Chapter 2

Introduction to supervised learning

Basic concepts of supervised learning. Empirical error. Maximum likelihood and error minimization. A regression example: curve fitting by least mean squares minimization. Curve fitting as a linear regression.

2.1 Supervised learning

We call *Supervised Learning* the task of learning to predict attributes from data that include those attributes. More formally, we have a set of examples with features X and some label Y, $\{(x^1, y^1), ..., (x^n, y^n)\}$, and we assume there is some unknown function $F(X) : X \to Y$. Our goal is to find a function $g(\theta, X) : X \to Y$, which is a function of some set of parameters θ, that approximates the unknown $F(X) : X \to Y$ and can tells us the Y values of any examples, even if not from our known set.

The reason for calling this supervised learning is that, by having all the Y values, we can supervise the learning process by comparing the predicted values to the known values in the data. This allows us to empirically estimate the error of each hypothesis.

The *empirical error*, or *training error*, is a measure of how any hypothesis obtained by instantiating the parameters θ of our model (the $g(\theta, X)$ set of functions in our *hypothesis class*) performs in predicting the Y values of the training data. Thus, we can formulate one machine learning problem in this way:

1. The task: Predict the Y values in the $\{(x_1, y_1), ..., (x_n, y_n)\}$ set.
2. The performance measure: training error, using Y.
3. The data: the $\{(x_1, y_1), ..., (x_n, y_n)\}$ set.

Note that this is not a very useful problem to solve because this only aims at predicting the values that we already know. In other words, this tries to approximate the unknown function $F(X) : X \to Y$ only within our known data set. A better alternative would be to find the hypothesis that would minimize the error for any examples, even those not included in the training set. That is usually the goal of a machine learning application. But we'll set aside that complication for now and focus on the simplified problem first.

Supervised learning problems can be split in two different categories. *Classification* problems are those in which the Y values belong to discrete categories. For example, the classification of email messages into spam and not spam or the classification of mushrooms into edible and poisonous categories. In this case the error can be something like the percentage of misclassified examples. *Regression* problems are those in which the Y values are continuous. We will focus on *regression* in this chapter and the next, and cover *classification* later on.

2.2 Linear Regression

A *linear regression* is a regression in which the hypothesis class corresponds to the model $y = \theta_1 x_1 + \theta_2 x_2 + ... + \theta_{n+1}$, where each x_n is one dimension of the input space. Let's suppose, to simplify, that our input space has only one dimension and we have a set of (x, y) points and want to find the best way to predict the y value of each point given the x value assuming some specific *hypothesis class*. Let us suppose the *hypothesis class* is the set of all straight lines, defined by the parameters of the model $y = \theta_1 x + \theta_2$. Figure 2.1 shows an example of a data set of points and possible lines from our *hypothesis class*, obtained by instantiating the model with different values of θ_1 and θ_2.

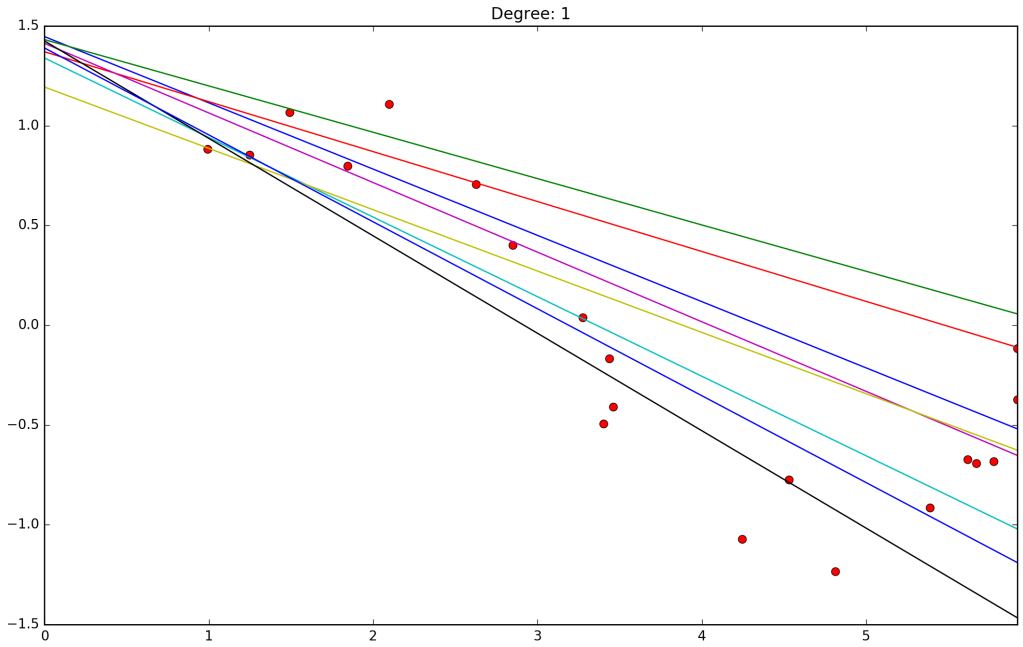

Figure 2.1: Example of lines for predicting the y values in these data.

How can we determine the best line? Let us assume that the dependent variable y is some (unknown) function of the independent variable x plus some error:

$$y = F(x) + \epsilon$$

We want to approximate $F(x)$ with a model $g(x, \theta_1, \theta_2)$. Assuming that the error is random and normally distributed:

$$\epsilon \sim N(0, \sigma^2)$$

then, if $g(x, \theta_1, \theta_2)$ is a good approximation of the true function $F(x)$, the probability of having a particular y value given some x value can be computed from our function $g(x, \theta_1, \theta_2)$ as:

$$p(y|x) \sim \mathcal{N}(g(x, \theta_1, \theta_2), \sigma^2)$$

This allows us to estimate the probability of the data coming out with the distribution we observe in our data set given any hypothesis instantiating θ, representing the vector of all $\theta_1, ..., \theta_n$ parameters (in this case, θ_1, θ_2). The probability of the data given the hypothesis is the *likelihood* of the hypothesis. Note that we cannot assume a probability for the hypothesis, at least in a frequentist sense, because the hypothesis is not a random variable. What we assume to be random here is the sampling of data that resulted in obtaining this dataset from the universe of all possible data.

Thus, given our dataset $\mathcal{X} = \{x^t, y^t\}_{t=1}^N$ and knowing that $p(x, y) = p(y|y)p(x)$, then the likelihood of the set of parameters θ is

$$l(\theta|\mathcal{X}) = \prod_{t=1}^{n} p(x^t, y^t) = \prod_{t=1}^{n} p(y^t|x^t) \times \prod_{t=1}^{n} p(x^t)$$

Now we know how to choose the best hypothesis: we pick the one with the *maximum likelihood*. In other words, we pick the hypothesis that estimates the largest probability of obtaining the data we have. This is a generic approach that is often used in machine learning. But, to simplify the math, let us change the expression. First, we know that the hypothesis that maximizes the likelihood also maximizes the logarithm of the likelihood, so we can focus on the logarithm of the likelihood, $\mathcal{L}$, instead of the likelihood l:

$$\mathcal{L}(\theta|\mathcal{X}) = log \prod_{t=1}^{n} p(y^t|x^t) + log \prod_{t=1}^{n} p(x^t)$$

We can also ignore the $p(x)$ term since this corresponds to the probability of drawing those x values in our data from the universe of possible values and this is the same for all hypotheses (all values of θ) we are considering.

$$\mathcal{L}(\theta|\mathcal{X}) \propto log \prod_{t=1}^{n} p(y^t|x^t)$$

Since we assume that the probability of obtaining some y value given some x is approximately normally distributed around our prediction, we can replace that term with the corresponding distribution:

$$p(y|x) \sim \mathcal{N}(g(x, \theta), \sigma^2)$$

and then replace it with the expression for the normal distribution:

$$\mathcal{N}(z, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(z-\mu)^2/2\sigma^2}$$

leaving:

$$\mathcal{L}(\theta|\mathcal{X}) \propto log \prod_{t=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} e^{-[y^t - g(x^t|\theta)]^2/2\sigma^2}$$

which can be simplified to:

$$\mathcal{L}(\theta|\mathcal{X}) \propto log \prod_{t=1}^{n} e^{-[y^t - g(x^t|\theta)]^2}$$

$$\mathcal{L}(\theta|\mathcal{X}) \propto -\sum_{t=1}^{n} [y^t - g(x^t|\theta)]^2$$

But this is the expression of the square of the training error:

$$E(\theta|\mathcal{X}) = \sum_{t=1}^{n} [y^t - g(x^t|\theta)]^2$$

So, basically, to find the hypothesis with the *maximum likelihood* we need (under our assumptions) to find the hypothesis with the *minimum squared error* on our training set. This problem is called a *Least Mean Squares minimization*.

Note that the squared error is often represented by this expression:

$$E(\theta|\mathcal{X}) = \frac{1}{2}\sum_{t=1}^{n}[y^t - g(x^t|\theta)]^2$$

The reason for this is that, when computing the derivative of this error as a function of the parameters, the square power cancels the 2 in the denominator, simplifying the algebra. However, the values obtained for the parameters minimizing the squared error or one half the squared error are the same. This is merely an algebraic convenience.

2.3 Least Mean Squares minimization

In our straight line model (Figure 2.1) we need to consider two parameters, θ_1 and θ_2. If we compute the squared error for all combinations of parameters in some range we will obtain something like shown in Figure 2.2.

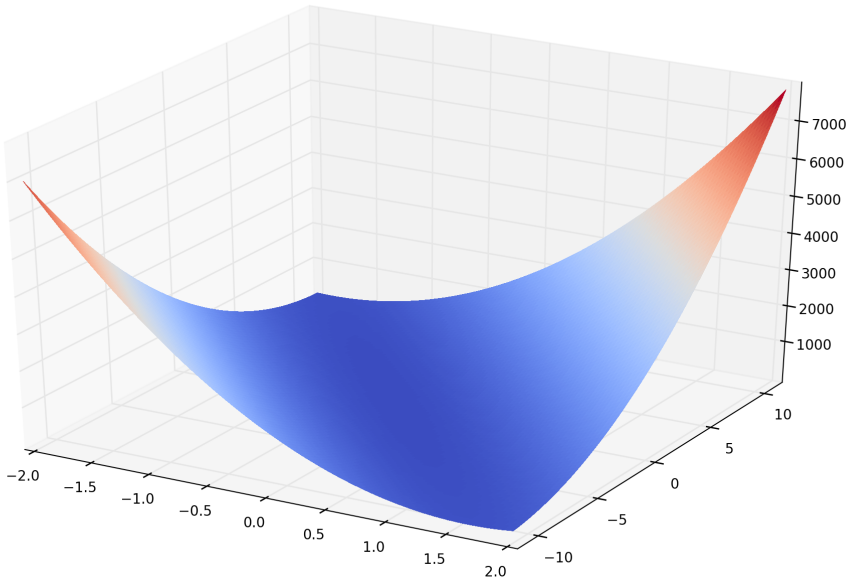

Figure 2.2: Surface of the squared error function for two parameters.

To find the optimal combination of parameters we need to find the lowest point of this surface. Thus we use a gradient descent algorithm that starts from an arbitrary point as an initial guess and then proceeds to descend the error surface in different directions until converging to the desired minimum. This is illustraded in Figure 2.3

This will be a useful approach in many different problems we will encounter. The important idea is this: if we assume that our model approaches the desired target values with some normal error, as a function of the features in our dataset and the parameters of the model, then maximizing the likelihood of our parameters is equivalent to minimizing the squared error. We shall see in the next chapter that considering only the training error may not be a good idea, but in any case this *least mean squares minimization* approach is an important tool in machine learning.

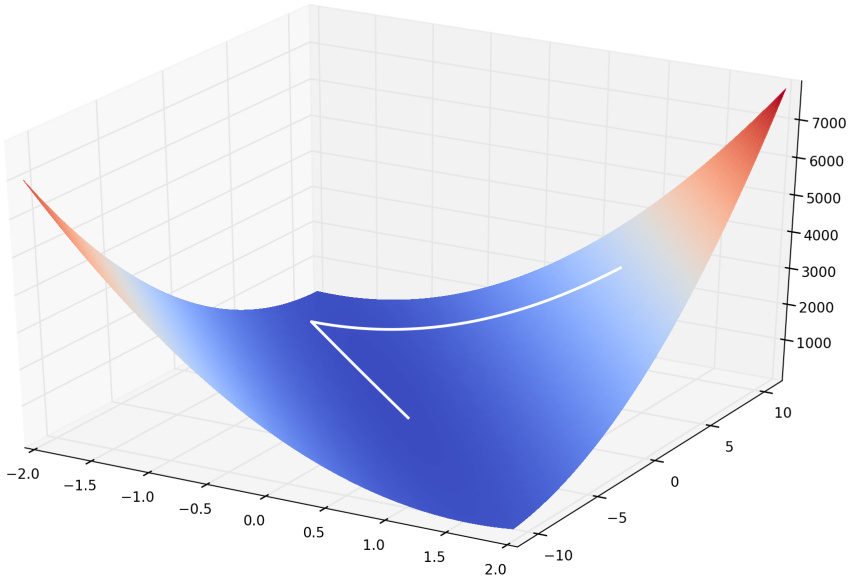

Figure 2.3: Gradient descent on the squared error surface.

In this way, we can find the hypothesis that best fits the data. The straight line that minimizes the squared error for our data set is shown in Figure 2.4. One thing we can notice immediately is that, despite being the best straight line to predict the y values in our data, it is still a very poor predictor of these values. We need to change your *hypothesis class* and try to find different hypotheses.

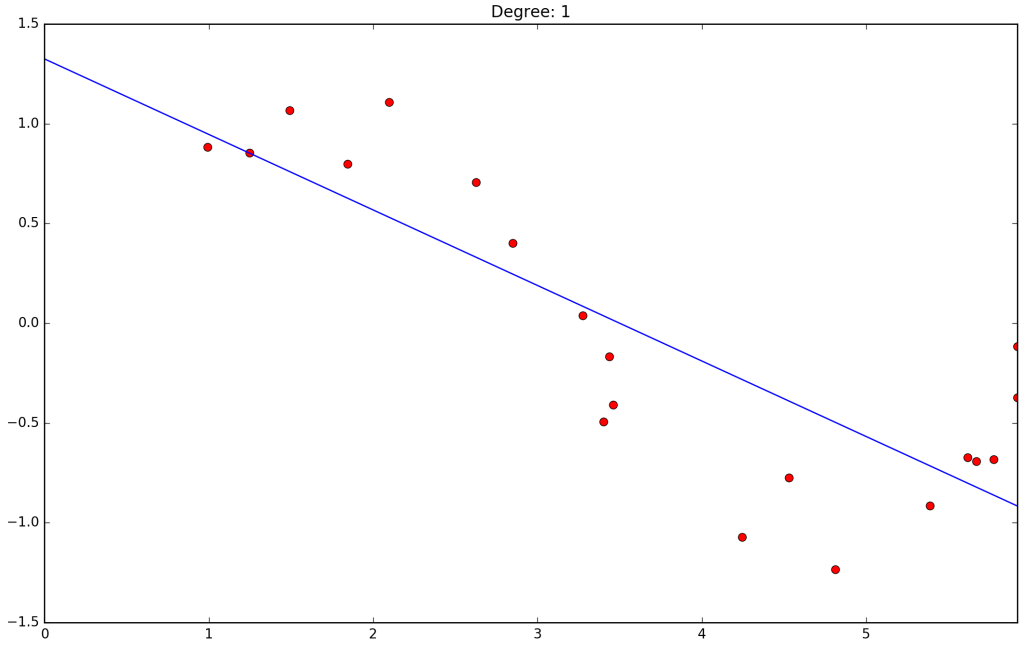

Figure 2.4: Best straight line for predicting the y values in our data.

2.4 Beyond the straight

Since fitting a straight line to these data is so evidently inadequate, we can try to consider alternatives. Let us start by changing the data. We have a set of values for x and y, and we fit a straight line that gives

us y given each x. But imagine that we spread our points over a plane with coordinates (x_1, x_2) instead of x, and found the plane that minimized the error between to the y values in this new space. We are still fitting a "straight" function, it's just in more dimensions than the initial one. So let us compute this new data set $\mathcal{X}_{\in} = \{x_1^t, x_2^t, y^t\}$ by making $x_1 = x^2$ and $x_2 = x$ for each point in the original set. This gives us the data set represented in Figure 2.5

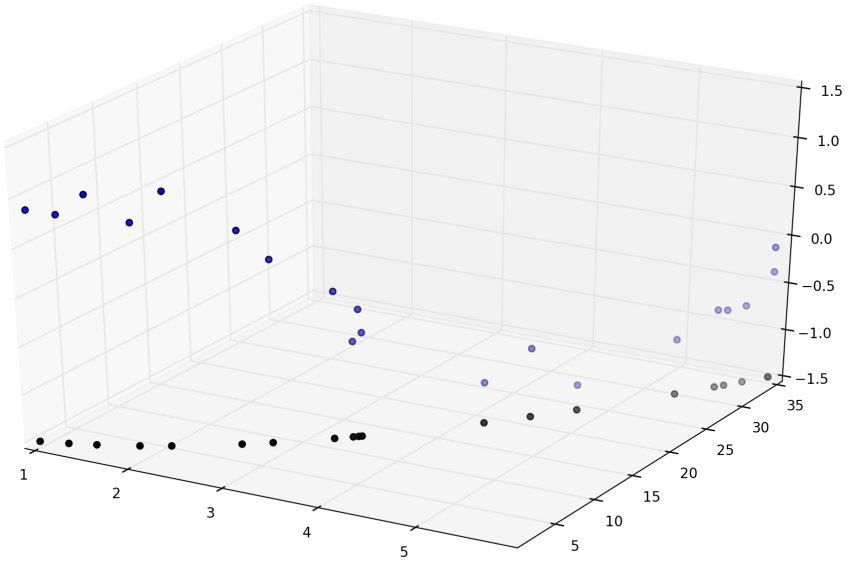

Figure 2.5: Transformed data set. Note that the black dots below are just the "shadows" to indicate the projection of the data in the (x^2, x) plane.

Now our model is the equation for the plane, which we can write as $y = \theta_1 x_1 + \theta_2 x_2 + \theta_3$, and each hypothesis in this hypothesis class will be a particular plane obtained by instantiating the three parameters. Minimizing the square error, we obtain the plane shown in Figure 2.6.

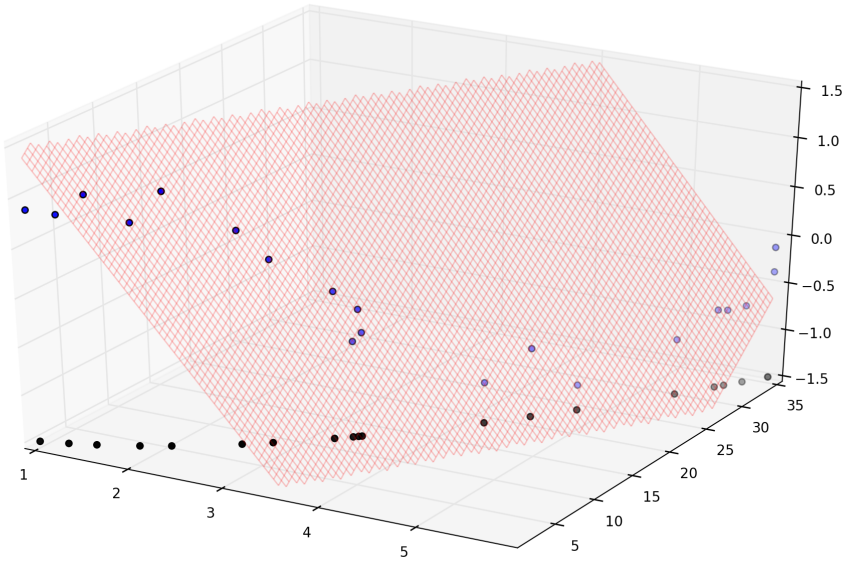

Figure 2.6: The plane that best fits the transformed data.

Now we can convert back into our original problem. We know that our data always falls in the line where $x_1 = x_2^2$, because that was our initial transformation. If we intersect the best plane we found with this line, we get a line that we can project into the initial (x, y) space. This line and the resulting projection is shown in Figure 2.7.

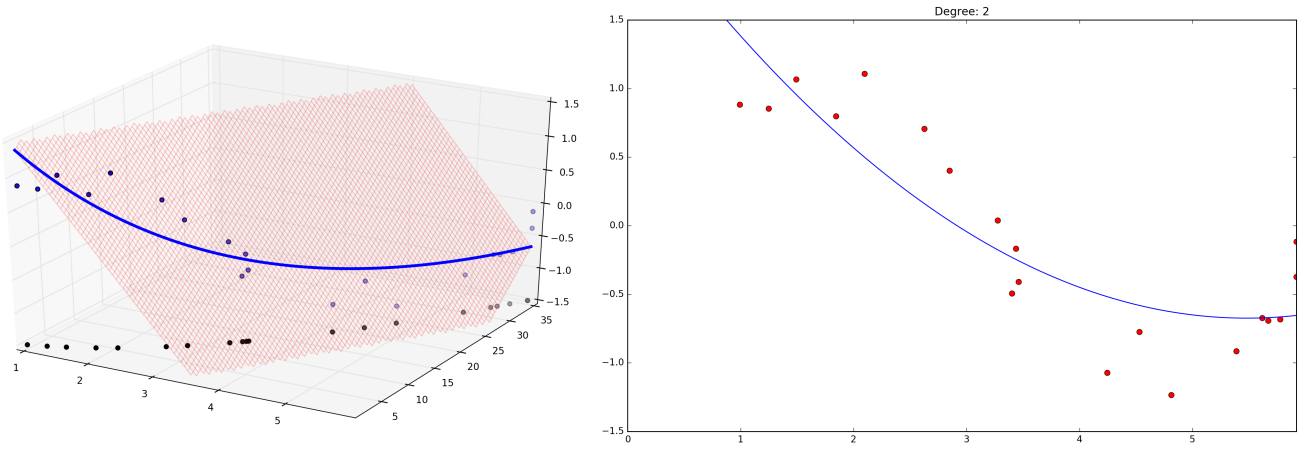

Figure 2.7: The line for the best fit projected back into the original data set.

This is the equivalent of doing a second degree polynomial regression on the original data. We could just have kept the original data set and just changed our model from the initial straight line, which is a first degree polynomial ($y = \theta_1 x + \theta_2$) to that of a quadratic curve, $y = \theta_1 x^2 + \theta_2 x + \theta_3$. In fact, this is the easiest way to solve this problem with the tools we are using in this course. Here is the code for loading the data and computing the best second degree polynomial.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 mat = np.loadtxt('polydata.csv',delimiter=';')
5 x,y = (mat[:,0], mat[:,1])
6 coefs = np.polyfit(x,y,2)
```

The first two lines import the `numpy` library for the computations and the `matplotlib` library for the plot, shown below. Then we load the `csv` file with the data, splitting the matrix into two variables `x` and `y`. Then line 6 does the actual work of computing the coefficients of the second degree polynomial. Now we can plot the results by first computing the polynomial over 100 points and plotting the data, the line and saving the figure. The code continues below.

```
7 pxs = np.linspace(0,max(x),100)
8 poly = np.polyval(coefs,pxs)
9
10 plt.figure(1, figsize=(12, 8), frameon=False)
11 plt.plot(x,y,'.r')
12 plt.plot(pxs,poly,'-')
13 plt.axis([0,max(x),-1.5,1.5])
14 plt.title('Degree: 2')
15 plt.savefig('testplot.png')
16 plt.close()
```

However, there is an important lesson here that will reveal its usefulness when we deal with more complex problems. We can use a simple hypothesis class, for example all linear relations of variables,

corresponding to all hyperplanes in an N-dimensional problem, and use that hypothesis class for fitting or classifying our data in complex ways by transforming our data into higher dimensional representations of the same problem. In this example, we saw how the (straight) plane we used in 3 dimensions to fit our data projects back into a curved line in the original problem with only one independent and one dependent dimensions. This is an important way of thinking about machine learning problems.

2.5 Getting carried away

If a quadratic curve fits our data better than a straight line, a third degree polynomial is even better. Or higher degrees. Figure 2.8 shows the result of fitting a third degree and a fifteenth degree polynomial to our data. The polynomial of degree 15 certainly fits the data better, greatly reducing the training error. But is this really the best option? We will discuss this problem in the next chapter and lecture.

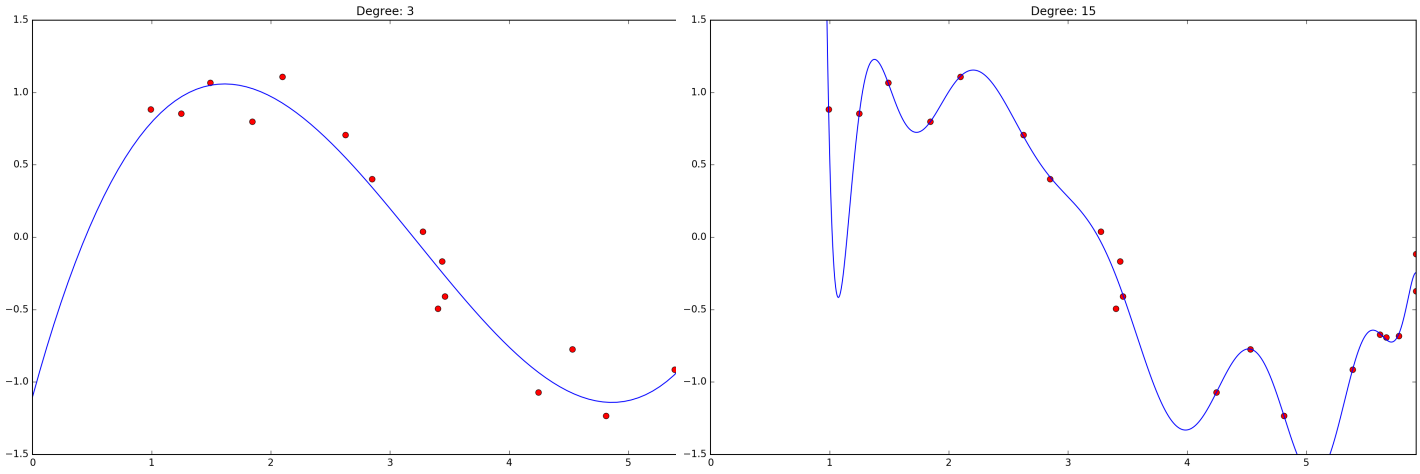

Figure 2.8: Fitting our data with polynomials of degree 3 and 15.

2.6 Summary

In this chapter we met several important ideas that we will revisit often during this course. First, we had to choose a *hypothesis class* for approximating the unknown function that determines the relations between the variables we are studying. Second, had to choose some measure of adjustment to select our parameters. In this case, we chose the *maximum likelihood*, which reduced to the *least squared error* measure for fitting our lines. Third, to adjust our parameters we needed to solve an optimization problem to find the values that optimize our adjustment measure. In this case, that minimize the squared error. Then we saw two more important ideas. One is that we can increase the power of linear models by increasing the number of features using non-linear transformations from the original feature values. The other is that this can result in fitting the known data too well. In the next chapter we will see how to address this last problem.

2.7 Further Reading

1. Bishop [4], Chapter 1
2. Alpaydin [2], Section 2.6

3. Marsland [17], Sections 1.4 and 2.4.

Bibliography

[1] Uri Alon, Naama Barkai, Daniel A Notterman, Kurt Gish, Suzanne Ybarra, Daniel Mack, and Arnold J Levine. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proceedings of the National Academy of Sciences*, 96(12):6745–6750, 1999.

[2] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010.

[3] David F Andrews. Plots of high-dimensional data. *Biometrics*, pages 125–136, 1972.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, New York, 1st ed. edition, oct 2006.

[5] Deng Cai, Xiaofei He, Zhiwei Li, Wei-Ying Ma, and Ji-Rong Wen. Hierarchical clustering of www image search results using visual. Association for Computing Machinery, Inc., October 2004.

[6] Guanghua Chi, Yu Liu, and Haishandbscan Wu. Ghost cities analysis based on positioning data in china. *arXiv preprint arXiv:1510.08505*, 2015.

[7] Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems*, pages 396–404. Morgan Kaufmann, 1990.

[8] Pedro Domingos. A unified bias-variance decomposition. In *Proceedings of 17th International Conference on Machine Learning. Stanford CA Morgan Kaufmann*, pages 231–238, 2000.

[9] Hakan Erdogan, Ruhi Sarikaya, Stanley F Chen, Yuqing Gao, and Michael Picheny. Using semantic analysis to improve speech recognition performance. *Computer Speech & Language*, 19(3):321–343, 2005.

[10] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[11] Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

[12] Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[13] Patrick Hoffman, Georges Grinstein, Kenneth Marx, Ivo Grosse, and Eugene Stanley. Dna visual and analytic data mining. In *Visualization'97., Proceedings*, pages 437–441. IEEE, 1997.

[14] Chang-Hwan Lee, Fernando Gutierrez, and Dejing Dou. Calculating feature weights in naive bayes with kullback-leibler measure. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 1146–1151. IEEE, 2011.

[15] Stuart Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982.

[16] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[17] Stephen Marsland. *Machine Learning: An Algorithmic Perspective*. Chapman & Hall/CRC, 1st edition, 2009.

[18] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[19] Yvan Saeys, Iñaki Inza, and Pedro Larrañaga. A review of feature selection techniques in bioinformatics. *bioinformatics*, 23(19):2507–2517, 2007.

[20] Roberto Valenti, Nicu Sebe, Theo Gevers, and Ira Cohen. Machine learning techniques for face analysis. In Matthieu Cord and Pádraig Cunningham, editors, *Machine Learning Techniques for Multimedia*, Cognitive Technologies, pages 159–187. Springer Berlin Heidelberg, 2008.

[21] Giorgio Valentini and Thomas G Dietterich. Bias-variance analysis of support vector machines for the development of svm-based ensemble methods. *The Journal of Machine Learning Research*, 5:725–775, 2004.

[22] Jake VanderPlas. Frequentism and bayesianism: a python-driven primer. *arXiv preprint arXiv:1411.5018*, 2014.