Chapter 3

Overfitting in Linear Regression

Overfitting. Training, validation and testing.

3.1 Estimating the true error

We saw in the previous lecture that we could improve the fit of a curve to a set of points by increasing the number of parameters. Figure 2.8 showed the difference between fitting the points with a third degree polynomial and a polynomial of degree 15. However, it is apparent that the result is fitting the known points by sacrificing the ability to correctly predict values not in the training set. This is a common concern in supervised learning problems.

In general, in a regression problem, we want to find a function to predict the Y values of elements of some universe $\mathcal{U}$ from their observable features X. The data is a labelled subset of $\mathcal{U}$, $\{(x_1, y_1), ..., (x_n, y_n)\}$, from which we can try to infer a function to predict Y and on which we can compute the *training error*, as we saw in the last lecture. However, the error we would like to minimise is the expected error over any element of $\mathcal{U}$, which is the *true error*, and not only the error measured in our training set (the *training error*).

One way to estimate the expected error in predicting the y value of any element of $\mathcal{U}$ is to reserve some elements of our data set specifically for this error estimate. These elements will not be used in training. Thus, we split our data into two sets: the *training set* and the *test set*. We use the *training set* to fit our function, minimizing the *training error*, and then the *test set* to estimate how our function performs in predicting the values of examples outside the *training set*. Figure 3.1 shows an example of fitting a fifth degree polynomial to 35 points in the data set (the training set, in blue) and then computing the *test error* with the remaining 15 points (shown in red).

The difference between the *test error* and the *training error* is called the *generalization error*, and is a measure of how well the learner can generalize from the training set to new examples.

It is important to note that the data set is split randomly between training and test sets. Depending on this selection, different curves will result from fitting the training set, with different test error values. Figure /ref3-diffsplits illustrates the result of a fifth degree polynomial fit to different subsets of the same data set, using the remaining points to evaluate the test error.

The *test error* is an unbiased estimate of the *true error*, but it is randomly distributed around the *true error*. This is because the *true error* is the average error for all the infinite possible data points while the *test error* is the error measured on the sample of points we assigned to the test set. The *test*

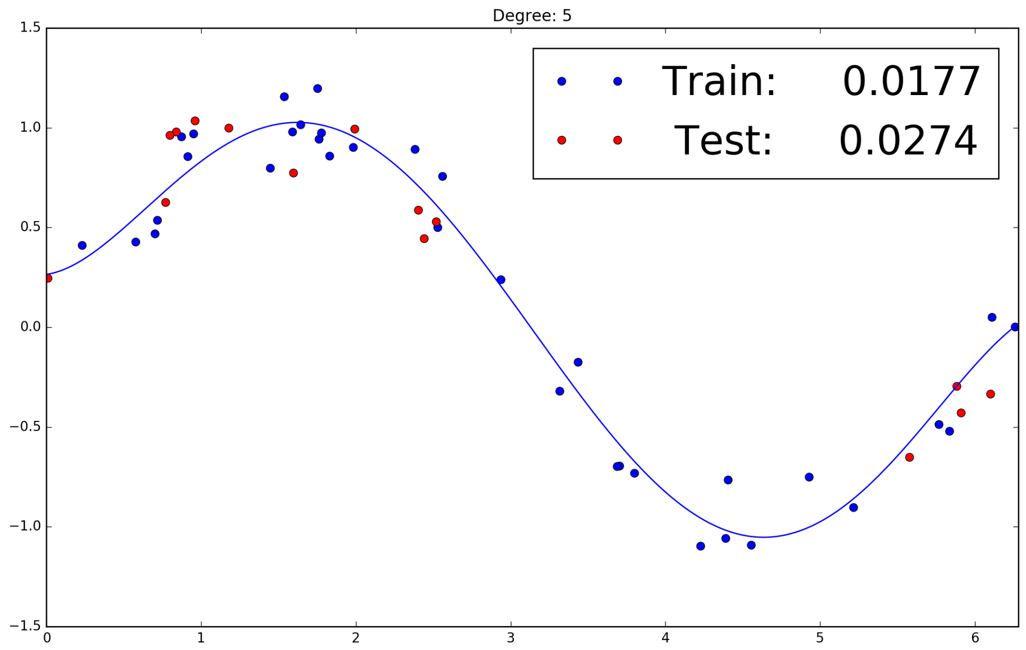

Figure 3.1: Training and test error example.

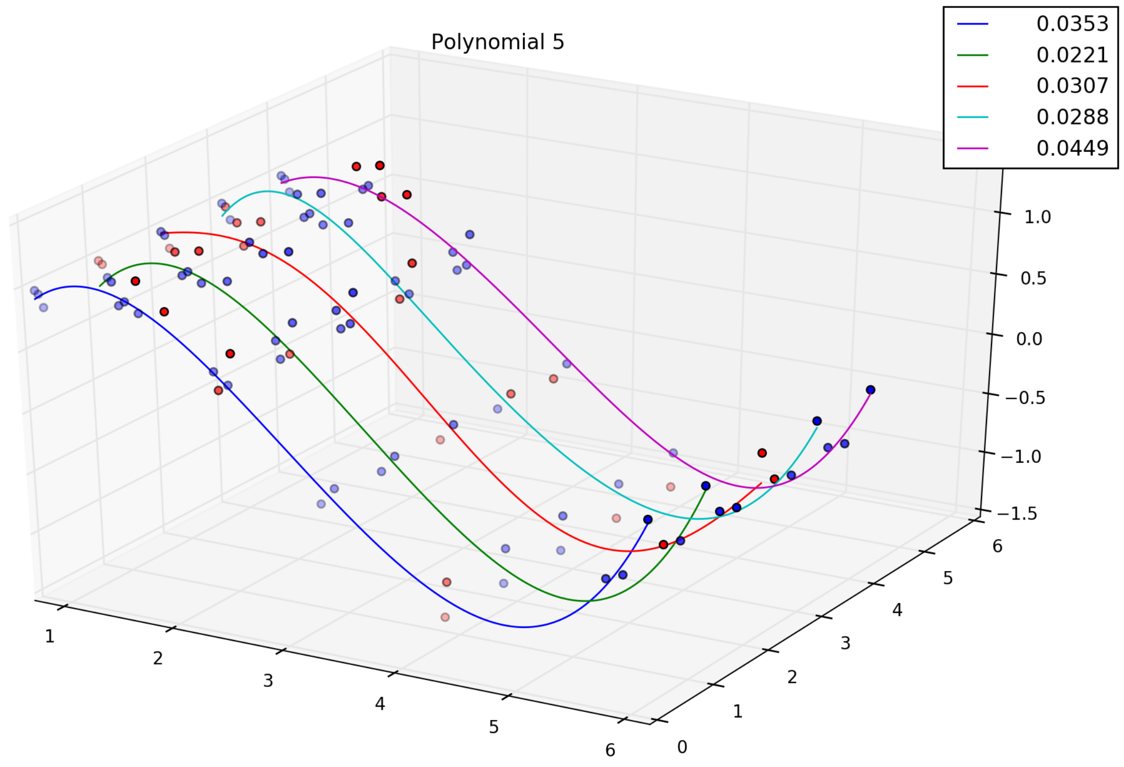

Figure 3.2: Fitting the same model to different training sets. The test error is indicated in the legend.

error is an unbiased estimator for the *true error* because, assuming the data set is a random sample of $\mathcal{U}$ and the training and test sets are randomly generated, then the average of the measured *test error*, over a large number of repetitions of the experiment, would tend towards the *true error*. However, a single measure of the *test error* will not correspond exactly to the *true error*. It is actually a sample from a probability distribution around the *true error*, like illustrated in Figure 3.3, because it depends on the points that were randomly assigned to the test set.

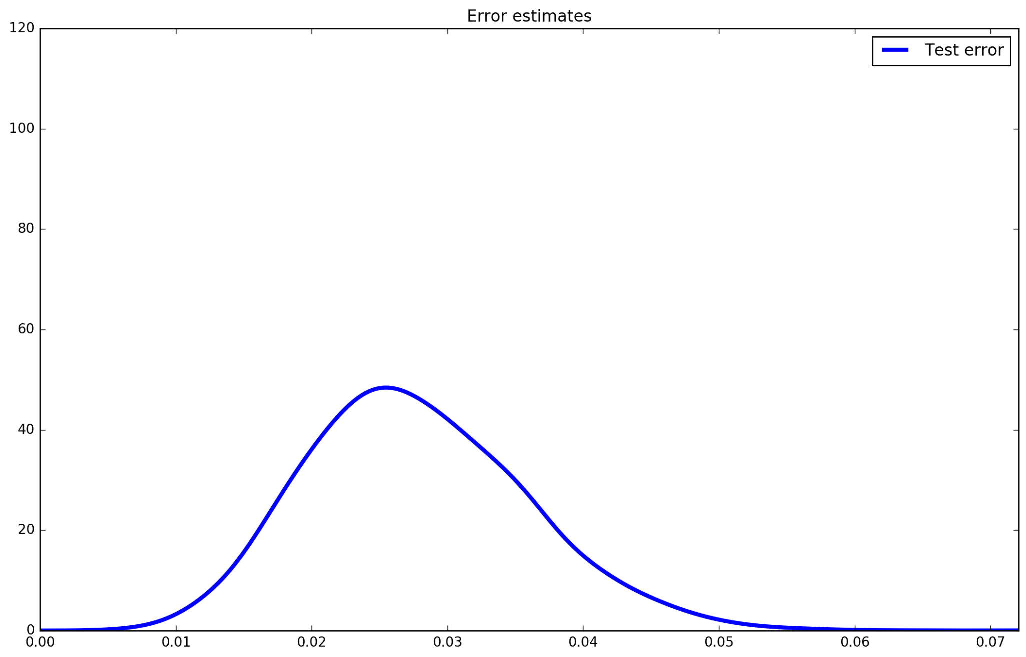

Figure 3.3: Probability distribution of the test error for different samples of the test set in the fifth degree polynomial example.

3.2 Underfitting and Overfitting

If the model is unable to fit the training set, both the training and test errors tend to be high because no hypothesis can be instantiated that accurately reflect the relation between the attributes and the value to predict. This is called *underfitting*. In the case of *underfitting*, replacing the model with one more capable of fitting the data will reduce both training and test errors. This is illustrated in Figure 3.4, as we move from degree 1 to degree 5. However, improving the fit between the model and the training set will eventually begin increasing the *test error*, even though the *training error* decreases. This is called *overfitting*, which is due to the model adapting to details of the training set that do not generalize to the universe from which the data was sampled. Higher degree polynomials have a lower *training error* but a larger *test error*

We can plot the two errors as a function of the degree of the polynomial to see this effect more clearly. Figure 3.5 illustrates this. The *training error*, in blue, decreases steadily as we increase the degree of the polynomial, increasing its ability to fit the training set. However, the *test error*, in red, only decreases until degrees 5 or 6. Afterwards, the models start *overfitting*, increasing the *test error* and the *generalization error*, which is the difference between the *test error* and the *training error*.

3.3 Model Selection and Validation

As Figure 3.5 shows, not all models are equally adequate for finding the hypothesis that best allows us to predict values in our universe $\mathcal{U}$. But, using the estimate of the true error in each case, we can find the model that performs best at this task. This procedure is called *model selection*: we use one set of data, the *training set*, to fit each model. Then we use another set of data to estimate the true error of each hypothesis resulting from fitting each model to the training set. We then select the hypothesis for which this estimate for the true error is lowest. In this case, this would be the polynomial of degree 5 shown in the second panel of Figure 3.4.

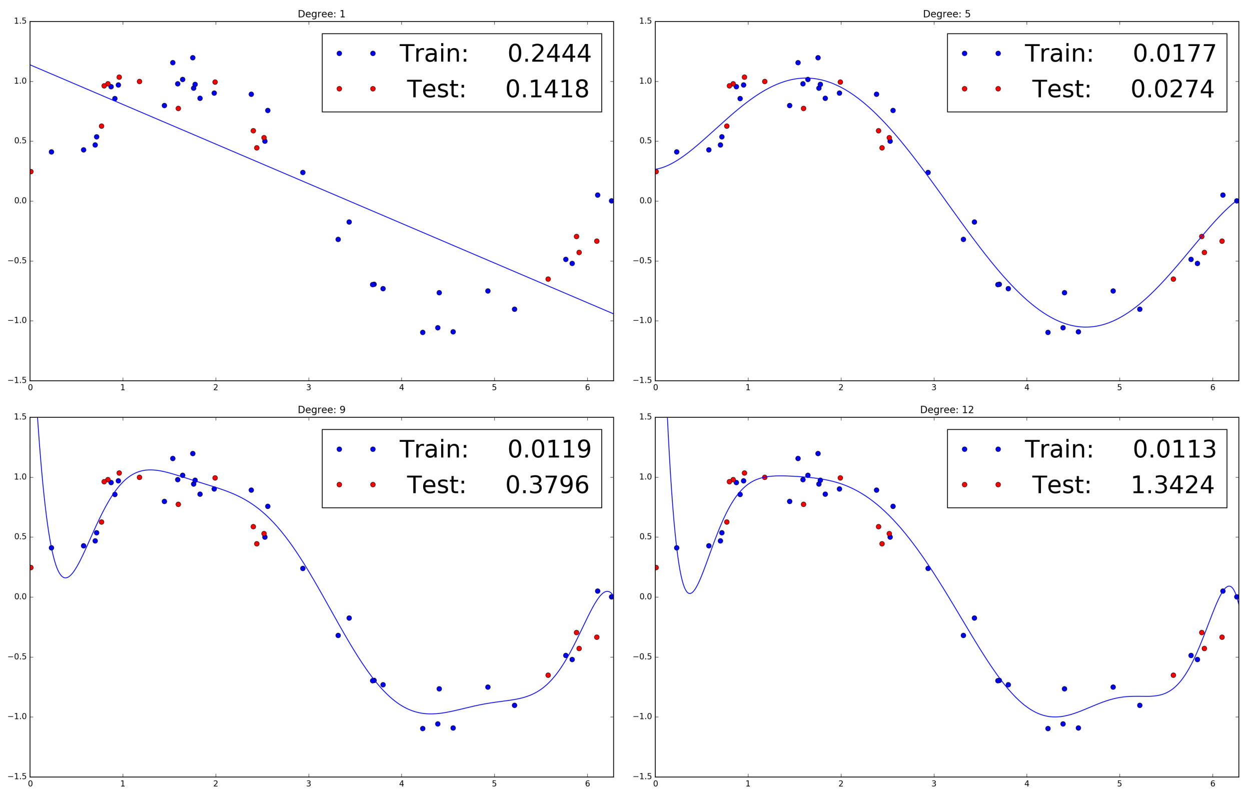

Figure 3.4: Different models fit to the same training set, evaluated with the same test set.

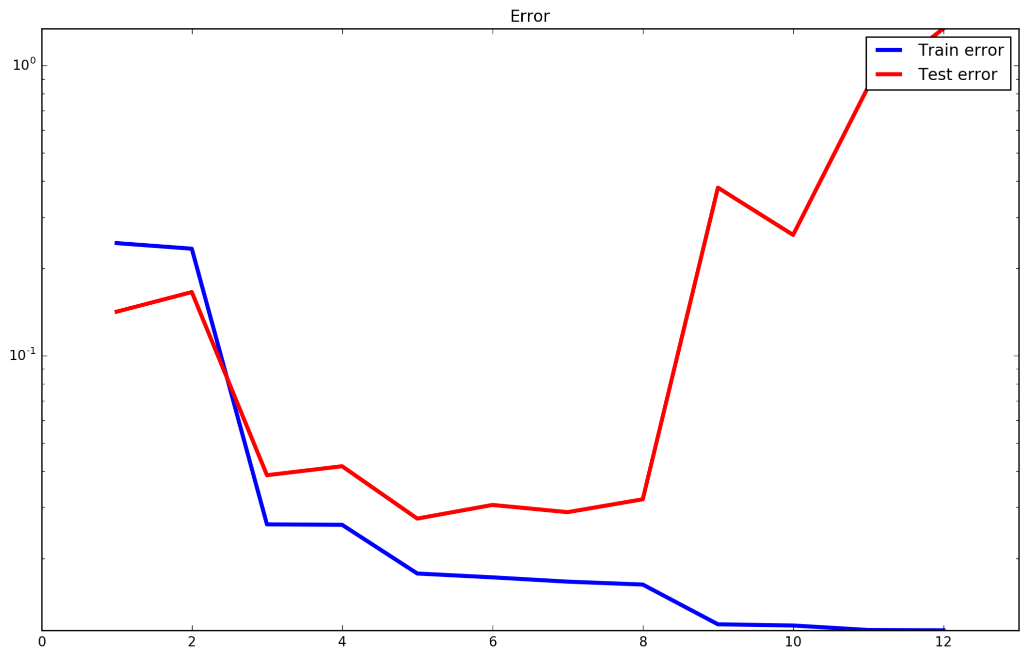

Figure 3.5: Plot of the training error (blue) and the test error (red) as a function of the degree of the polynomial.

However, if we select the hypothesis with the lowest error from a set of error estimates, then this error estimate is no longer an unbiased estimate of the true error. This is because we are selecting the smallest value out of that set of measured errors (one for each model). We can understand this with an analogy. If we choose people at random, some will be taller than average, others will be shorter than average but the average of their height will tend towards the average height of the population.

So, even though the height of people at random is not exactly the average height, it is an unbiased estimator of the average height. This is what happens when we use the *test error* of one hypothesis to estimate its *true error*. However, if we choose people at random in groups and then, from each group, we always pick the shortest person, the average height of those shortest people from each group will no tend towards the average height of the population.

Figure 3.6 compares the unbiased test error distribution, in blue, and the distribution of the smallest error measured in groups of 10 (in red). So, if we use the error estimate to select the best model and hypothesis, then we can no longer use that value as an unbiased estimate of the true error. It will tend to underestimate the true error. This is why, for *model selection* using the error estimates, we need to split our data set in three subsets. The *training set*, to fit each model, the *validation set* to obtain the error estimates to select the best hypothesis, and then a *test set* to obtain a final, unbiased, estimate of the true error. This *test error* can only be used for the final estimate.

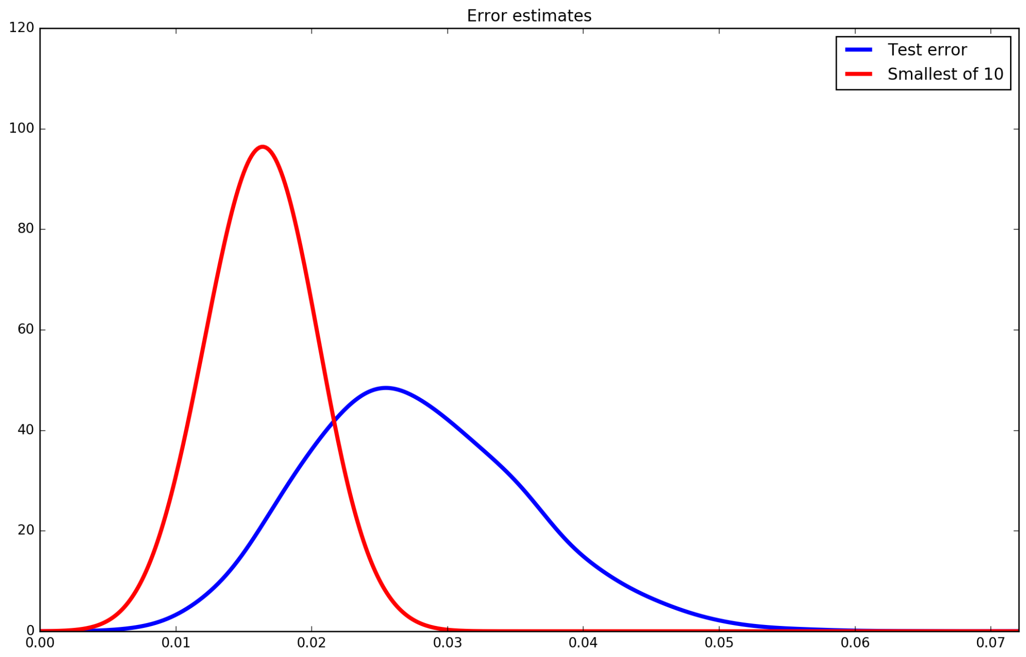

Figure 3.6: The probability distribution of the test error (in blue) and the smallest of groups of 10 test errors.

There are other methods of *model selection*, which we will see later on. In this lecture, the main point is to note this difference between training, validation and testing. Training is the process of fitting the model; validation allows us to choose an hypothesis and testing gives us an unbiased estimate of the true error. To ensure that this final estimate is unbiased, the test set cannot be used at any stage to train hypotheses or select models.

3.4 Regularization

Another approach to solve the *overfitting* problem is to change the learning algorithm to try to prevent the model from adjusting too much to details that do not generalize. One way to do this with our polynomial models is to use a high degree polynomial but add to the error function a penalty as a function of the coefficient values:

$$J(\theta) = \sum_{t=1}^{n} \left[y^t - g(x^t|\theta)\right]^2 + \lambda \sum_{j=1}^{m} \theta_j^2$$

This example, a quadratic penalty function, is called *ridge regression* [12]. Figure 3.7 shows the result of fitting a degree 15 polynomial with different values of the regularization weight λ.

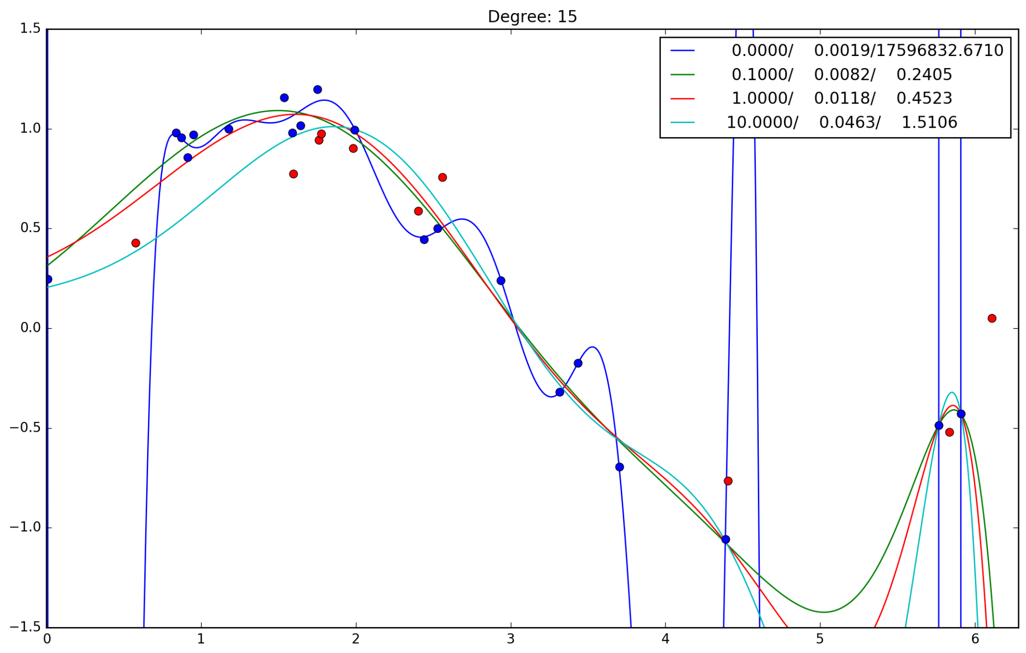

Figure 3.7: Fitting a degree 15 polynomial with different values of λ for regularization. The legend shows the λ, training error and test error.

3.5 Application Example

Figure 3.8 shows the plot of life expectancy versus *per capita* GDP for 180 countries in 2003[1].

In order to find the best model for this data, we will split it randomly into three sets. The *training set*, consisting of half of the points (90 points), the *validation set*, with 45 points, and the *test set*, also with 45 points. This is the code for loading and splitting the data:

```python
def random_split(data,test_points):
    """return two matrices splitting the data at random
    """
    ranks = np.arange(data.shape[0])
    np.random.shuffle(ranks)
    train = data[ranks>=test_points,:]
    test = data[ranks<test_points,:]
    return train,test

data = np.loadtxt('life_exp.csv',delimiter='\t')
scale=np.max(data,axis=0)
data=data/scale
train, temp = random_split(data, 90)
valid, test = random_split(temp, 45)
```

[1]http://www.indexmundi.com/g/correlation.aspx?v1=30&v2=67&y=2003&l=en

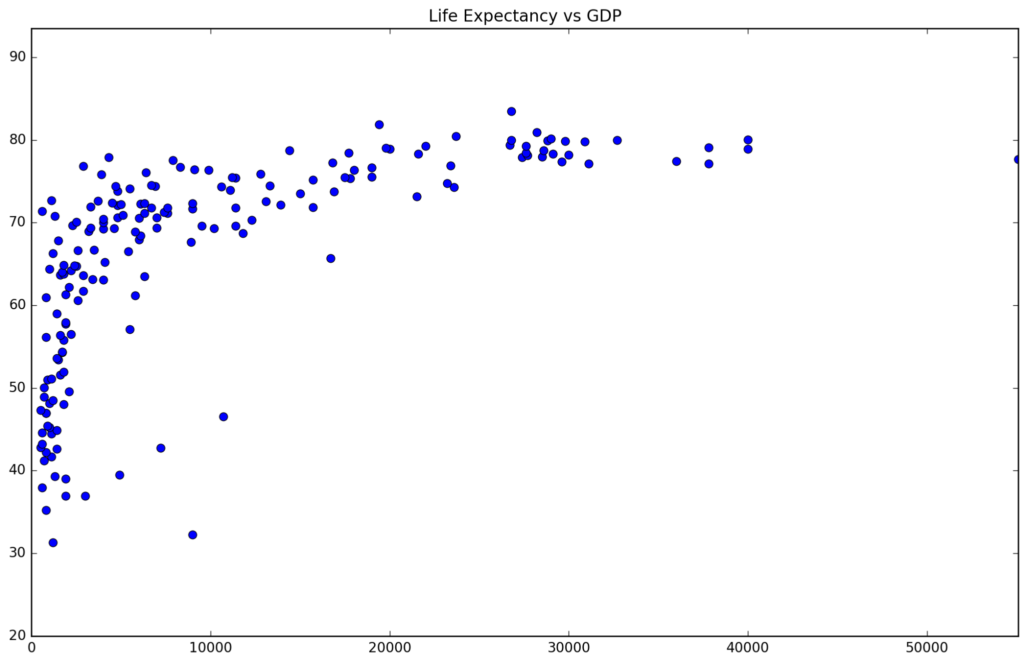

Figure 3.8: Life expectancy versus per capita GDP.

Note that we rescale the data values by dividing them all by the maximum for each column. These maximum values are obtained with the `np.max()` function, specifying the argument `axis=0` to indicate that we want the maximum values computed in the first dimension (the rows). The division of a matrix by a vector is broadcast on the last dimensions (which must match) and, in this case, will divide each row of `data` by the values in `scale`. This rescaling is advisable because large magnitude differences in values can cause instabilities in the polynomial regressions, especially at higher degrees. Now we test different degree polynomials and keep the one with the lowest validation error.

```
def mean_square_error(data,coefs):
    """Return mean squared error
       X on first column, Y on second column
    """
    pred = np.polyval(coefs,data[:,0])
    error = np.mean((data[:,1]-pred)**2)
    return error

best_err = 10000000 # very large number
for degree in range(1,9):
    coefs = np.polyfit(train[:,0],train[:,1],degree)
    valid_error = mean_square_error(valid,coefs)
    if valid_error < best_err:
        best_err = valid_error
        best_coef = coefs
        best_degree = degree

test_error = mean_square_error(test,best_coef)
print best_degree,test_error
```

The result is shown in Figure 3.9, representing the different polynomials.

Once we select the best hypothesis (in this case, the best polynomial with degree 3, with a validation error of 0.0150), we can estimate the true error for this hypothesis using the *test set*. Unlike the

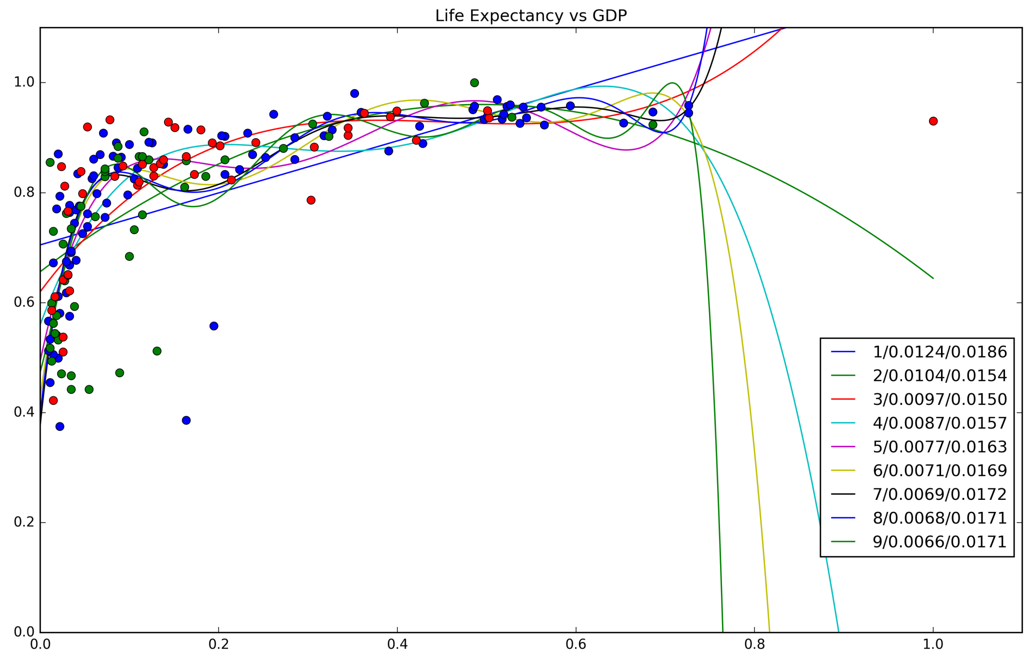

Figure 3.9: Evaluating different models. The training set is represented in blue, the validation set in green and the test set in red. The legend shows the degree of each polynomial and the training and validation errors.

validation error, the test error is an unbiased estimate of the true error because we are not using this value to select any parameter or model. Note, however, that these estimates depend on the random assignment of examples to training, validation and test, and the test error is an estimate of the true error.

Alternatively, we can use regularization with *ridge regression*. The `sklearn` library provides a class `Ridge` (in the `linear_model` module) for *ridge regression*. This is a linear regression solver, but since it is a multivariate regression solver we can expand our data set in order to obtain the same result as a polynomial fit.

First we import the libraries and define the `expand` function to expand the data matrix to a polynomial representation with the specified degree. Then we load the data, rescale the GDP values so they fall into the range [0..1], split into the training, validation and test sets and expand to a polynomial representation of degree 10. In this way, each of our points will have 10 features instead of one.

```
import numpy as np
from sklearn.linear_model import Ridge

def expand(data,degree):
    """expands the data to a polynomial of specified degree"""
    expanded = np.zeros((data.shape[0],degree+1))
    expanded[:,0]=data[:,0]
    expanded[:,-1]=data[:,-1]
    for power in range(2,degree+1):
        expanded[:,power-1]=data[:,0]**power
    return expanded

orig_data = np.loadtxt('life_exp.csv',delimiter='\t')
scale=np.max(orig_data, axis=0)
orig_data=orig_data/scale
data = expand(orig_data,10)
train, temp = random_split(data, 90)
```

```
valid, test = random_split(temp, 45)
```

The reason for rescaling is to avoid having numbers of very different orders of magnitude, because we are going to go up to the original values raised to a power of 10. This would cause instabilities in the numeric solver.

Now we try different values of the λ constant (which in the *ridge regression* algorithm is actually designated as α and called `alpha` in the `Ridge` class parameters). We use the `np.linspace()` function to give us a set of evenly spaced values between the minimum and maximum values given. By default, this function returns an array of 50 values, so that is the number of λ values we will try.

```
lambs = np.linspace(0.01,0.2)

best_err = 100000
for lamb in lambs:
    solver = Ridge(alpha = lamb, solver='cholesky',tol=0.00001)
    solver.fit(train[:,:-1],train[:,-1])
    ys = solver.predict(valid[:,:-1])
    valid_err = np.mean((ys-valid[:,-1])**2)
    if valid_err < best_err:
        # keep the best
```

If we plot the validation error as a function of the λ constant, we get something like what is shown in Figure 3.10. For more information on the `Ridge` class, consult the Scikit-learn documentation[2].

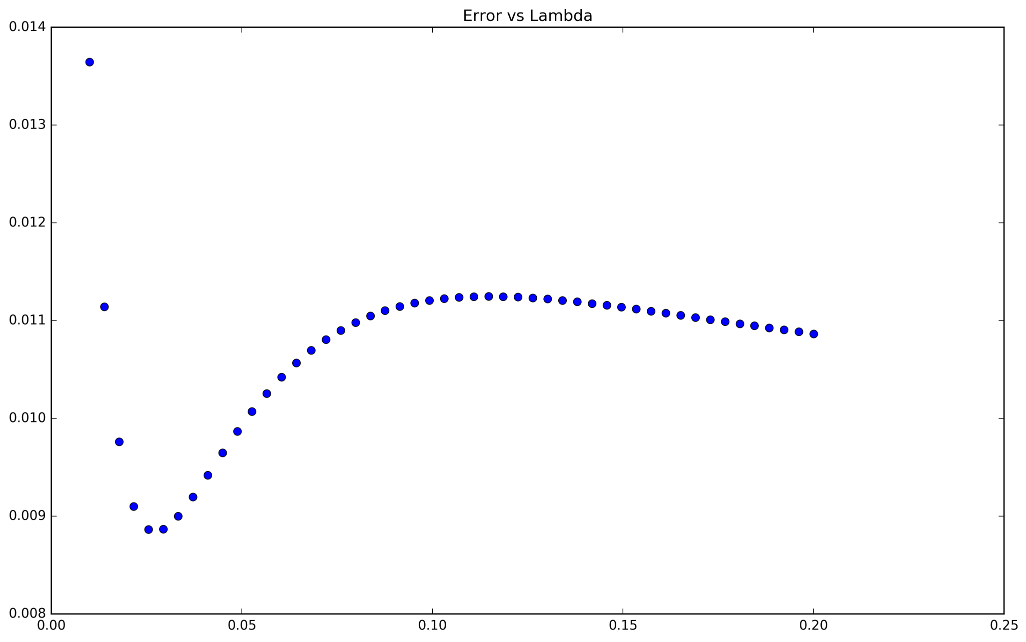

Figure 3.10: Plot of the validation error as a function of the λ constant.

3.6 Summary

The hypothesis that best predicts the target value given the feature vectors of examples from some universe of data is not necessarily the hypothesis that best fits the *training set*. As we improve the fit,

[2] http://scikit-learn.org/stable/modules/linear_model.html

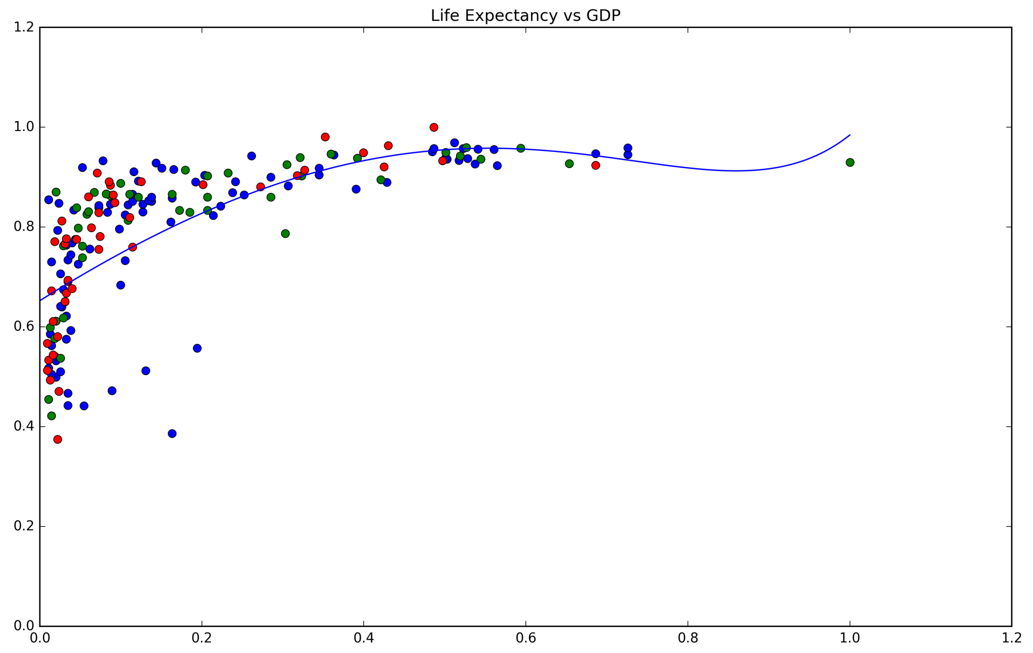

Figure 3.11: Plot of the best line found with the ridge regression. The training set is represented in blue, the validation set in green and the test set in red.

we run into the *overfitting* problem. To solve this, we saw that we can fit our models with a *training set* and use a *validation set* to select the hypothesis that minimizes the error estimated with the *validation set*. However, when we select an hypothesis based on the estimated error, this error estimate will no longer be unbiased, since it influenced our choice. To obtain a better estimate of the true error, we retain a *test set* that we only use for this final estimate.

Later in this course we will revisit *model selection* and explore more sophisticated and reliable ways of selecting the best model. The examples given in this chapter are meant only to illustrate the fundamental aspects of the *overfitting* problem and how to solve it.

3.7 Further Reading

1. Bishop [4], Section 3.1
2. Alpaydin [2], Section 2.6 through 2.8

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.