CSCI 6521
Advanced Machine Learning I

Supp. Material:
Regularization and Model Selection

Md Tamjidul Hoque

Regularization

- We can have more than one model to fit the data. Say, we have models from linear to polynomial of power or order 9. In this context we will discuss the followings:
    - underfitting and overfitting issues
    - how to take care of the overfitting problem to take the benefit of the higher order polynomials, as well as
    - how to select the best model.
- Next we investigate some simple cases.

A Simple Curve Fitting Target

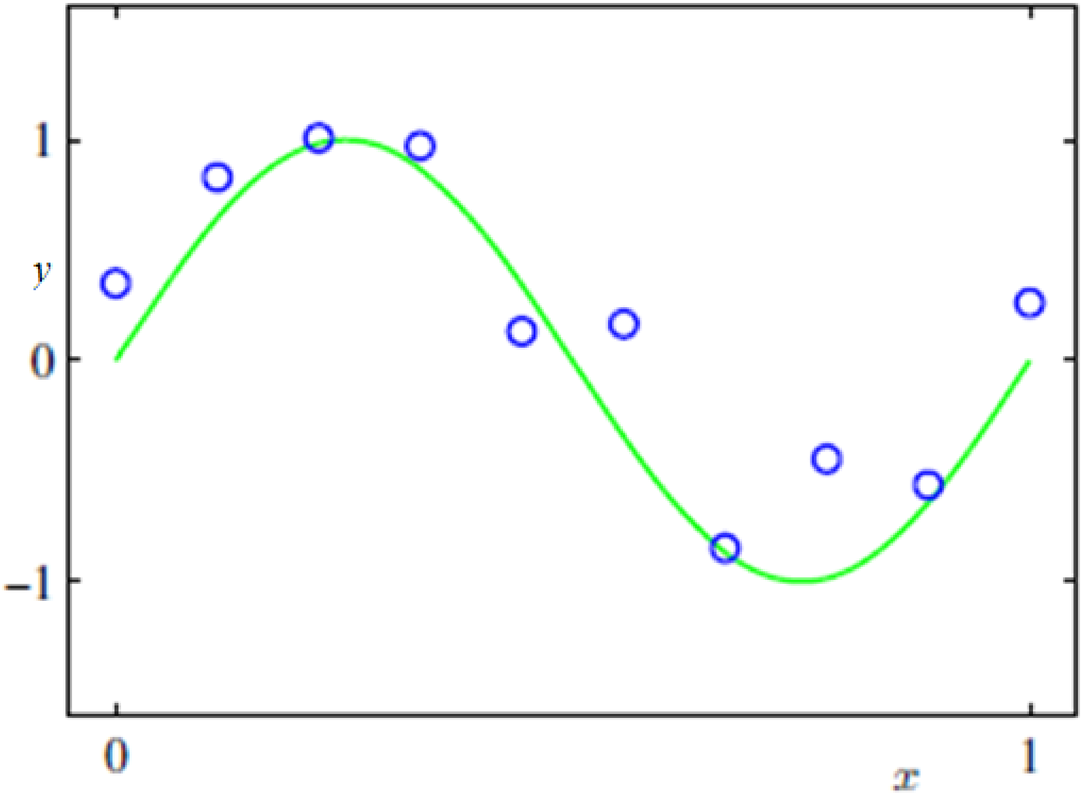
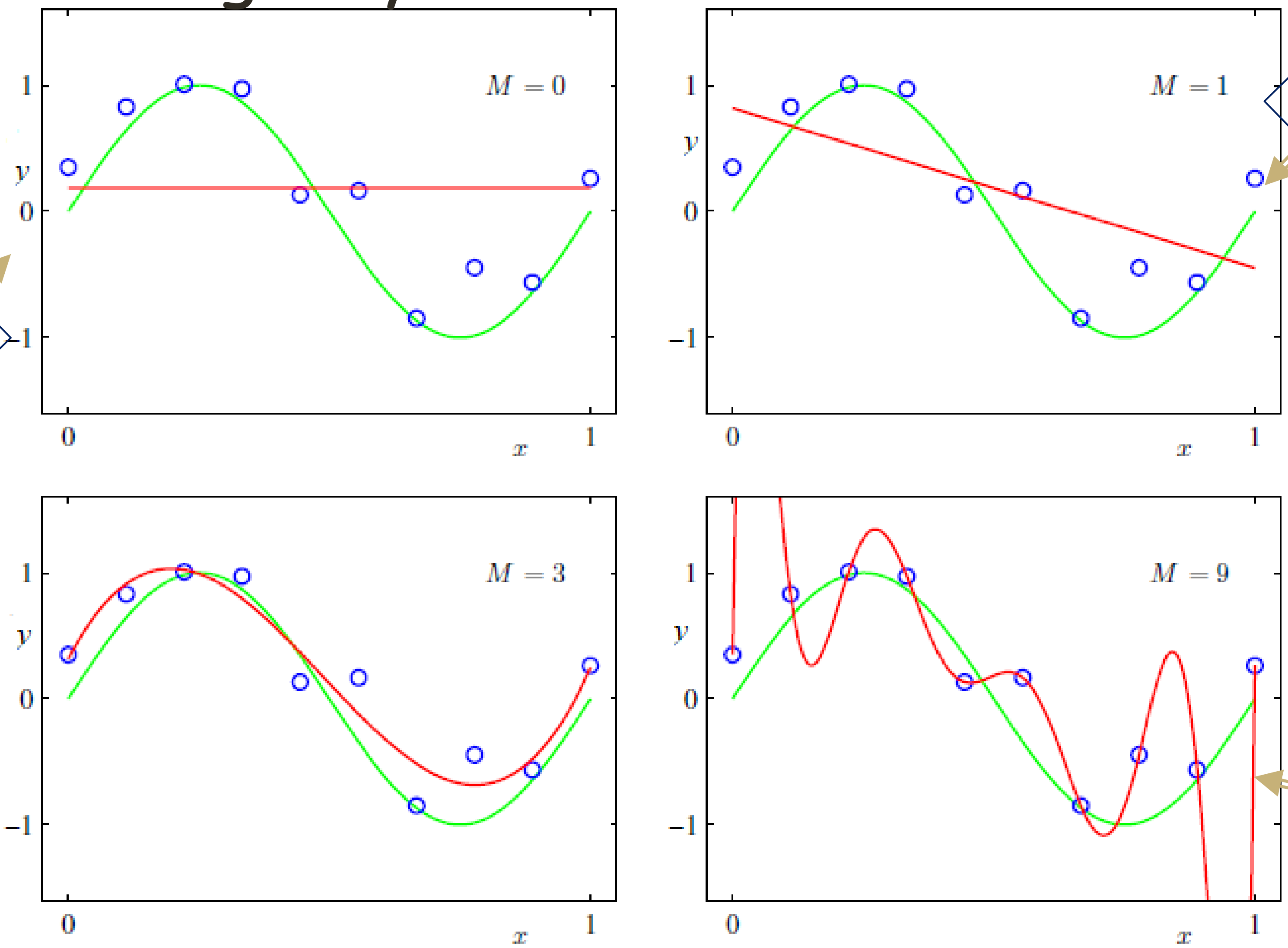


Figure 1: Plot of a training data set of $N = 10$ points, shown as blue circles, each comprising an observation of the input variable *x* along with the corresponding target variable *y*. The green curve shows the function $sin\,(2\pi x)$ used to generate the data. Our goal is to predict the value of *y*, for some new value of *x*, without knowledge of the green curve.

Fitting Polynomials of Various Order



Under fitting

Under fitting

Over fitting

Figure 2: Plots of polynomials having various orders M, shown as red curves, fitting to the data set shown here

Polynomial Fitting

- There remains the problem of choosing the order *M* of the polynomial, and as we shall see this will turn out to be an example of an important concept called ***model selection****.*

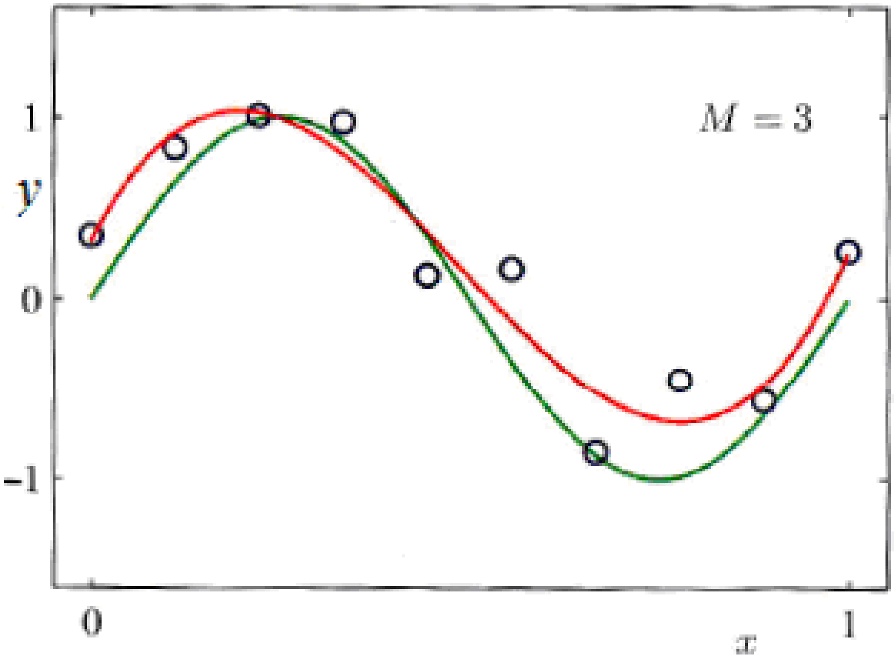
- In Figure 2, we show four examples of the results of fitting polynomials having orders *M* = 0, 1, 3 and 9 to the data set show in Figure 1.

- We give the general Equation as:

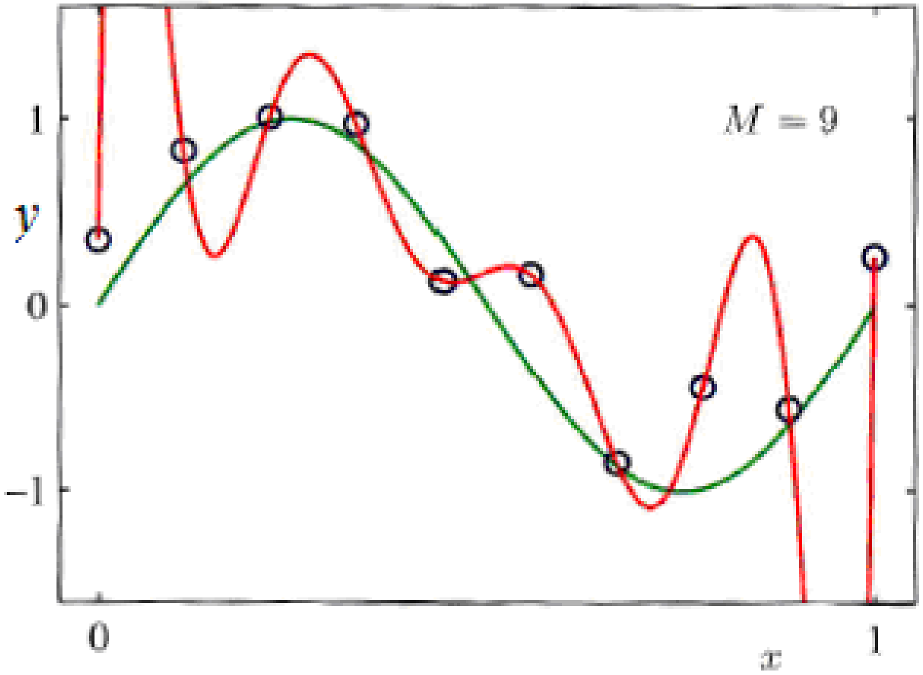
$$\hat{y}(x,\beta) = \beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_M x^M = \sum_{i=0}^{M} \beta_i x^i \qquad (1)$$

- We notice that the constant (M = 0) and first order (M = 1) polynomials give rather poor fits to the data and consequently rather poor presentations of the function *sin*(2 π x).

- The third order (M = 3) polynomial seems to give the best fit to the function *sin*(2 π x).



- When we got to a much higher order polynomial (*M* = 9), we obtain an **excellent fit** to the training data. In fact, the polynomial passes exactly through each data points and the *residual sum of squares* (RSS) must be 0.



- However, the fitted curve oscillates wildly and gives a very poor representation of the function *sin*(2 π x). This latter behavior is known as ***overfitting***.

Error Rate w.r.t Test Set

- Here, we define the root-mean-square (RMS) error by:

$$E_{RMS} = \sqrt{RSS(\beta) / N} \tag{2}$$

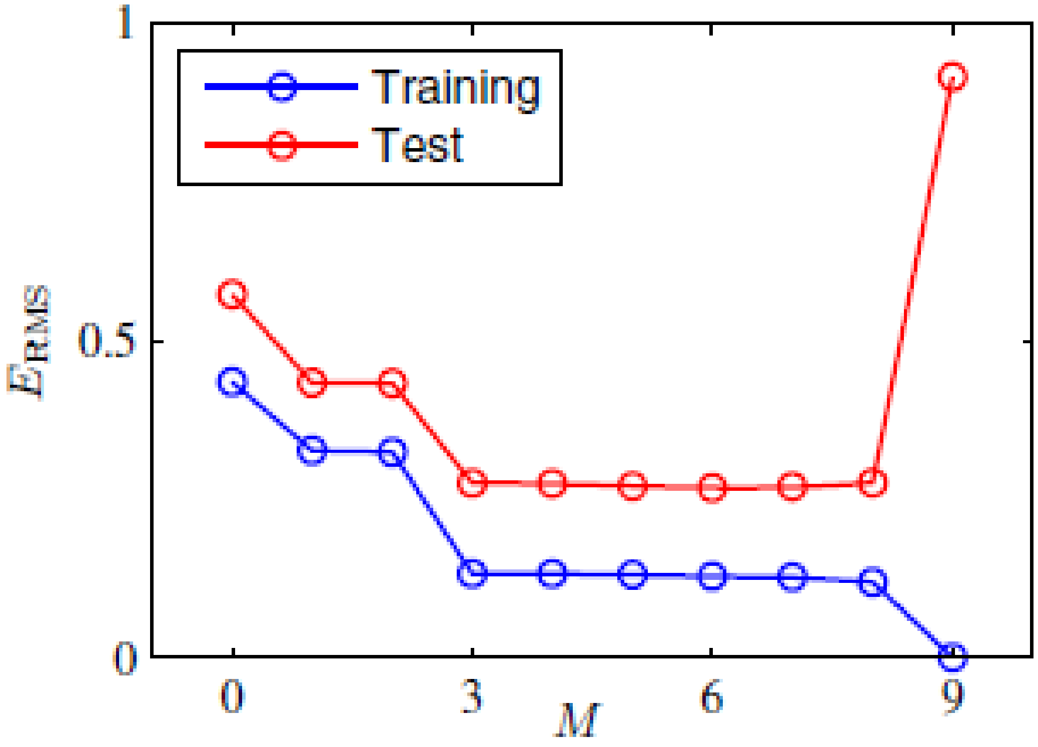
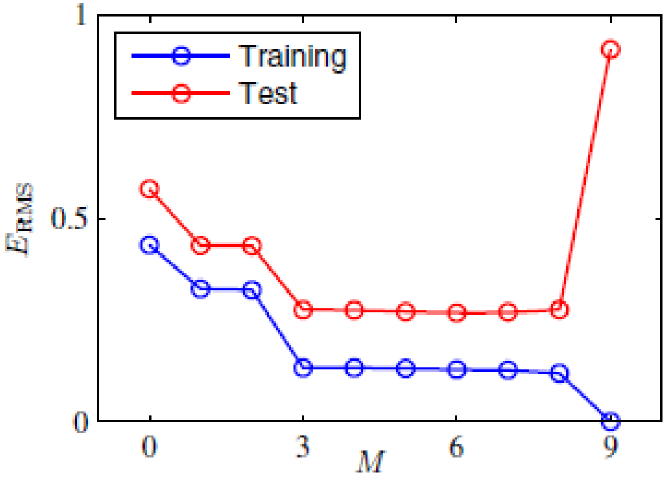
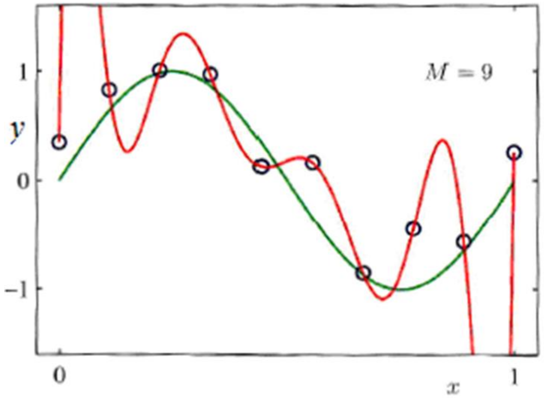


Figure 3: Graphs of the root-mean-square error, defined by (Eq^n 2), evaluated on the training set and on an independent test set for various values of M.



- For M = 9, the training set error goes to zero, as we might expect because this polynomial contains 10 degrees of freedom.

- However, the test set error has become very large.



← function (with M = 9) exhibits wild oscillations as we saw before.

Paradox!

- This may seem paradoxical because a polynomial of given order contains all lower order polynomials as special cases.

- The *M* = 9 polynomial is therefore capable of generating results at least as good as the *M* = 3 polynomial.

- We know that a power series expansion of the function *sin*(2 π x) contains terms of all orders, so we might expect that results should improve monotonically as we increase *M*.

- Let us investigate the status of the coefficients before solving the paradox.

Table 1: Table of the coefficients β for polynomials of various order. Observe how the typical magnitude of the coefficients increases dramatically as the order of the polynomial increases.

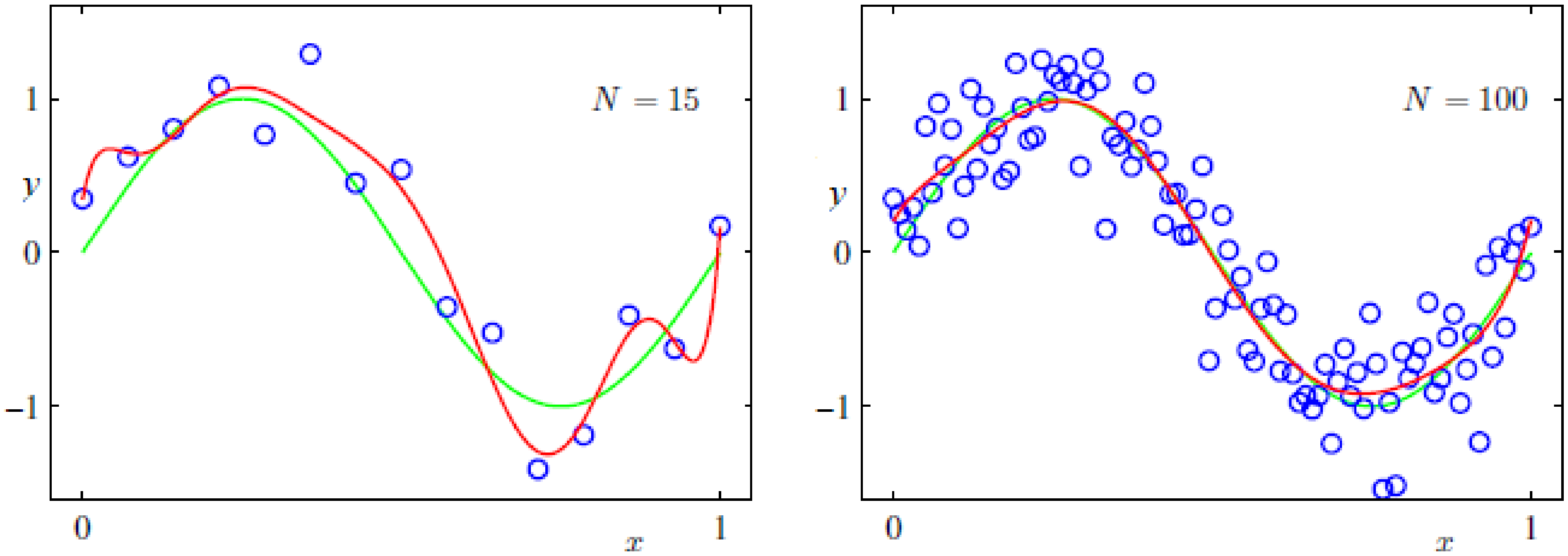
	$M = 0$	$M = 1$	$M = 3$	$M = 9$
β_0	0.19	0.82	0.31	0.35
β_1		-1.27	7.99	232.37
β_2			-25.43	-5321.83
β_3			17.37	48568.31
β_4				-231639.30
β_5				640042.26
β_6				-1061800.52
β_7				1042400.18
β_8				-557682.99
β_9				125201.43

- We see that, as *M* increases, the magnitudes of the coefficients typically get larger.

- In particular for *M* = 9 polynomial, the coefficients have become finely tuned to the data by developing large positive and negative values so that the corresponding polynomial function matches each of the data points exactly, but between data points (particularly near the ends of the range) the function exhibits the large oscillations (observed).

- Intuitively, what is happening is that the more flexible polynomials with larger values of *M* are becoming increasingly tuned to the random noise on the target values.

More Data for M=9



- It is also interesting to examine the behavior of a given model as the size of the data set is varied

- We see that, for a given model complexity, the overfltting problem become less severe as the size of the data set increases

Managing Overfitting for limited sized Dataset

- In many experiments, data is not found enough.

- Therefore, we need to control the coefficients /parameters to get the fitting advantage of higher order polynomial as well as to fit the equation more correctly.

- One technique that is often used to control the overfitting phenomenon in such cases is that of ***regularization***, which involves adding a penalty term to the error function:

$$RSS(\beta) = \sum_{i=1}^{N} \{(\hat{y}(x_i, \beta) - y_i)^2\} + \lambda \sum_{j=1}^{p} \beta_j^2 \qquad (3)$$

Note that often the coefficient β_0 is omitted from the regularizer because its inclusion causes the results to depend on the choice of origin for the target variable, or it may be included but with its own regularization coefficient.

Adjusting the Regularizer (M=9)

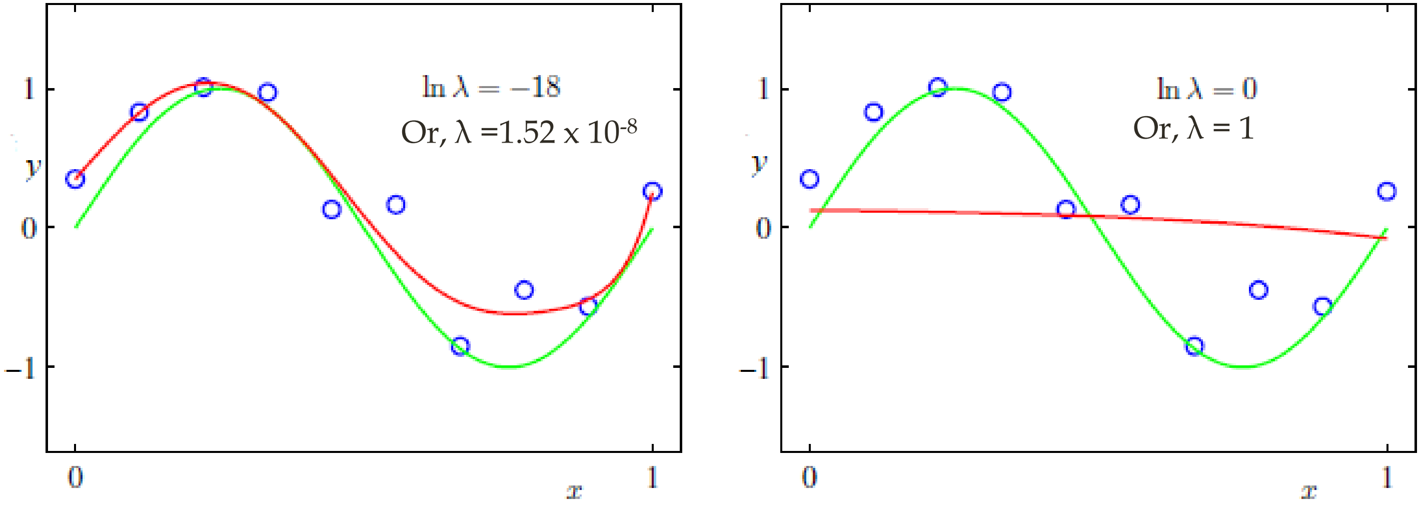
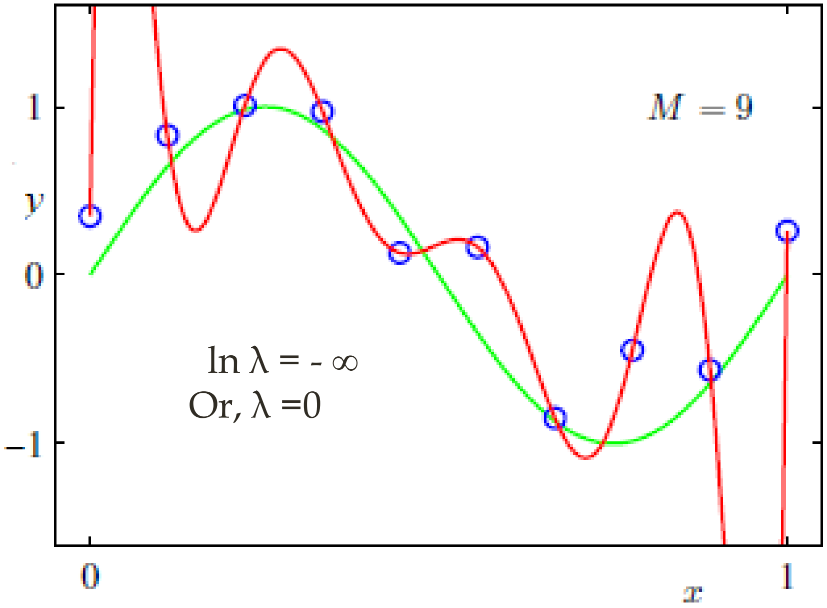


Fig 5: Two values of the regularization parameter λ corresponding to $\ln\lambda$ = -18 and $\ln\lambda$ =0.



Note: ln λ = -18 => λ = e ^ (-18) =1.52 × 10^-8
ln λ = 0 => λ = e^0 = 1

← The case of no regularizer, i.e. λ =0, corresponding to $\ln\lambda = -\infty$

Coefficients Table, For M=9

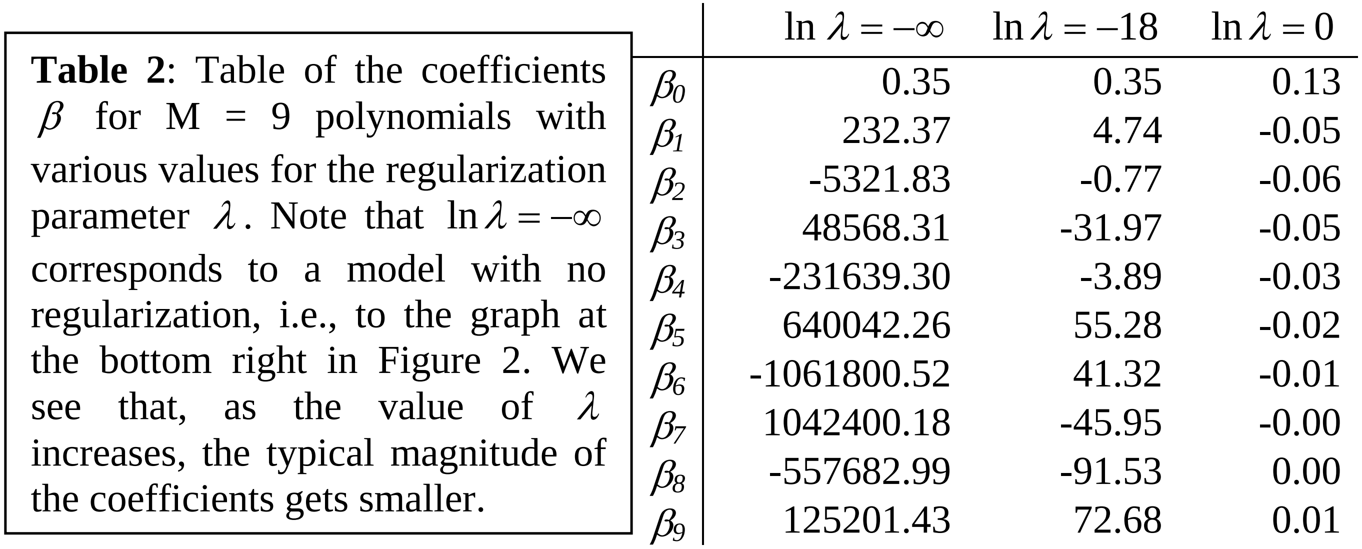


Table 2: Table of the coefficients β for M = 9 polynomials with various values for the regularization parameter λ. Note that $\ln\lambda = -\infty$ corresponds to a model with no regularization, i.e., to the graph at the bottom right in Figure 2. We see that, as the value of λ increases, the typical magnitude of the coefficients gets smaller.

	$\ln\lambda = -\infty$	$\ln\lambda = -18$	$\ln\lambda = 0$
β_0	0.35	0.35	0.13
β_1	232.37	4.74	-0.05
β_2	-5321.83	-0.77	-0.06
β_3	48568.31	-31.97	-0.05
β_4	-231639.30	-3.89	-0.03
β_5	640042.26	55.28	-0.02
β_6	-1061800.52	41.32	-0.01
β_7	1042400.18	-45.95	-0.00
β_8	-557682.99	-91.53	0.00
β_9	125201.43	72.68	0.01

Note: ln λ = -Inf => λ = e ^ (-Inf) = 0.0000000000 … => 0
ln λ = -18 => λ = e ^ (-18) =1.52 × 10^-8
ln λ = 0 => λ = e^0 = 1

- We see that, for a value of ln λ = -18, the overfltting has been suppressed and we now obtain a much closer representation of the underlying function *sin*(2πx). If, however, we use too large a value for λ then we again obtain a poor fit, as shown in Figure 5 for ln λ =0.

The Effects of the values of λ

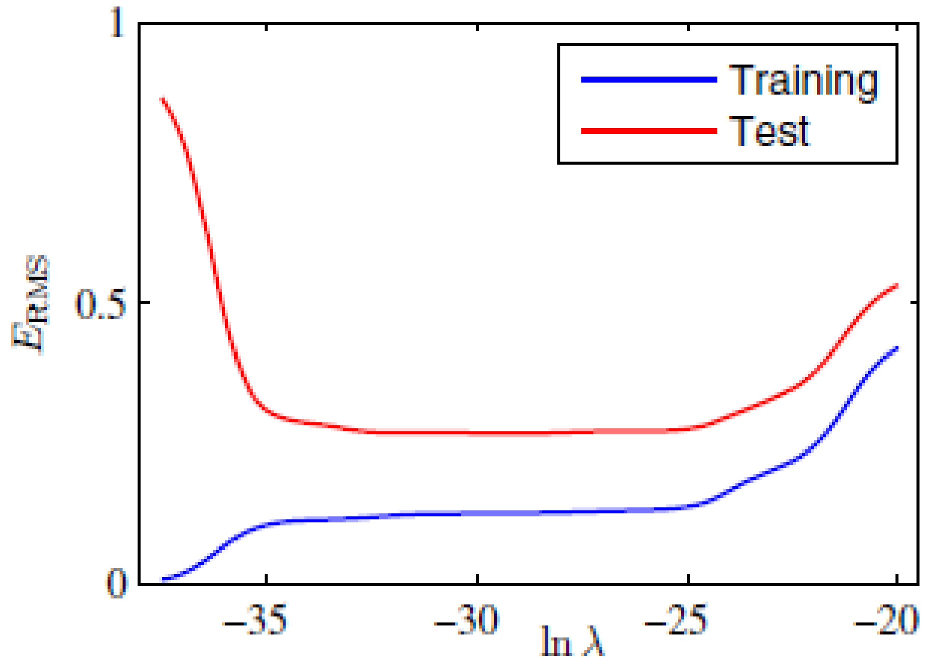


Figure 6: Graph of the root-mean-square error (Eq^n 2) versus **ln λ** for the $M = 9$ polynomial.

- The issue of model complexity is an important one and here we simply note that, if we were trying to solve a practical application using this approach of minimizing an error function, we would have to find a way to determine a suitable value for the model complexity.

- To solve, we can take the available data and partitioning it into a training set, used to determine the coefficients, and a separate validation set, also called a ***hold-out set***, used to optimize the model complexity (either *M* or λ).

Minimization Target and Eq^n

Objective: $$\underset{min}{RSS_\lambda(\beta)} = \sum_{i=1}^{N} \{(\hat{y}(x_i, \beta) - y_i)^2\} + \lambda \sum_{j=1}^{p} \beta_j^2$$

- To have minimum error, the $\lambda\sum_{j=1}^{p}\beta_j^2$ term has to adjust such a way that if we set the value λ too large then the values of β will be too low and other way around.

- The Minimization Eqn the we drove before can be like (next):

Minimization Equations

Newton's Method:

From
$$\beta_j(t+1)=\beta_j(t)-\frac{\sum_{i=1}^{N}\left[\left(x^T(i)\,\beta-y(i)\right).\,x(i)_j\right]}{\sum_{i=1}^{N}\left[x(i)_j.x(i)_j\right]}$$

To
$$\boxed{\beta_j(t+1)=\beta_j(t)+\left[\frac{\sum_{i=1}^{N}\left(y(i)-x^T(i)\,\beta\right).x(i)_j-\lambda\,\beta(t)_j}{\sum_{i=1}^{N}\left[x(i)_j.x(i)_j\right]+\lambda}\right]}$$

Gradient Descent:

From
$$\beta_j(t+1)=\beta_j(t)+\frac{2\alpha}{N}\sum_{i=1}^{N}\left(y(i)\;-x^T(i)\,\beta\right).\,x(i)_j$$

To
$$\beta_j(t+1)=\beta_j(t)+\frac{2\alpha}{N}\left[\sum_{i=1}^{N}\left(y(i)\;-x^T(i)\,\beta\right).\,x(i)_j-\lambda\beta(t)_j\right]$$

Or,
$$\boxed{\beta_j(t+1)=\beta_j(t)(1-\frac{2\alpha\lambda}{N})+\frac{2\alpha}{N}\left[\sum_{i=1}^{N}\left(y(i)\;-x^T(i)\,\beta\right).\,x(i)_j\right]}$$

Comparing the previous gradient descent (without regularization), we see the $\beta_j(t)$ is actually shrinking due to the term $(1-\frac{2\alpha\lambda}{N})$ which is < 1 as α,λ and N are positive quantities.

Here is how: (…Minimization Equations)

$$\frac{\partial}{\partial \beta_j} RSS_\lambda(\beta) = \frac{\partial}{\partial \beta_j} \sum_{i=1}^{N} \left[\left(y(i) - x^T(i)\ \beta \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right]$$

$$= \sum_{i=i}^{N} \left[2\left(y(i) - x^T(i)\ \beta \right).\ \frac{\partial}{\partial \beta_j} \left(y(i) - x^T(i)\ \beta \right) + \lambda \frac{\partial}{\partial \beta_j} (\beta_1^2 + \beta_2^2 + \ldots + \beta_j^2 + \ldots) \right]$$

$$= \sum_{i=1}^{N} \left[2\left(y(i) - x^T(i)\ \beta \right).\ \frac{\partial}{\partial \beta_j} \left(y(i)\ \ - x(i)_0 \beta_0 - x(i)_1 \beta_1 - \ldots - x(i)_j \beta_j - \ldots \right) + 2\lambda\beta_j \right]$$

$$= 2\sum_{i=1}^{N} \left[\left(x^T(i)\ \beta - y(i) \right).\ x(i)_j + \lambda\beta_j \right]$$

And, $\frac{\partial}{\partial \beta_j} \left(\frac{\partial}{\partial \beta_j} RSS_\lambda(\beta) \right) = 2 \frac{\partial}{\partial \beta_j} \left(\sum_{i=1}^{N} \left[\left(x^T(i)\ \beta - y(i) \right).\ x(i)_j + \lambda\beta_j \right] \right)$

$$= 2 \frac{\partial}{\partial \beta_j} \sum_{i=1}^{N} \left[\left(\left(x(i)_0 \beta_0 + x(i)_1 \beta_1 + \ldots + x(i)_j \beta_j + \ldots - y(i) \right).\ x(i)_j \right) + \lambda\beta_j \right]$$

… Here is how (Minimization Equations)

$$= 2\sum_{i=1}^{N}\Big[(x(i)_j \,.\, x(i)_j) + \lambda\Big]$$

Since, Newton's method: $$\beta_j(t+1) = \beta_j(t) - \frac{\dfrac{\partial}{\partial \beta_j} RSS_\lambda(\beta)}{\dfrac{\partial}{\partial \beta_j}\left(\dfrac{\partial}{\partial \beta_j} RSS_\lambda(\beta)\right)}$$

$$RHS = \beta_j(t) - \frac{2\sum_{i=1}^{N}\Big[\big(x^T(i)\,\beta - y(i)\big).\,x(i)_j + \lambda\beta_j\Big]}{2\sum_{i=1}^{N}\Big[(x(i)_j \,.\, x(i)_j) + \lambda\Big]} = \beta_j(t) - \frac{\sum_{i=1}^{N}\big(x^T(i)\,\beta - y(i)\big).\,x(i)_j + \lambda\beta_j}{\sum_{i=1}^{N}(x(i)_j \,.\, x(i)_j) + \lambda}$$

Finally, $$\beta_j(t+1) = \beta_j(t) + \left[\frac{\sum_{i=1}^{N}\big(y(i) - x^T(i)\,\beta\big).x(i)_j - \lambda\,\beta(t)_j}{\sum_{i=1}^{N}\Big[x(i)_j . x(i)_j\Big] + \lambda}\right]$$

Note: The minus signs inside and outside of the square bracket are changed just to look better.

… Here is how (Minimization Equations)

Similarly, for **gradient descent** method: $\beta_j(t+1) = \beta_j(t) - \alpha \dfrac{\partial}{\partial \beta_j} RSS_\lambda(\beta)$, we can have:

$$= \beta_j(t) - 2\alpha \sum_{i=1}^{N} \left[\left(x^T(i)\,\beta - y(i) \right).\, x(i)_j + \lambda \beta_j \right]$$

$$\boxed{\text{Finally, } \beta_j(t+1) = \beta_j(t) + \frac{2\alpha}{N} \left[\sum_{i=1}^{N} \left(y(i) \; - x^T(i)\,\beta \right).\, x(i)_j - \lambda \beta(t)_j \right]}$$

Normal Equation

- Recall: here is our normal equation derived before

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\ \mathbf{X}^T\ \mathbf{y}$$

To involve regularization, we rewrite the equation:

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X} + \lambda M_\lambda)^{-1}\ \mathbf{X}^T\ \mathbf{y}$$

Here, M_λ is a (p+1) by (p+1) matrix with all the elements assigned to ‘0’, except the diagonal elements. The diagonal elements are assigned to ‘1’. The only exception is that the first diagonal element, i.e., $M_\lambda(1,1) = 0$.
The is done to keep β_0 away from the influence of the regularization.

Octave code:

```
            M_lamda=eye(3,3);
            M_lamda(1,1)=0;
```

How do we use Normal Eq. for higher order polynomial?

- Recall our housing example and data. Let us assume that we want to involve features: (1) Number of Beds (x_1) and (2) Living Areas (x_2) and we want to fit a second order polynomial. Our equation should look like:

$$\hat{Y} = \beta_0 x_0 + \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_2 + \beta_4 x_2^2$$

And, our matrix **X** of equation $\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\,\mathbf{y}$ or, $\hat{\beta} = (\mathbf{X}^T\mathbf{X} + \lambda M_\lambda)^{-1}\,\mathbf{X}^T\,\mathbf{y}$ will look like, for example:

$$\mathrm{X} = \begin{array}{ccccc} \underline{x_0 = 1} & \underline{x_1} & \underline{x_1^2} & \underline{x_2} & \underline{x_2^2} \\ \end{array}$$

$$\mathrm{X} = \begin{pmatrix} 1 & 3 & 9 & 2000 & 4000000 \\ 1 & 2 & 4 & 1500 & 2250000 \\ 1 & 1 & 1 & 1000 & 1000000 \\ 1 & 4 & 16 & 1200 & 1440000 \\ \dots & \dots & \dots & \dots & \dots \end{pmatrix}_{N \times (p+1)}$$

Here, in X, p = 4 (not 2).

Here, we use the developed linear model to fit nonlinear data by simply adding powers of feature as new features, and then train a linear model on this extended set of features. This technique is called ***Polynomial Regression***.

… How do we use Normal Eq. for higher order polynomial?

Practically, we can build the column x_1^2 and x_2^2 from x_1 and x_2 in excel or, once we load the matrix we can manipulate the values in Octave/MATLAB.

For example, if you have x_1 loaded (e.g., by x1=X(:,2) MATLAB), you can generate x_1^2 by x1.^2.

For the above example assume you have loaded x1 and x2 variables as column vectors, then:

X=[ones (N,1), x1, x1.^2, x2, x2.^2], will form the **X** matrix.

We can use Scikit-Learn’s **PolynomialFeatures** class to transform our training data, adding the square of each feature in the training set as a new feature.

Model Selection

- Question: Given a finite set of models, **M** = {M_1, M_2, …, M_n}, for a problem, how can we select the best model? Here, assume, M_i is the i^{th} order polynomial regression model.
  - **Answer**: We assess model quality by cross-validation.
- **Cross-validation:** Here, we split the dataset into training and test datasets.
- We train the model using training dataset and test the model performance using test-dataset.

We discuss cross-validation algorithms next.

… Model Selection

- **Algorithm: hold-out cross validation**

  1. Randomly split data (D): D_{train} and $D_{holdout}$ (Say, 25-30% of the data).

  2. For $\forall i$, train M_i using D_{train} and get the corresponding P_i (i^{th} predictor).

  3. For $\forall i$, compute the error E_i of P_i using $D_{holdout}$.

  4. Pick the predictor where the error is the lowest.

The disadvantage of the hold-out method is the wastage of the data, as we typically hold out 30% data to get the error reliably.

Also, in many experiment the data is not abundant.

Further, we have seen that for higher order polynomial, it is better to feed more data to get a reliable predictor.

To avoid these issues, we follow k-fold cross validation where we hold out less data.

… Model Selection

➢ Algorithm: k-fold cross validation

```
1. Randomly split data (D) into $k$ disjoint subsets as: $D_1, D_2, ..., D_k$ .
2. For $\forall i$ , $M_i$ is evaluated as:
        For $j$ =1 to k:
                Train the $M_i$ using data: $(D - D_j)$ and Get predictor $P_{ij}$
                Test $P_{ij}$ using $D_j$ and get the error $E_{ij}$.
        EndFor $j$

        $E_i$ = average of $E_{ij}$ for $\forall j$ , which is the generalized error of $M_i$.
   EndFor $i$
3. Pick the best model $M_i$ having lowest generalized error $E_i$.
4. Re-train $M_{i\,=\,\text{best}}$ using full dataset D.
```

A typical choice of the value of k in such k-fold cross validation is 10.

We prefer to choose k to make the $1/k$ portion of the data to be a smaller fraction.

This method is also called *leave-one-out cross* validation.

How to obtain the best value of regularization parameter, λ?

1. Take a range of values with regular interval: {0, …, R} of real or integer numbers for λ.

2. Evaluate the performance for each of the value of λ using cross validation approach for example.

3. Pick the best value of λ for which the error was lowest.

How did we obtain, $\hat{\beta} = (\mathbf{X}^T\mathbf{X} + \lambda M_\lambda)^{-1}\, \mathbf{X}^T\, \mathbf{y}$?

Recall, our target involving regularization was: $\underset{\min}{RSS_\lambda(\beta)} = \sum_{i=1}^{N}\left[\left(\hat{y}(x_i,\beta) - y_i\right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2\right]$

In vector form we can rewrite the term: $\mathrm{RSS}(\beta) = (y - X\beta)^T(y - X\beta) + \lambda\beta^T\beta$

$$\mathrm{RSS}(\beta) = (y^T - \beta^T X^T)(y - X\beta) + \lambda\beta^T\beta \quad [\because (\mathrm{A} \pm \mathrm{B})^{\mathrm{T}} = \mathrm{A}^{\mathrm{T}} \pm \mathrm{B}^{\mathrm{T}}, (\mathrm{AB})^{\mathrm{T}} = \mathrm{B}^{\mathrm{T}}\mathrm{A}^{\mathrm{T}}]$$

$$= y^T y - \beta^T X^T y - y^T X\beta + \beta^T X^T X\beta + \lambda\beta^T\beta \qquad [\because a^T b = b^T a, \therefore y^T X\beta = (X\beta)^T y = \beta^T X^T y]$$

$$= y^T y - \beta^T X^T y - \beta^T X^T y + \beta^T X^T X\beta + \lambda\beta^T\beta$$

$$= y^T y - 2\beta^T X^T y + \beta^T X^T X\beta + \lambda\beta^T\beta$$

Now, differentiating $\mathrm{RSS}(\beta) = y^T y - 2\beta^T X^T y + \beta^T X^T X\beta + \lambda\beta^T\beta$ with respect to β and equating it to zero, we get:

$$\frac{\partial}{\partial\beta}(y^T y - 2\beta^T X^T y + \beta^T X^T X\beta + \lambda\beta^T\beta) = 0$$

… How did we obtain, $\hat{\beta} = (\mathbf{X}^T \mathbf{X} + \lambda M_\lambda)^{-1} \mathbf{X}^T \mathbf{y}$?

$$=> 0 - 2X^T y + \frac{\partial}{\partial \beta}(\beta^T) X^T X\beta + \beta^T X^T X \frac{\partial}{\partial \beta}(\beta) + \frac{\partial}{\partial \beta}(\lambda \beta^T I \beta) = 0$$

[Here, M_λ is the indetify martix, and we write $\lambda\beta^T \beta => \lambda\beta^T \mathrm{M}_\lambda \beta$]

$$=> 0 - 2X^T y + X^T X\beta + \beta^T X^T X + \lambda \mathrm{M}_\lambda \beta + \lambda \beta^T \mathrm{M}_\lambda = 0$$

$$[\because \beta^T X^T X = \beta^T (X^T X) = (X^T X)^T \beta = X^T X \beta)]$$

$$=> -2X^T y + 2X^T X\beta + 2\lambda \mathrm{M}_\lambda \beta = 0 \quad [\because \beta^T \mathrm{M}_\lambda = \mathrm{M}_\lambda{}^{\mathrm{T}} \beta = \mathrm{M}_\lambda \beta]$$

$$=> -2X^T y + 2\beta(X^T X + \lambda \mathrm{M}_\lambda) = 0$$

$$=> X^T y = \beta(X^T X + \lambda\, \mathrm{M}_\lambda)$$

$$=> \beta = (X^T X + \lambda\ \mathrm{M}_\lambda)^{-1} X^T y$$ [Proved]

Here, further in the identity matrix M_λ, we assign $\mathrm{M}_\lambda(1,1) = 0$ to keep β_{o} out of the regularization effect.

Variation of the Regularization Term (1)

- We can rewrite RSS (β) in terms of more general regularizer as:

$$RSS(\beta) = \sum_{i=1}^{N} \{(\hat{y}(x_i, \beta) - y_i)^2\} + \lambda \sum_{j=1}^{p} |\beta_j|^q \qquad (8)$$

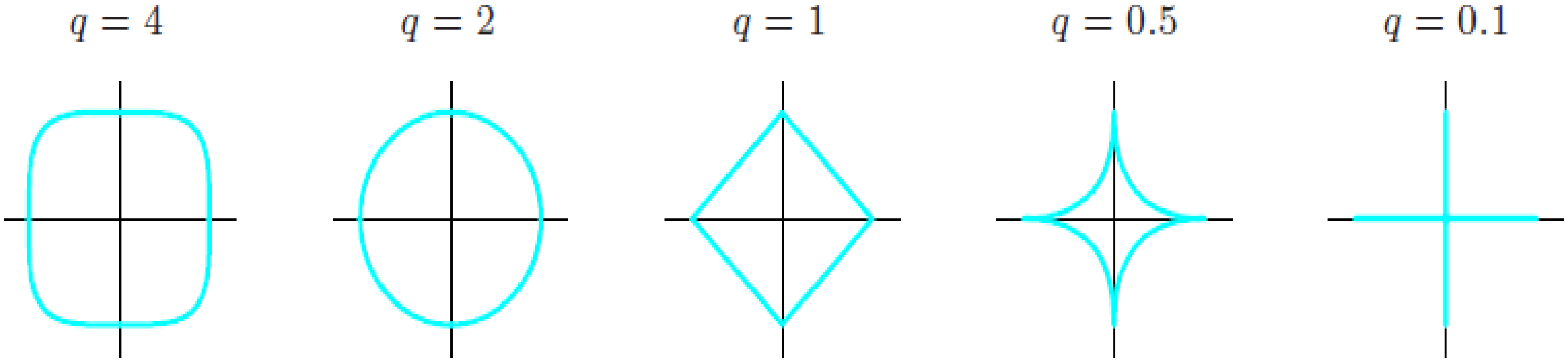


Figure: Contours of the regularization term in (8) for various values of the parameter *q*.

Variation of the Regularization Term (2)

q = 2:

is a quadratic regularizer, known as:
- ridge regression,
- L_2 regularization,
- Tikhonov regularization.

- In the context of neural networks, this approach is called **weight decay**.
- The ridge solutions are not equivariant under scaling of the inputs, and so one normally standardizes the inputs before solving.

q = 1:

is known as:
- L_1 regularization,
- lasso (least absolute shrinkage and selection operator),
- basis pursuit.

Variation of the Regularization Term (3)

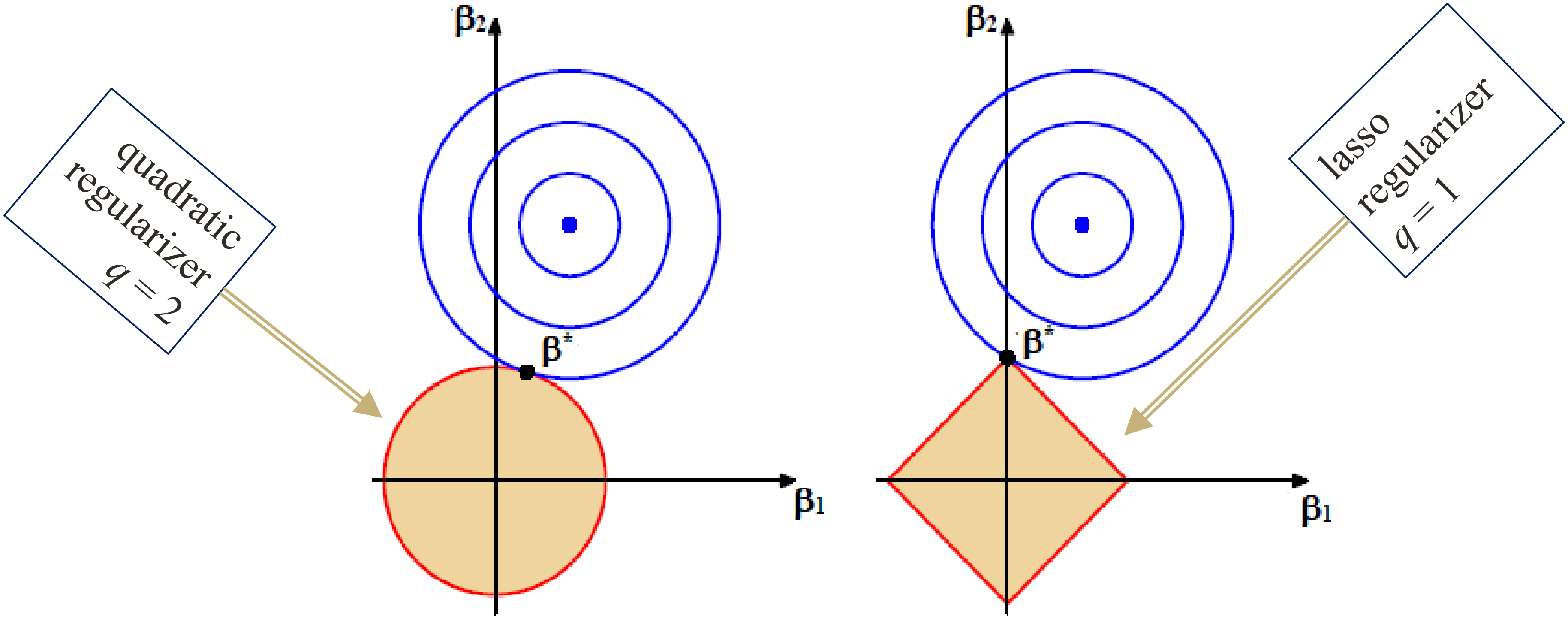


Figure: Plot of the contours of the unregularized error function (blue) along with the constraint region (solid area) for the quadratic regularizer q = 2 on the left and the lasso regularizer q = 1 on the right, in which the optimum value for the parameter vector β is denoted by β*. The lasso gives a sparse solution in which $\beta_1^* = 0$.

Variation of the Regularization Term (3)

For further reading, you may want to see (check Canvas for additional info):

- **Elastic Net** regularization (which linearly combines L_1 and L_2),
- **Group** lasso,
- **Fused** lasso,
- **Adaptive** lasso

Some Basic Concepts in Machine Learning (3)

Matrix completion:

- Sometimes we have missing data, that is, variables whose values are unknown.
- For example, we might have conducted a survey, and some people might not have answered certain questions.
- Or, we might have various sensors, some of which fail.
- The corresponding design matrix will then have "holes" in it; these missing entries are often represented by "**NaN**" or "**?**"
- The goal of **matrix completion** is to infer possible values for the missing entries.

Some Basic Concepts in Machine Learning (4)

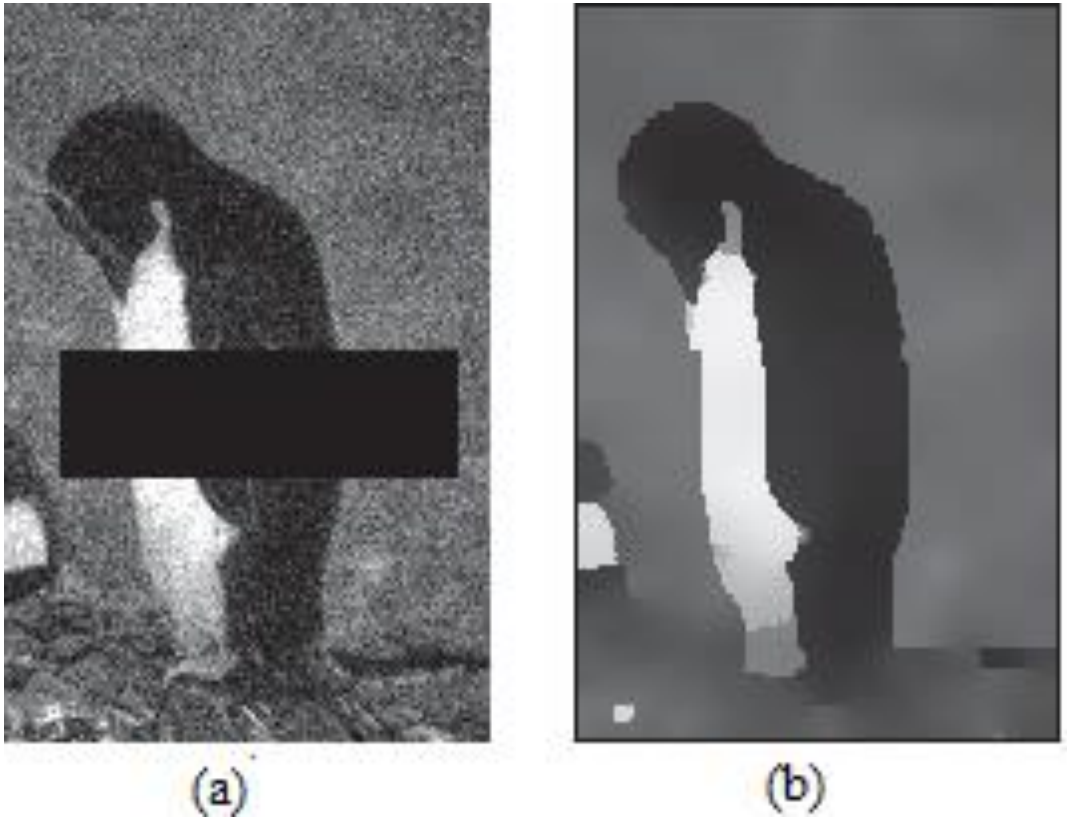


Figure: (a) A noisy image with an occluder. (b) An estimate of the underlying pixel intensities.

- We can use “Image Inpainting” or, “Market Basket Analysis” to *estimate missing value* or *matrix computation*.

Some Basic Concepts in Machine Learning (5)

No free lunch theorem (Wolpert 1997):

- Much of machine learning is concerned with devising different models, and different algorithms to fit them.

- We can use methods such as cross validation to empirically choose the best method for our problem.

- However, there is no universally best model — this is sometimes called the no free lunch theorem.

- The reason for this is that a set of assumptions that works well in one domain may work poorly in another.

- Because of the no free lunch theorem, we need to develop many different types of models, to cover the wide variety of data that occurs in the real world.

Some Basic Concepts in Machine Learning (6)

Training, testing, and validation sets:

- We can use **three** sets of data:
  - the training set is to train the algorithm,
  - the validation set is to keep track of how well it is doing as it learns,
  - the test set is to produce the final results.

- Typical, the proportion of training: testing: validation is something like 50:25:25 to 60:20:20 (can vary further).

- **Splitting method** (**Important**): How you do the splitting can also matter.
  - Many datasets are presented with the first set of datapoints being in class 1, the next in class 2, and so on.
  - If you pick the first few points to be the training set, the next the test set, etc., then the results are going to be pretty bad, since the training did not see all the classes.
  - This can be dealt with by randomly reordering the data first, or by assigning each datapoint randomly to one of the sets.

Some Basic Concepts in Machine Learning (7)

- The approach to split data into 3 sets could be infeasible:
  - especially since for supervised learning it all has to have target values attached, and
  - it is also not always easy to get accurate labels,
  - the size of the training data could be very small.

- **Remedy**: to perform leave-some-out, multi-fold cross-validation -
  - The idea is to randomly partition the dataset into K subsets, and one subset is used as a validation set, while the algorithm is trained on all the others (see Figure (in the next slide)).
  - A different subset is then left out and a new model is trained on that subset, repeating the same process for all the different subsets.
  - Finally, the model that produced the lowest validation error is tested and used.
  - We've traded off data for computation time, since we've had to train K different models instead of just one.
  - In the most extreme case of this there is leave-one-out cross-validation, where the algorithm is validated on just one piece of data, training on all the rest.

Some Basic Concepts in Machine Learning (8)

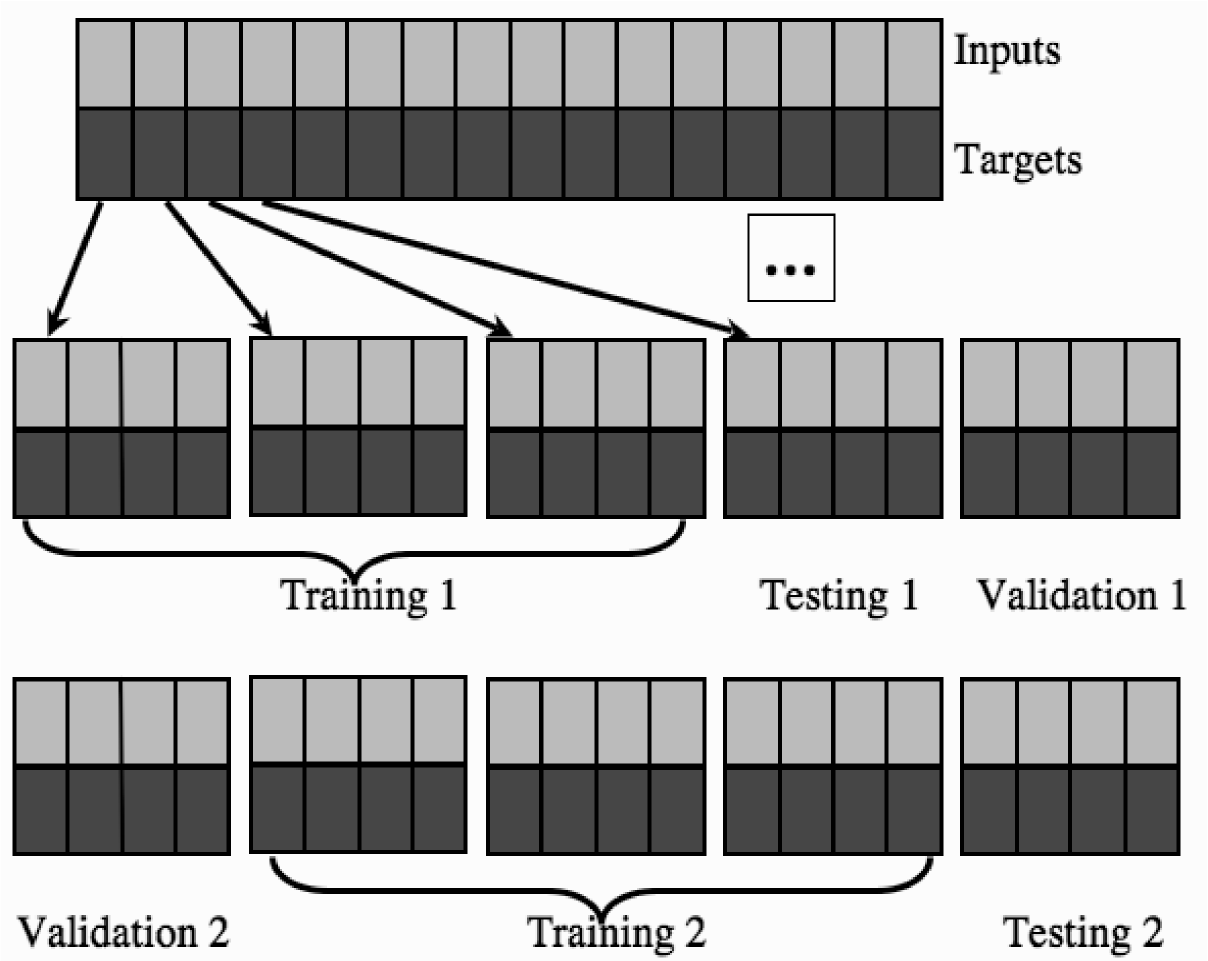


Figure: Leave-some-out, multi-fold cross-validation gets around the problem of data shortage by training many models. It works by splitting the data into sets, training a model on most sets and holding one out for validation (and another for testing). Different models are trained with different sets being held out.

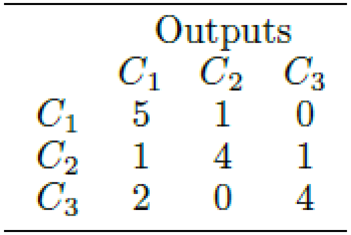
Some Basic Concepts in Machine Learning (9)

- **The confusion matrix:** Here, we make
  - a square matrix that contains all the possible classes in both the horizontal and vertical directions and
  - list the classes along the top of a table as the predicted outputs, and then
  - down the left-hand side as the targets.

- Example, the element of the matrix at (i, j) tells us how many input patterns were put into class i in the targets, but class j by the algorithm.

- Anything on the leading diagonal (the diagonal that starts at the top left of the matrix and runs down to the bottom right) is a correct answer.

- Suppose that we have three classes: C_1, C_2, and C_3. Now we count the number of times that the output was class C_1 when the target was C_1, then when the target was C_2, and so on until we've filled in the table:



	Outputs		
	C_1	C_2	C_3
C_1	5	1	0
C_2	1	4	1
C_3	2	0	4

Some Basic Concepts in Machine Learning (10)

Performance metrics:

- Assume a new test method is applied to screen people for cancer. The test results can be positive (have cancer) or negative (do not have cancer).

- The prediction or test results for each subject may or may not match the subject's actual status. In this context, the terminologies –

    - ✓ True Positive (TP) refers to the situation, where subjects having cancers are predicted to have cancer;

    - True Negative (TN) refers to the situation, where subjects do not have cancer are predicted to have no cancer;

    - False Positive (FP) refers to the situation, where subjects do not have cancer but are predicted to have cancer;

    - False Negative (FN) refers to the situation, where subjects have cancer but are predicted to have no cancer.

Some Basic Concepts in Machine Learning (11)

… Performance metrics:

- *Accuracy* is then defined as the sum of the number of true positives and true negatives divided by the total number of examples (where, TP stands for True Positive, FP stands for False Positive, etc.):

$$\boldsymbol{Accuracy} = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ prediction\ made} = \frac{TP+TN}{TP+FP+TN+FN}$$

- The problem with accuracy is that it doesn’t tell us everything about the results, since it turns four numbers into just one. Thus, we need additional metrics:

a. ***Sensitivity*** (also known as the ***true positive rate***, ***recall***, ***probability of detection***, ***hit rate***) is the ratio of positive data points that are correctly predicted as positive, with respect to all positive data points.

$$Sensitivity = \frac{TP}{TP + FN}$$

Some Basic Concepts in Machine Learning (12)

… Performance metrics:

b. Specificity (also known as the ***true negative rate, selectivity***) is the ratio of the negative data points that are correctly predicted as negative, with respect to all negative data points.

$$Specificity = \frac{TN}{TN+FP}$$

c. ***Precision*** (also known as the ***positive predictive value***) is the number of correct positive results divided by the number of positive results predicted by the classifier. That is, precision is defined as the accuracy of the judgment.

$$Precision = \frac{TP}{TP + FP}$$

Some Basic Concepts in Machine Learning (13)

… Performance metrics:

d. False positive rate is calculated as the ratio between the number of negative events wrongly categorized as positive (false positives) and the total number of actual negative events.

$$False\ positive\ rate\ (FPR) = \frac{FP}{TN+FP} = 1 - specificity$$

e. **F-measure (**also known as F-score): An F$_1$-score is composed of precision and recall, both calculated as percentages and combined as harmonic mean to assign a single number, easy for comprehension. This is also known as F-measure or balanced F-score:

$$F_1\ score = 2\frac{precision \times recall}{precision+recall} = \frac{2\,TP}{2TP+FP+FN}$$

Some Basic Concepts in Machine Learning (14)

The Receiver Operator Characteristic (ROC) curve: To draw ROC curve, TPR and FPR both are computed at threshold values such as (0.00, 0.02, 0.04, …., 1.00) and a graph is drawn (see Figure 1). The AUC is the area under the curve of the plot. AUC close to or equal to 1.0 indicates the best performance.

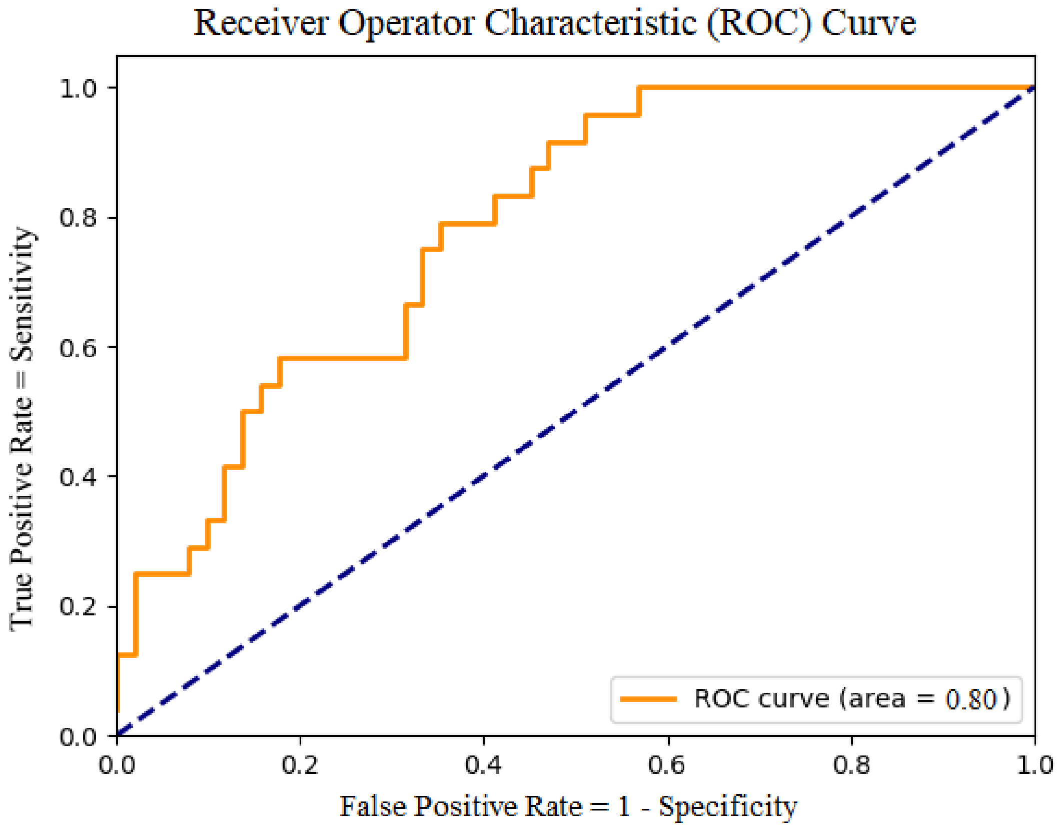


Figure 1: An example of an ROC curve. The diagonal line represents exactly a chance, so anything above the line is better than chance, and the further from the line, the better.

Some Basic Concepts in Machine Learning (15)

Unbalanced datasets:

- Note that for the accuracy, we have implicitly assumed that there are the same number of positive and negative examples in the dataset (which is known as a balanced dataset).

- However, this is often not true (this can potentially cause problems for the learners). In the case where it is not, we can compute the ***balanced accuracy*** as the sum of sensitivity and specificity divided by 2.

$$Balanced\ Accuracy = \frac{(Sensitivity + Specificity)}{2} = \frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right)$$

- A more correct measure is ***Matthew's Correlation Coefficient*** **(MCC)**, which is computed as:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Scikit-Learn Design

Scikit-Learn's API is remarkably well designed, and the main design principles are:

Consistency: All objects share a consistent and simple interface:

- *Estimators*:
  - Any object that can estimate some parameters based on a dataset is called an **estimator** (e.g., an imputer is an estimator).
  - The estimation itself is performed by the **fit()** method, and it takes only a dataset as **a** parameter (or **two** for supervised learning algorithms; the second dataset contains the labels).
  - Any other parameter needed to guide the estimation process is considered a **hyperparameter** (such as an imputer's strategy), and it must be set as an instance variable (generally via a constructor parameter).

- *Transformers*:
  - Some estimators (such as an imputer) can also transform a dataset; these are called transformers.
  - The transformation is performed by the **transform()** method with the dataset to transform as a parameter. It returns the transformed dataset.
  - This transformation generally relies on the learned parameters, as is the case for an imputer.
  - All transformers also have a convenience method called **fit_transform()** that is equivalent to calling fit() and then transform() (but sometimes fit_transform() is optimized and runs much faster).

… Scikit-Learn Design

- *Predictors*:
  - Finally, some estimators, given a dataset, are capable of making predictions; they are called predictors.
  - For example, the **LinearRegression** model was a predictor.
  - A predictor has a **predict()** method that takes a dataset of new instances and returns a dataset of corresponding predictions.
  - It also has a **score()** method that measures the quality of the predictions, given a test set (and the corresponding labels, in the case of supervised learning algorithms).

Inspection:

- All the estimator's hyperparameters are accessible directly via public instance variables (e.g., imputer.strategy), and
- all the estimator's learned parameters are accessible via public instance variables with an underscore suffix (e.g., imputer.statistics_).

Nonproliferation of classes:

- Datasets are represented as NumPy arrays or SciPy sparse matrices, instead of homemade classes.
- Hyperparameters are just regular Python strings or numbers.

… Scikit-Learn Design

Composition:

- Existing building blocks are reused as much as possible.
- For example, it is easy to create a **Pipeline** estimator from an arbitrary sequence of transformers followed by a final estimator, as we will see.

Sensible defaults:

- Scikit-Learn provides reasonable default values for most parameters, making it easy to quickly create a baseline working system.

End