Chapter 15

Introduction to Unsupervised Learning

Introduction to unsupervised learning. Data visualization and feature selection.

15.1 Unsupervised Learning

Unsupervised learning, simply put, is the process of adjusting some model to the structure of the data without relying on an error measure evaluated with reference to known labels. This does not mean the data cannot have known labels. It is quite possible, and often useful, to use unsupervised learning with labelled data. But, unlike supervised learning, the goal of unsupervised learning is simply to describe aspects of the data — how it is distributed, relations between features and so forth — instead of trying to predict some value that is known for the training set and which can be used to supervise learning. In other words, the goal of unsupervised learning is to transform the data into some representation that makes it easier to understand it or use it for some other purpose, like supervised learning. Figure 15.1 illustrates this process.

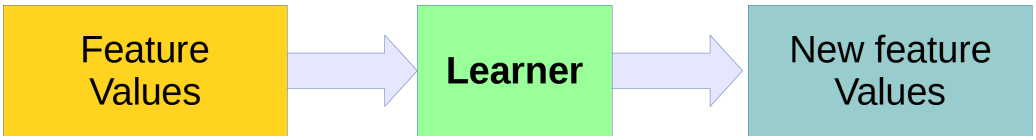

Figure 15.1: Diagram representing unsupervised learning.

Although, strictly speaking, data visualization by itself may not include unsupervised learning, as it may not involve learning at all, it is a good starting point to understand the purpose of unsupervised learning. So we shall begin with the problem of visualizing data with more than two dimensions.

15.2 Visualizing Data

We shall consider, as an example, the Iris dataset, first introduced by Fisher in 1936 [1]. Figure 15.2 shows examples of the three classes of flowers. Each flower is described by four features, the length and width of sepals and petals.

[1]The dataset can be downloaded from the MIST repository: https://archive.ics.uci.edu/ml/datasets/Iris

Figure 15.2: The Iris data set contains values for sepal and petal lengths and widths for flowers of three Iris species. Images CC BY-SA. Authors Setosa: Szczecinkowaty; Versicolor: Gordon, Robertson; Virginica: Mayfield.

The problem is how to visualize this four-dimensional data, since we can only understand three dimensions and, for practical purposes, two-dimensional representations are preferable. For this purpose, we will use the Python Data Analysis (Pandas) library, since it includes many convenient features for image visualization [2].

We begin by reading the `.csv` data file using the `read_csv` function from the Pandas library and then plot the histograms of the features in a single figure. This is the file format:

```
SepalLength,SepalWidth,PetalLength,PetalWidth,Name
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
...
5.1,2.5,3.0,1.1,Iris-versicolor
5.7,2.8,4.1,1.3,Iris-versicolor
...
6.2,3.4,5.4,2.3,Iris-virginica
5.9,3.0,5.1,1.8,Iris-virginica
```

This is the code for creating the histogram.

```
from pandas import read_csv
import matplotlib.pyplot as plt

data = read_csv('iris.data')
data.plot(kind='hist', bins=15, alpha=0.5)
plt.savefig('L15-stackedhist.png', dpi=200,bbox_inches='tight')
plt.close()
```

Alternatively, instead of using the `plot` method of the class returned by the `read_csv` function to plot the histogram of all features in the same chart, we can plot the different histograms separately by using the `hist` method instead:

```
...
data.hist(color='k', alpha=0.5, bins=15)
...
```

The resulting images can be seen in Figure 15.3

[2] More information available on http://pandas.pydata.org/pandas-docs/stable/visualization.html

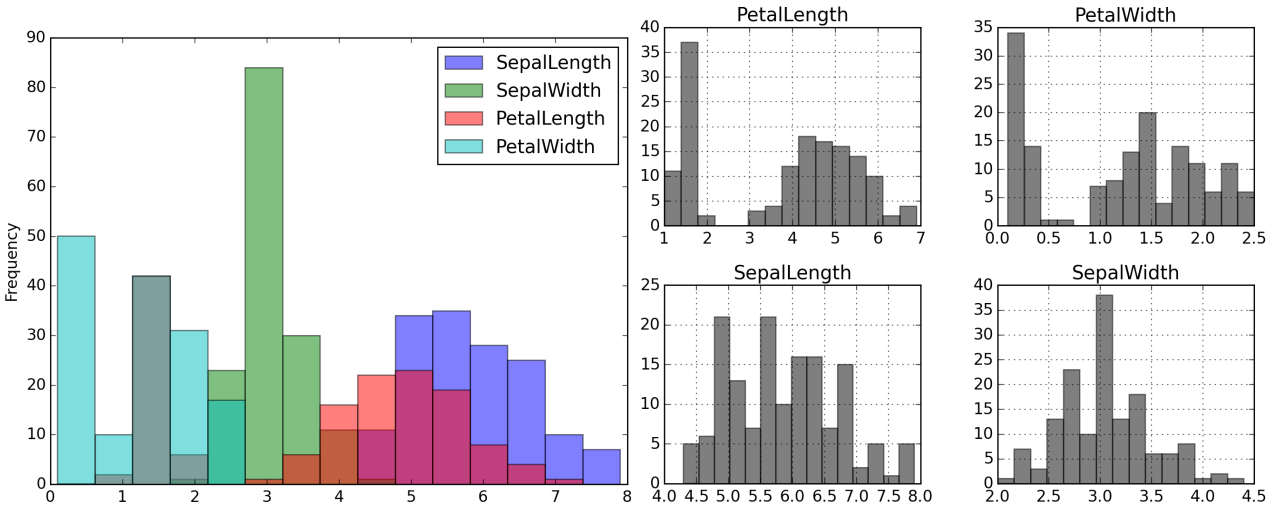

Figure 15.3: Examples of histograms for the Iris dataset. On the left, histograms of the four features in the same chart. On the right, separated in four charts.

Another way of visualizing the distribution of each feature is using a *box plot*. In this type of plot, the box represents the range between the first and third quantiles (25% and 75%), a line represents the median value and the "whiskers" are placed away from the first and third quantile values at a distance equal to the difference between these two quantile values multiplied by a constant parameter, often 1.5. We can do this with the Pandas library by using the `kind='box'` argument on the plot method. The result is shown in Figure 15.4.

```
5 ...
6 data.plot(kind='box')
7 ...
```

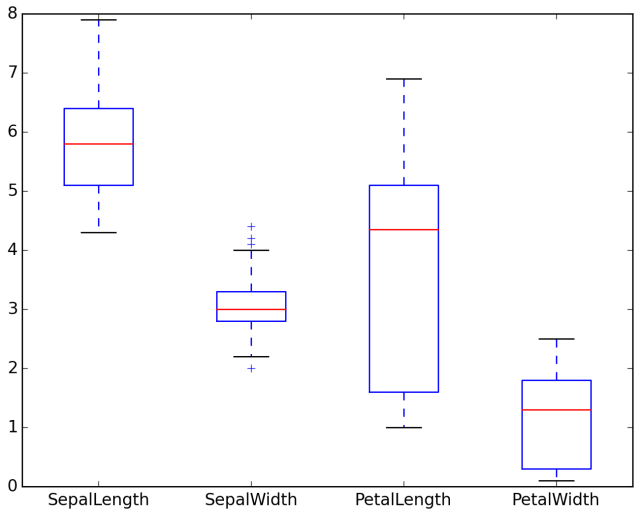

Figure 15.4: Box plot of the four features for the Iris dataset.

While histograms and box plots are useful to represent the distribution of each isolated feature, they give us no idea about how features correlate. One alternative plot to visualize correlations between pairs of features is the *scatter matrix* plot. This is a two-dimensional array of plots representing the histogram or kernel density estimation plot of each feature in the diagonal and scatter plots of each feature as a function of another in the remaining plots. Figure 15.5 illustrates these plots for the four

features in the Iris dataset. We can do this easily with the Pandas library sith the `scatter_matrix` function:

```
1 from pandas import read_csv
2 import matplotlib.pyplot as plt
3 from pandas.plotting import scatter_matrix
4 data = read_csv('iris.data')
5 scatter_matrix(data.ix[:,[0,1,2,3]], alpha=0.5,figsize=(15,10), diagonal='kde')
6 plt.savefig('L15-scatter.png', dpi=200,bbox_inches='tight')
7 plt.close()
```

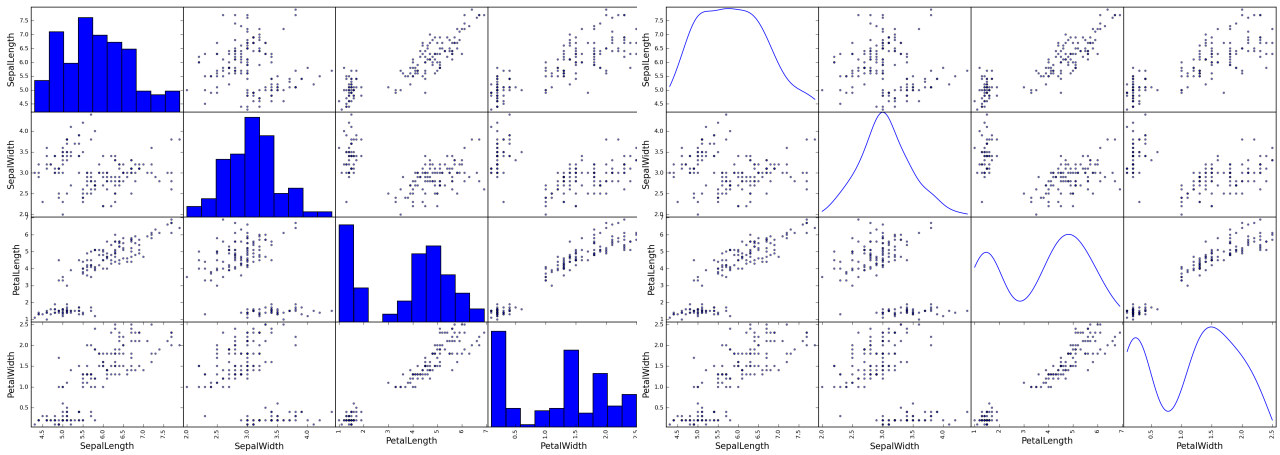

Figure 15.5: Scatter matrix plot examples, with histograms and Kernel Density Estimation in the diagonals.

Another useful method for visualizing multidimensional data is the *parallel coordinates plot*, in which features are represented as a set of parallel axes and each data point is represented as a set of line segments intersecting the feature axes at the corresponding values for each feature. The code below shows how we can do this to plot all the Iris data in single category by adding a new column, titled `'All'`, in which all points have the value `'Iris'`. Then we do the `parallel_coordinates` plot using this new column as the category field.

```
1 from pandas import read_csv
2 from pandas.plotting import parallel_coordinates
3 import matplotlib.pyplot as plt
4 data = read_csv('iris.data')
5 all_data=data.ix[:,[0,1,2,3]]
6 all_data['All']='Iris'
7 parallel_coordinates(all_data,'All',color='b')
8 plt.savefig('L15-parallel-all.png', dpi=200,bbox_inches='tight')
9 plt.close()
```

Alternatively, we can plot the different categories in different colors by using the `'Name'` column for the category and giving three color labels in the `color` argument of the `parallel_coordinates` function. The result is shown in Figure 15.6.

```
1 ...
2 parallel_coordinates(data, 'Name', color=('r','g','b'))
3 ...
```

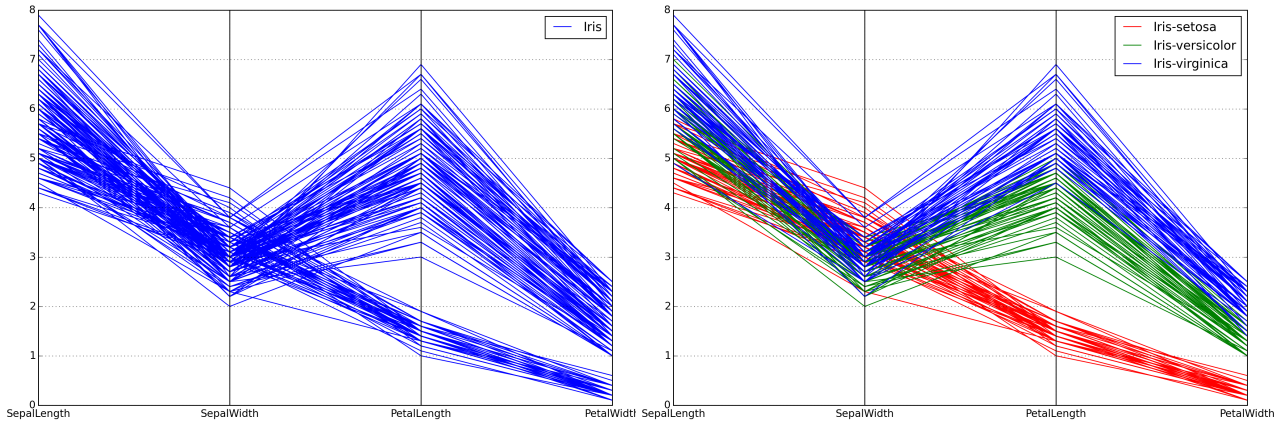

Figure 15.6: Parallel coordinates plot. Each point is represented as a line crossing the feature axes at the respective feature values. In the right panel, the lines are coloured by class.

A similar method is to use *Andrew's curves* [3]. In this plotting method, each data point is converted into a line resulting from the sum of trigonometric terms of different frequencies. Given a data point $\vec{x} = \{x_1, x_2, x_3, ...\}$, the resulting line is:

$$f_{\vec{x}}(t) = \frac{x_1}{\sqrt{2}} + x_2 \sin(t) + x_3 \cos(t) + x_4 \sin(2t) + x_5 \cos(2t) + x_6 \sin(3t) + x_7 \cos(3t)...$$

The result is that different features contribute to different frequencies on the curve, and points with similar features result in similar curves. With the Pandas library, these curves can be plotted using the `andrews_curves` function:

```
from pandas import read_csv
import matplotlib.pyplot as plt
from pandas.plotting import andrews_curves
data = read_csv('iris.data')
andrews_curves(data, 'Name', color=('r','g','b'))
plt.savefig('L15-andrews.png', dpi=200,bbox_inches='tight')
plt.close()
```

The *Radial Visualization* (RADVIZ) method [13] represents each multidimensional data point as a point in two dimensions, but places the points by spreading the feature axes radially and using the value of each feature to "pull" the point in the corresponding direction. The position of the point results from the outcome of all these "forces" pulling it in different directions. This way, points that have a balanced distribution of values across the features tend to be in the middle of the plot, whereas points that favour some feature over the others are pulled by that feature's axis. This can be done with the `radviz` function of the Pandas library. The Andrew's curves and RADVIZ plots are shown in Figure 15.7.

```
from pandas import read_csv
import matplotlib.pyplot as plt
from pandas.plotting import radviz
data = read_csv('iris.data')
radviz(data, 'Name', color=('r','g','b'))
plt.savefig('L15-radviz.png', dpi=200,bbox_inches='tight')
plt.close()
```

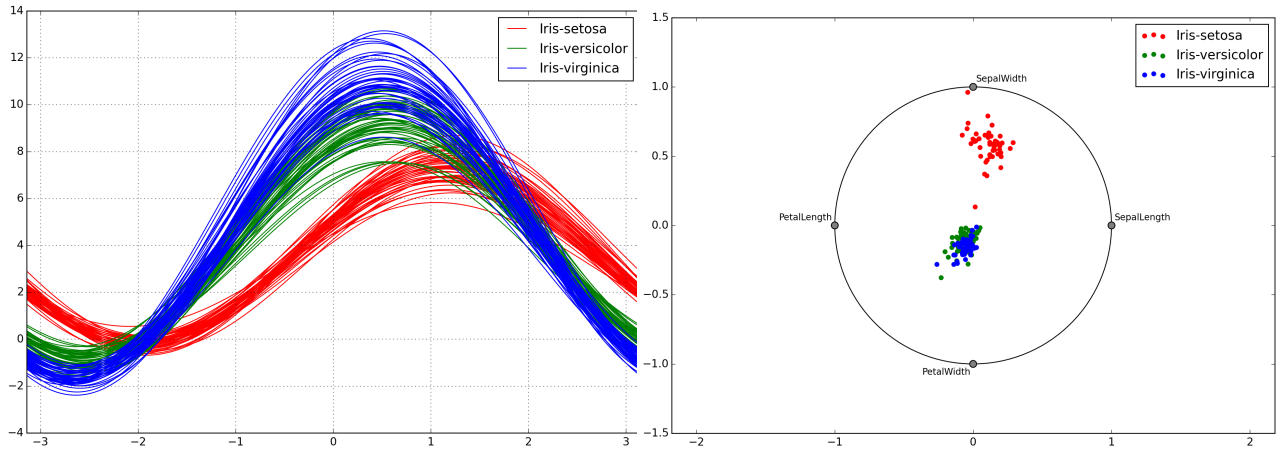

Figure 15.7: Andrews curves and RADVIZ plot. Data coloured by class.

15.3 Feature Selection

In general, not all features are equally useful for learning and it may be beneficial to reduce the dimensionality of the training set, both for supervised and unsupervised learning. There may be several reasons for this. There may be too many features for the available data, leading to overfitting; some features may be too noisy or uninformative; some features may be costly to measure and so forth. One way of reducing the number of features is to discard all but the best. This is *feature selection* and can be done by examining and discarding features before the learning process, or according to the performance of the hypotheses learned or even as an integral part of the learning process. Discarding features before beginning to train the learner is called filtering, an can be either *univariate filtering* if the features are discarded by examining each feature individually or *multivariate filtering* if features are examined jointly with other features.

Univariate filtering is easier to understand when we are dealing with labelled data and want to prepare the data for supervised learning. In this case, we can select features by comparing each feature with the data labels. One criterion for selecting features in this case can be the statistical independence of each feature and the class, since features that are statistically independent from the class are not useful for predicting the class. Statistical independence can by assessed by the χ^2 (chi-squared) test, a generic test that gives us the probability of obtaining some sample when drawing at random from some distribution. If O_n are the observed frequencies and E_n the expected frequencies, the chi-squared value is:

$$\chi^2 = \sum_{i=1}^{N} \frac{(O_i - E_i)^2}{E_i}$$

If we have a feature with K categorical values and a classification problem with C classes, we can compute the observed number of cases where the feature has a value k in points with class c, O_{kc}, and the expected number E_{kc} assuming the feature and class are independent, which is obtained from the fraction of value k and class c. In this case, the chi-squared value (for $(K-1)(C-1)$ degrees of freedom) is:

$$\chi^2 = \sum_{k=1}^{K} \sum_{c=1}^{C} \frac{(O_{kc} - E_{kc})^2}{E_{kc}}$$

Using the chi-squared test we can eliminate those features that, having a low χ^2 value, are closer to being statistically independent of the class.

Another statistical test for labelled data is the *Analysis of Variance* (ANOVA) F-test, which compares the variance between groups with the variance within the groups. Again, this proportion has a known probability distribution under the assumption that the variables are independent, and thus can be used to find the likelihood of that assumption. If the F-test value is low, and thus the likelihood of independence is high, we can reject the feature as uninformative. The code below shows how to use the ANOVA F-test with Scikit-Learn library, on the Iris dataset:

```
1 from sklearn.feature_selection import f_classif
2 from sklearn import datasets
3
4 iris = datasets.load_iris()
5 X = iris.data
6 y = iris.target
7 f,prob = f_classif(X,y)
8 print f
9 print prob
```

The F-test values and respective probabilities indicate which features deviate the most from being independent of the class (those with the smallest probability values). Figure 15.8 shows the scatter plot of the two best features from the Iris dataset, according to the F-test, which are the two features with the lowest F-test probabilities.

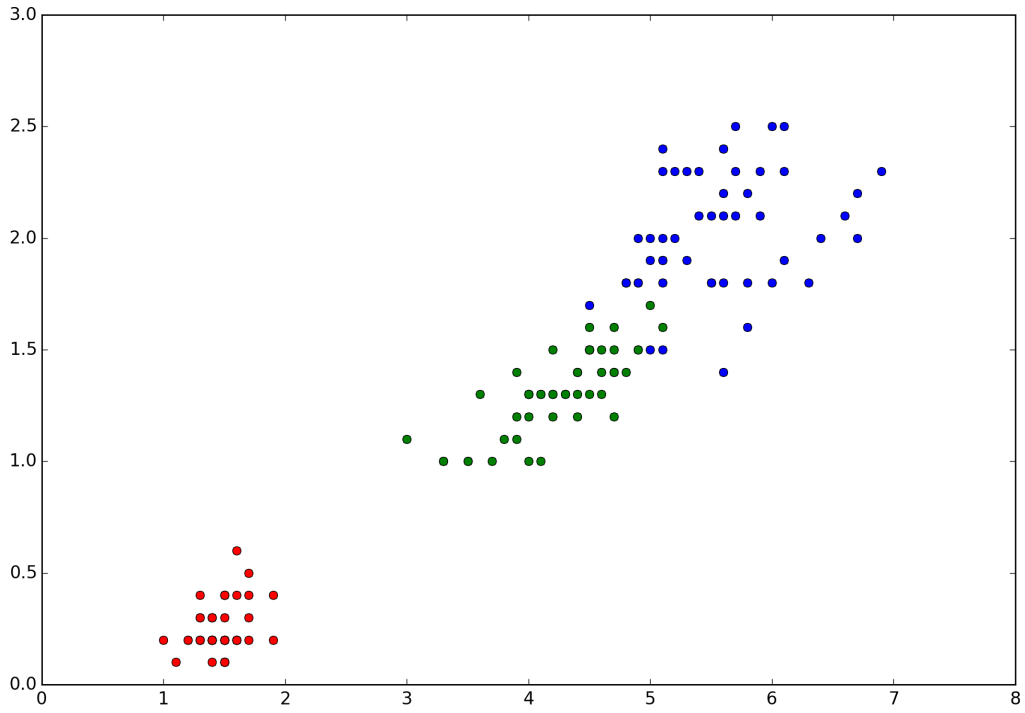

Figure 15.8: Scatter plot of the two best features (petal length and petal width) according to the ANOVA F-Test.

These methods rely on labelled data, and determine the relevance of each feature for predicting the labels. A feature is *relevant* if it correlates to the labels, and irrelevant if it is independent of the labels, in which case we discard it. But we can also filter features according to their correlation to other features, because a feature is *redundant* if it correlates to another features. This requires filtering features by comparing them to each other, which is called *multivariate fitering*. In this approach, if several features are strongly correlated one to the others, we can discard all but one of the set, since the

information given by that one is nearly the same as that given by all other correlated features. Since this can be applied both to labelled and unlabelled data sets, it can be more useful for unsupervised learning.

Instead of filtering features prior to training, we can also use the performance of the trained hypotheses to evaluate the adequacy of the set of features used. These are called *wrapper methods* for feature selection, and consist of a scoring algorithm, typically the machine learning algorithm we wish to use, and a search algorithm that runs the learning algorithm on subsets of all features to find the best subset. For this we can use a *deterministic wrapper* that iterates through all possible subsets. With *sequential forward selection* we start with the empty set and, at each iteration, loop through all remaining features to find the best one to add to that set, according to the performance of our learning algorithm. This is repeated until we reach the desired number of features or performance level. With *sequential backward elimination* we do the search in the opposite direction, starting with all features and removing, at each iteration, we eliminate one feature so that the performance of the classifier is maximized.

Alternatively, we can also use a *non-deterministic wrapper* that searchs the subsets of features with non-deterministic algorithms such as genetic algorithms or simulated annealing. The procedure is the same, trying to maximize the performance with a limited number of features, but without the greedy search of the deterministic wrapper methods.

Finally, some learning algorithms incorporate feature selection. This is called *embedded feature selection*. Decision trees with a limited depth are an example of this kind of algorithm, since the best features are used earlier in the tree and, by limiting the tree depth, less useful features end up being ignored. Naïve Bayes with weighted features is another example. For example, features may be weighted according to how much the conditional distribution of the feature values given a class differs from the prior probability of the class, which indicates more relevant features [14]. Embedded feature selection can also be done through regularization. For example, using L1 regularization, which penalizes the sum of the absolute values of the parameters. This forces some parameters to be 0, effectively ignoring the corresponding features. Logistic Regression in Scikit-Learn can be done with L1 regularization.

15.4 Further Reading

1. Pandas library visualization tutorial: `http://pandas.pydata.org/pandas-docs/stable/visualization.html`
2. Scikit-Learn feature selection tutorial: `http://scikit-learn.org/stable/modules/feature_selection.html`
3. Alpaydin [2], Sections 6.2. and 6.9
4. A review of feature selection techniques in bioinformatics [19]

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.