Chapter 6

Lazy Learning

Lazy vs Eager learning. K-Nearest Neighbours classification and regression. Kernel Density Estimation. Kernel Regression.

6.1 Lazy and Eager Learning

So far, we have approached all machine learning problems with the intent of finding a function that can predict the class or value of new points. This is called *Eager Learning*, a process in which the training data is used to fit some model and form a hypothesis that generalizes how the input features relate to the value to predict. In *Lazy Learning*, this step is delayed until the moment the system is queried. In this chapter we will see some examples of this approach.

6.2 Classification with K-Nearest Neighbours

A k-nearest neighbours classifier is an example of a *lazy learning* method. More specifically, an *instance learning* method, because the algorithm involves comparing new instances with instances in the training set. Keeping the labelled training set, the k-NN classifier will label a new point with the label of the majority of the k points in the training set that are closer to the new point. Figure 6.1 shows the tesselations of a 1-NN, 3-NN and 5-NN classifiers. For a 1-NN classifier, the new points are labelled with the label of the closest point in the training set, resulting in a *Voronoy tesselation*. For classifiers with more neighbours, the labels are determined by the majority label of the k nearest neighbours and the tesselation becomes more complex.

In all cases, the decision surface is piecewise linear, composed of the hyperplanes across which the nearest neighbours change. As the number of neighbours used increases, the classifier becomes less determined by local conditions. Figure 6.2 shows the decision frontiers for 1-NN, 13-NN and 25-NN classifiers.

To implement a k-NN classifier, we need to start by defining a distance function. For continuous numerical features we can use the *Minkowski distance*, or *p-norm*, which is given by

$$D_{x,x'} = \sqrt[p]{\sum_d |x_d - x'_d|^p}$$

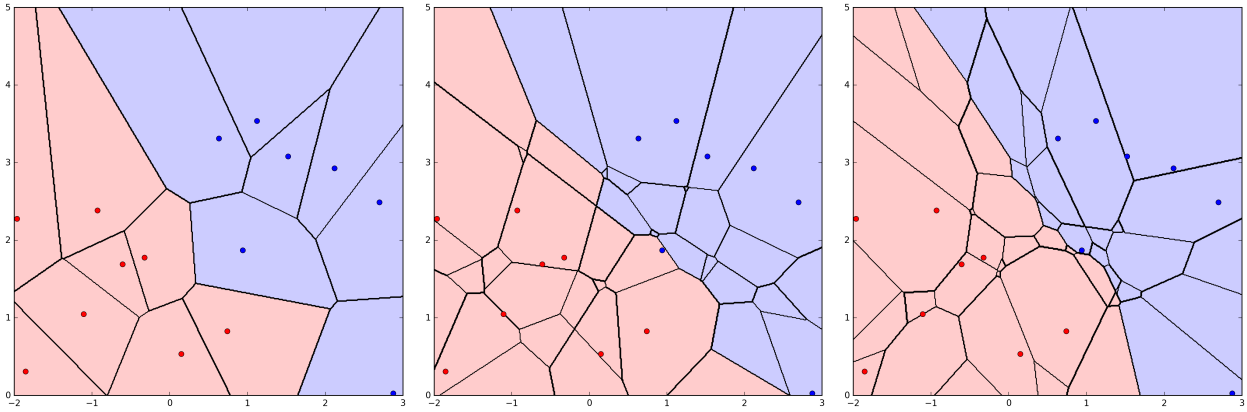

Figure 6.1: Tesselations for 1-NN, 3-NN and 5-NN classifiers.

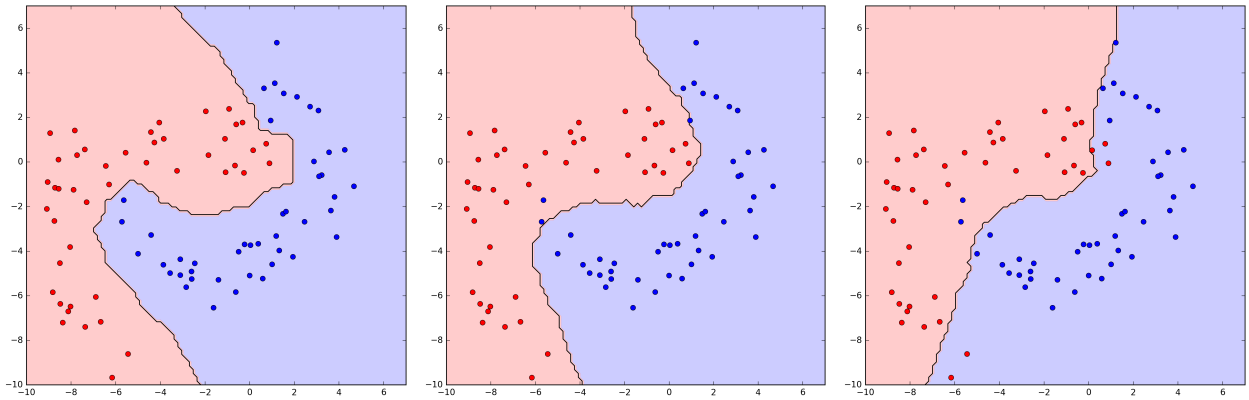

Figure 6.2: Comparison of 1-NN, 13-NN and 25-NN classifiers on the same data set.

Depending on the value of p, this corresponds to the Manhattan distance ($p = 1$) or Euclidean distance ($p = 2$). Other values of p result in different distance measures. For example, for p values between 0 and 1, similarities in one feature become more important. Figure 6.3 shows the effect of different p values.

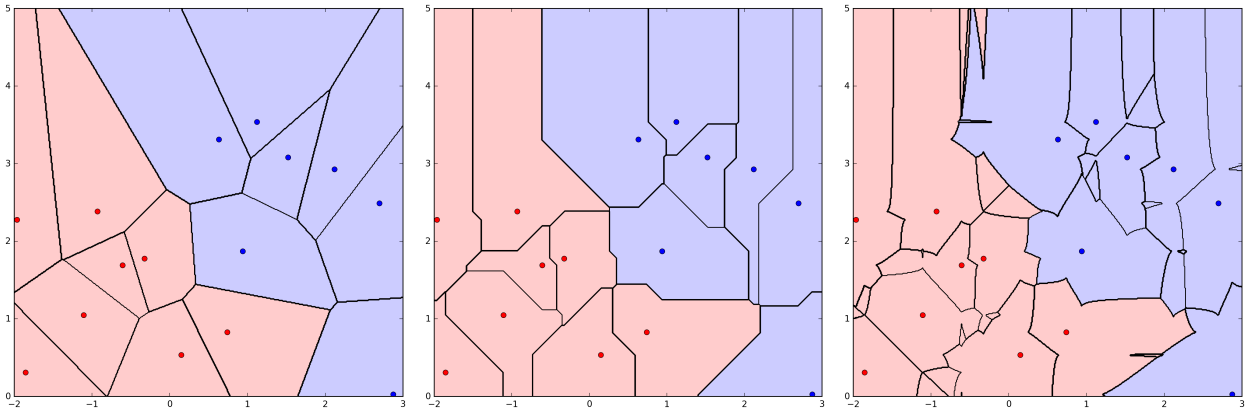

Figure 6.3: Comparison of three distance measures, $p = 2$, $p = 1$ and $p = 0.7$.

For categorical features, a good distance function is the *Hamming distance*:

$$D_{x,x'} = \sum_d \begin{cases} 1, & x_d \neq x'_d \\ 0, & x_d = x'_d \end{cases}$$

Since we are dealing with continuous numerical features, we can start by defining the Minkowski distance, or p-norm, with a default p value of 2, so that it defaults to the euclidean distance.

```
import numpy as np

def mink_dist(x, X, p = 2):
    """return p-norm values of point x distance to vector X"""
    sq_diff = np.power(np.abs(X - x),p)
    dists = np.power(np.sum(sq_diff,1),1.0/p)
    return dists
```

Now we create a function to list the nearest k neighbours in the training set given some example, and then the mode (the most common value) of the label in these nearest k neighbours. This is all we need to classify new data points given the training set X and respective labels Y.

```
from scipy.stats import mode

def k_nearest_ixs(x, X,k):
    """return indexes of k nearest neighbours
    """
    ixs = np.argsort(mink_dist(x,X))
    return ixs[:k]

def knn_classify(x,X,Y,k):
    """return class of x"""
    ix = k_nearest_ixs(x,X,k)
    return mode(Y[ix,0], axis=None)[0][0]
```

Depending on the data and features we have to deal with, it may be desirable to standardize or normalize the inputs. However, this will influence the distance measured between two points and has to be considered with some care. We should do this preprocessing only if we do not wish for features with a greater range of values to weigh more heavily on the distance function. This may often be the case but there can be exceptions. Suppose, for example, that we are dealing with geographical coordinates and want to predict some property of a point by looking up the properties of the neighbours. In that case we should not standardize our data because that would distort the distances by shrinking our data distribution in the direction it spreads the most.

6.3 Example of k-NN Classification

We can use cross-validation to determine the best k value. We load the data set, set aside a third of the points for testing, and then plot the training and validation error with 10-fold cross-validation. Figure 6.4 shows this process. The first panel shows the data set, the second panel the plot of the errors as a function of the k value and the third panel the best model obtained by cross-validation, with $k = 9$. Note that we also plotted the test error. However, we cannot use the test error to choose the model, otherwise the test error would no longer be an unbiased estimator of the true error.

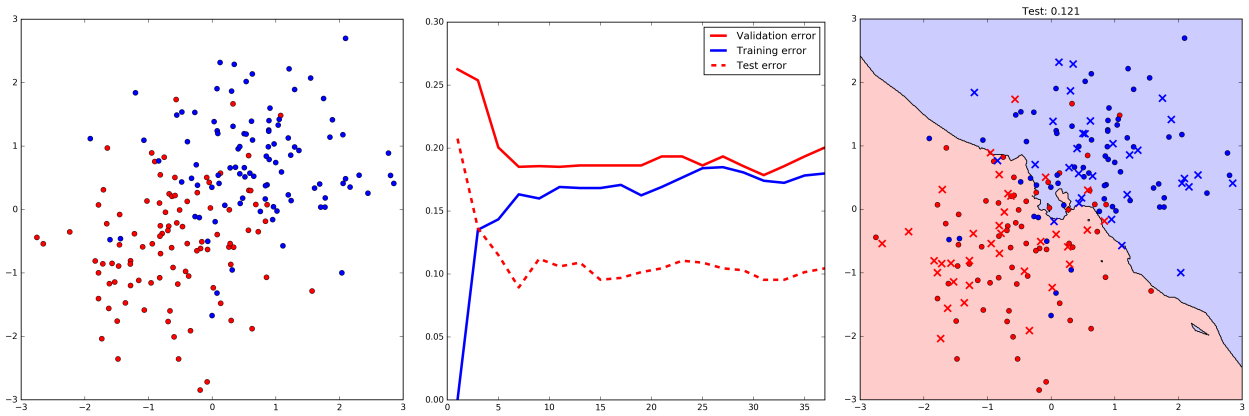

Figure 6.4: Finding the best k value for the k-NN classifier. The first panel shows the data, the second the training, validation and test error plot, and the last panel shows the result, with $k = 9$.

6.4 Curse of Dimensionality

The *curse of dimensionality* is a generic term for a set of problems that arise from dealing with data with many dimensions. In the case of distance-based methods, the problem of high dimensionality is that, with many dimensions, most points are at the frontier of any region. Figure 6.5 shows the proportion of an N-dimensional sphere occupied by another sphere whose diameter is 95

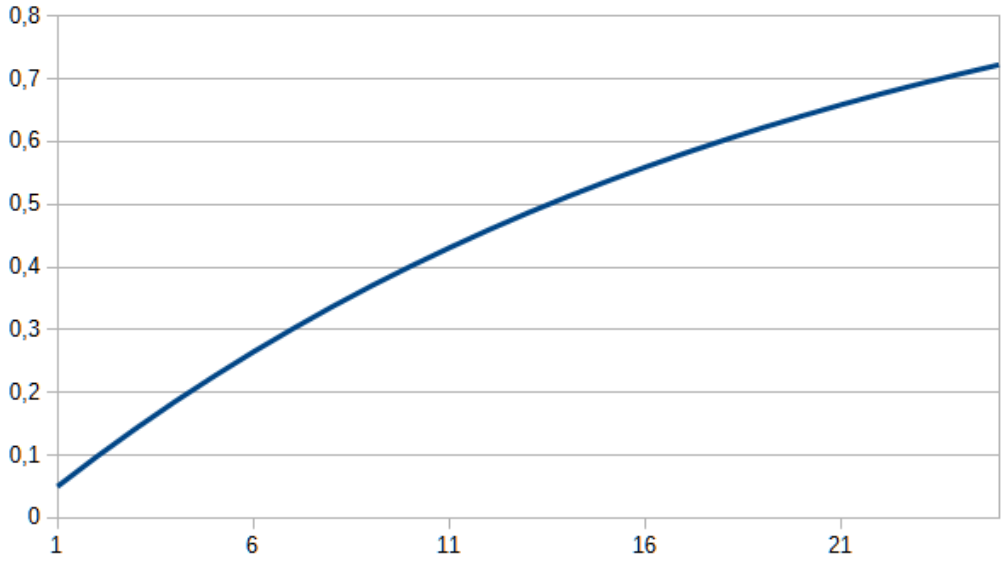

Figure 6.5: Fraction of region occupied by a frontier that is 5% of the diameter as a function of dimension.

6.5 Instance Based Regression

The k-NN approach can also be used for regression, making the predicted value equal to the average of the values of the k nearest neighbours. Figure 6.6 shows a regression curve using different values of k.

However, a better way to perform instance based regression is to use a continuous function that reduces the weight of points farther from the point of interest, and then estimate the desired value using a weighted average of these values. This is called *kernel regression*. The function that weighs

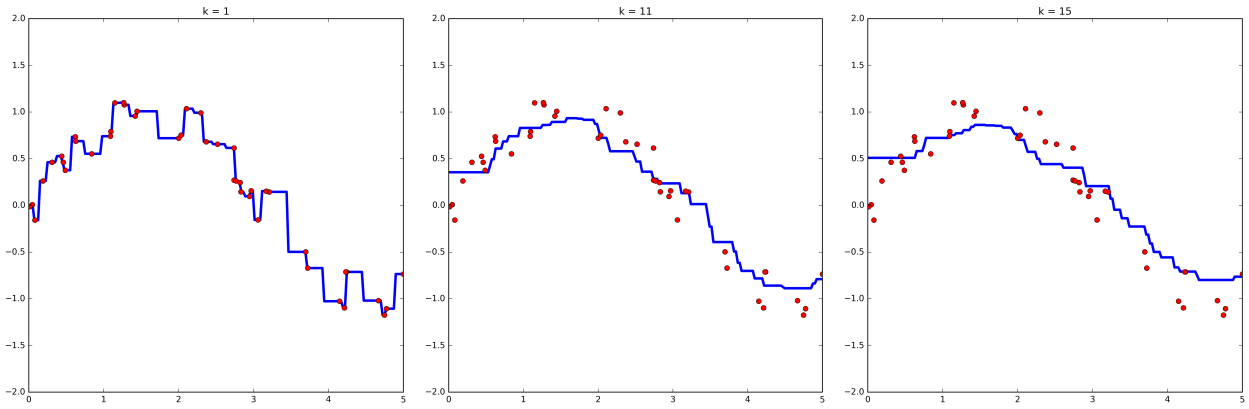

Figure 6.6: K-Nearest Neighbours regression with different values of k.

different points in the training set according to distance is a *kernel function*. A *kernel function* $K(u)$ is a function that satisfies these three conditions:

$$K(u) \geq 0 \ \ \forall u \tag{6.1}$$

$$\int_{-\infty}^{\infty} K(u)du = 1 \tag{6.2}$$

$$K(-u) = K(u) \ \ \forall u \tag{6.3}$$

An example of an often used *kernel function* is the *gaussian kernel*[1].

$$K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}}$$

Then we also need an estimator that predicts the value at x from some function of the y values in the data set weighted by the *kernel function*. For example, the *Nadaraya-Watson* estimator:

$$\hat{y}(x) = \frac{\sum_{t=1}^{N} K\left(\frac{x-x^t}{h}\right) y^t}{\sum_{t=1}^{N} K\left(\frac{x-x^t}{h}\right)}$$

or the *Priestley-Chao* estimator

$$\hat{y}(x) = \frac{1}{h} \sum_{t=2}^{N} (x^t - x^{t-1}) K\left(\frac{x - x^t}{h}\right) y^t$$

For example, we can implement the *gaussian kernel* and the *Nadaraya-Watson* estimator:

```
1 def gaussiank(u):
2   k=np.e**(-0.5*u**2)/np.sqrt(2*np.pi)
3   return k
4
5 def nad_wat(K, h, X, Y, x):
6     num = 0
7     den = 0
8     for ix in range(len(X)):
```

[1] A list of common kernel functions can be found on Wikipedia: https://en.wikipedia.org/wiki/Kernel_(statistics)

```
        yf = Y[ix]
        u = (x-X[ix])/h
        k = K(u)
        den = den + k
        num = num + yf * k
    return num/den
```

And then use them to compute the regression curve from the data, as shown in Figure 6.7.

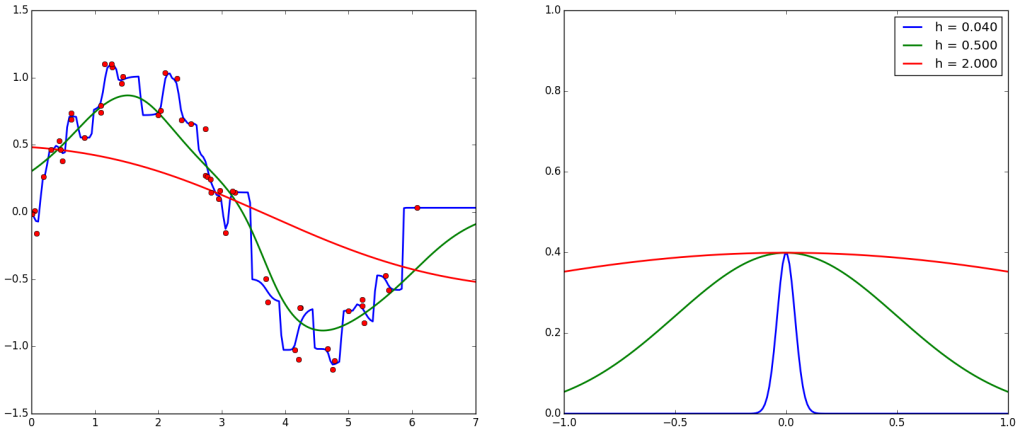

Figure 6.7: Kernel regression with a *gaussian kernel* and a *Nadaraya-Watson* estimator. The three lines show the effect of three different values of the parameter h.

6.6 Kernel Density Estimation

Kernel functions can also be used to smooth density estimates. Given a distribution of points, for example sampled from a normal distribution as shown in the left panel of Figure 6.8, we can depict the varying density of the points using histograms. However, histograms are discontinuous and the result is very dependent on bin size. An alternative is to apply a kernel function to each point and then sum them all. This, shown on the right panel, and leads to a much smoother estimate.

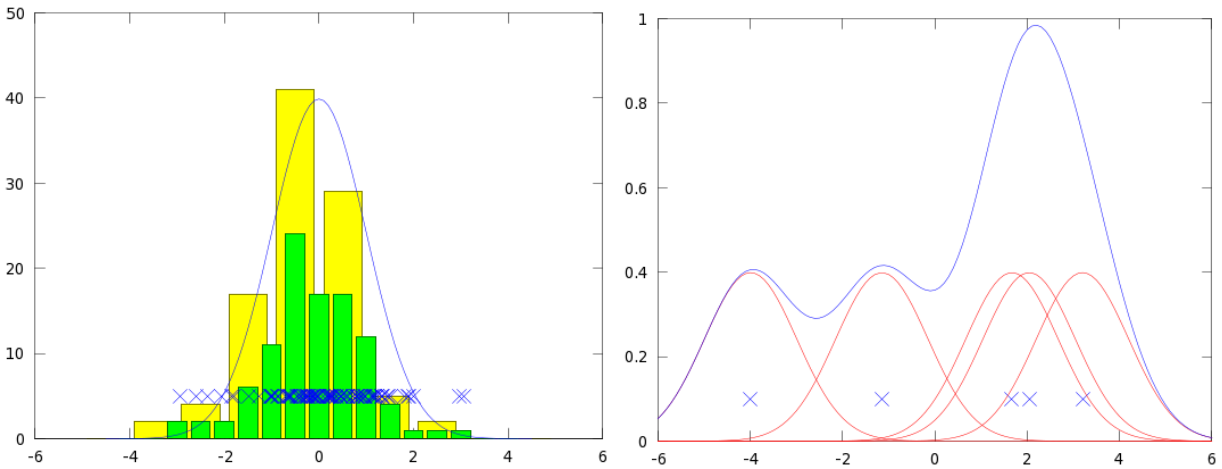

Figure 6.8: Kernel density estimation.

6.7 Summary

In this chapter we saw *lazy learning*, in which inference from the data is delayed until the moment the system is queried, in contrast with *eager learning*, which we covered before, and which involves first training a model of the data. K-nearest neighbours is a *lazy learning* technique for predicting values of new points based on the neighbouring points of the training data, for some distance function. Finally, we covered kernel regression and density estimation.

6.8 Further Reading

1. Alpaydin [2], Sections 8.1 through 8.4
2. Mitchell [18], Sections 8.1 and 8.2
3. Marsland [17], Section 8.4.

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.