticipated long-term reward from a given initial state.
- *Control*: our focus shifts to learning the optimal course of action to maximize the expected long-term reward derived from the underlying process.

G.2 Prediction

Given a policy, we aim to determine the value of each state. The agent's behavior is defined by a policy:

$$\pi : \mathcal{S} \to \Delta(\mathcal{A})$$

Once a specific policy $\pi(a|s)$ is chosen, P^π and R^π are defined as follows:

$$P^\pi(s'|s) = \sum_{a \in \mathcal{A}} \pi(a|s) P(s'|s,a) \qquad \dim(P^\pi) = |\mathcal{S}| \times |\mathcal{S}|$$

$$R^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) P(s,a) \qquad \dim(R^\pi) = |\mathcal{S}|$$

State values We employ the Bellman expectation equation:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}^\pi \left[\sum_{t=0}^{+\infty} \gamma^t R(s_t, a_t) | s_0 = 0 \right] \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^\pi(s') \right] \\ &= R^\pi(s) + \gamma \sum_{s' \in \mathcal{S}} P^\pi(s'|s) V^\pi(s') \end{aligned}$$

This equation can be represented in matrix form:

$$V^\pi = R^\pi + \gamma P^\pi V^\pi \qquad \dim(V^\pi) = |\mathcal{S}|$$

We can solve this equation in two ways:

- *Closed-form solution*: utilizing the Bellman expectation equation:

  $$V^\pi = (I - \gamma P^\pi)^{-1} R^\pi$$

  Since P^π is a stochastic matrix, the eigenvalues of $(I - \gamma P^\pi)$ are in the range $[1-\gamma, 1]$ for $\gamma \in [0, 1)$, ensuring invertibility. Inverting $(I - \gamma P^\pi)^{-1}$ has a computational complexity of $\mathcal{O}(|\mathcal{S}|^3)$ with a straightforward algorithm.

- *Recursive solution*: if matrix inversion is infeasible due to a large state space, we can use the recursive form of the Bellman expectation equation:

  $$V^\pi = R^\pi + \gamma P^\pi V^\pi$$

Policy evaluation By altering the policy, represented in matrix form as:

$$\pi(s|a) = \Pi(s, a|s) \qquad \dim(\Pi) = |\mathcal{S}| \times |\mathcal{S}|\,|\mathcal{A}|$$

We can compute the state values using different strategies based on the given problem.

G.3 Control

Policy selection Choosing a policy can be achieved through:

- *Brute force*: exhaustively enumerate all possible policies, evaluate their values, and select the one maximizing value. This method guarantees a deterministic optimal policy but requires evaluating $|\mathcal{A}|^{|\mathcal{S}|}$ policies.
- *Dynamic Programming*:
  - *Policy iteration*: iteratively evaluate the current policy and update it in the greedy direction.
  - *Value iteration*: repeatedly apply the Bellman optimality equation in its recursive form. However, solving the Bellman optimality equation in closed form is not possible due to the non-linearity introduced by the max operator:

    $$V^*(s) = \max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|a,s) V^*(s') \right\}$$

Policy iteration Iterate until convergence:

1. *Policy evaluation*: compute the value V^{π_k} of the current policy π_k.
2. *Policy improvement*: update the policy from π_k to π_{k+1} based on the newly estimated values (greedy improvement):

   $$\begin{aligned} \pi_{k+1}(s) &= \operatorname*{argmax}_{a \in \mathcal{A}} Q^{\pi_k}(s,a) \\ &= \operatorname*{argmax}_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|a,s) V^{\pi_k}(s') \right\} \qquad \forall s \in \mathcal{S} \end{aligned}$$

   Policy iteration guarantees convergence to π^* in a finite number of steps.

Value iteration Directly evaluate the optimal policy by computing $V^*(s)$. Repeatedly apply the Bellman optimality equation:

$$V_{k+1}(s) = \max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V_k(s') \right\}$$

Once $V^*(s)$ is obtained, recover the optimal policy, i.e., the greedy one with respect to $V^*(s)$. Value iteration asymptotically converges to $V^*(s)$.

APPENDIX H

Reinforcement Learning

APPENDIX I

Multi-armed Bandit