Machine Learning

by Jake Sauter

Introduction

- Machine learning refers to a set of topics dealing with the creation and evaluation of algorithms that facilitate pattern recognition, classification, and prediction, based on models derived from existing data [4]
- During supervised learning, objects are classified using a set of features. The classes are known in advance and the goal is to build a model that can extrapolate from the labelled examples to successfully classify new samples or predict the next continuous value
- During unsupervised learning, unlabelled data are examined with the goal of discovering similarities between objects

Applications

- A perceptron was employed to define criteria for start sites in *E. coli.* during the early work on the analysis of translation initiation
- Gene expression data has successfully been used to classify patients in different clinical groups and identify new disease groups
- Genetic code allowed for prediction of the protein secondary structure
- Continuous variable prediction with machine learning algorithms have been used to estimate bias in cDNA microarray data

Definitions

- If only a few labelled samples are available in a data set, then semi-supervised learning can be employed in which class labels of known samples are imposed on the most similar samples to them in feature space
- Machine learning problems can be divided into three different categories
  - Class prediction
  - Class comparison
  - Class discovery

Class Prediction

- In a class prediction problem, classes are defined in advance and labelled data is available for each class. Each data point is usually a vector of values for a number of features

$$example_{ci} =< val(feature_1), val(feature_2), \cdots, val(feature_n) >$$

- The goal of a class prediction problem is to build a classifier which will be able to assign a previously unseen input vector and correctly assign it to the class it belongs to

Class Comparison

- In a class comparison problem, the classes are still predefined such as in a class prediction problem, however now the goal is to find **all** the features that distinguish the classes
- Class comparison problems serve the role of classical discriminant analysis

Class Discovery

- In a <u>class discovery</u> problem, the classes are not known in advance
    - Input vectors are unlabelled
- The goal of the class discovery problem is to identify the input vectors that share certain features, <u>essentially clustering them</u>

Relatedness of Problem Type

- Class prediction problems require a feature selection step, in which useful discerning features of the input vectors are selected to be used in the creation of the classifier
  - This is very similar to the class comparison task however is to only to identify the features needed to successfully discern the classes
  - Ex. The only feature needed to discern Kubota tractors from John Deere tractors is the color, while if one was interested in comparing the two types of tractors before a purchase many more factors would be taken into consideration [4]

Relatedness of Problem Type

- The class discovery task is very different from both previous problem types, as the classes are not known in advance, and in most cases different classes or clusters can be obtained from the same data by applying different methods
- It is important that the type of problem is identified in advance of exploring solutions, as solutions to the different problem types invoke very different machinery

Supervised Learning

- Assume that we wish to classify a collection of $i = 1, \ldots, n$ objects into K predefined classes
  - For example distinguishing different types of tumors based on gene expression values
- A classifier $C(x)$ may be viewed as K discriminant functions $g_c(x)$ such that the object with feature vector x will be assigned to class c for which $g_c(x)$ is maximized over class labels c in $\{1, \ldots, K\}$
- The feature space X is thus partitioned by the classifier $C(x)$ into K disjoint subsets

Supervised Learning

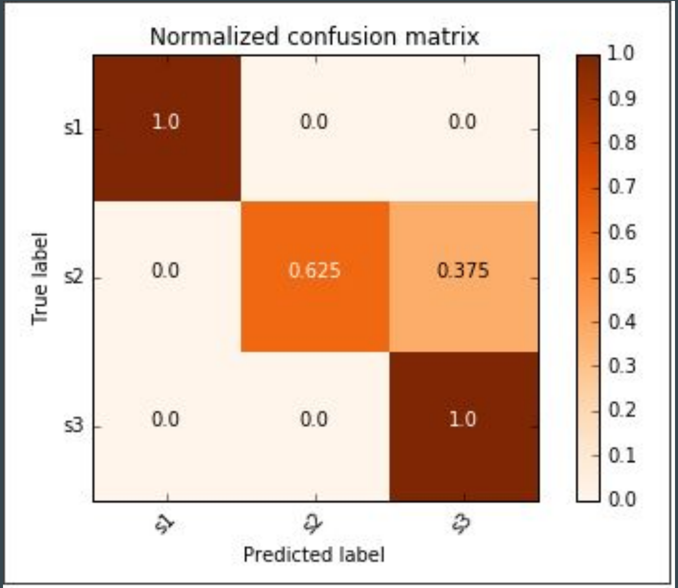
- Two main approaches to the identification of the discriminant functions $\boldsymbol{g_c(x)}$ are
  - Assume knowledge of the underlying class conditional probability density functions and assign $g_c(x) = f(p(x \mid y = c))$ where ***f*** is a monotonically increasing function such as ln
  - Class boundaries can also be directly estimated without explicitly calculating the probability density functions
- Intuitively the probability based classifier will classify object *x* to its most probable class, though in practice $\boldsymbol{p(x \mid y = c)}$ is unknown and must be estimated from the training set
  - Both parametric and nonparametric methods for density estimations can be used for this purpose

Supervised Learning

- Concerning probability density based classifiers
  - Parametric approach: **linear and quadratic discriminants**
  - Nonparametric approach: **k-nearest neighbors**
- Concerning classifiers that directly estimate class boundaries
  - **Decision trees, support vector machine, neural networks**

Error Estimation and Validation

- In order to assess how well our classifier is doing, we must first discuss some metrics and how they can be applied
- We have previously discussed a **confusion matrix** for the two class case, though we will show what this would look like for the three class case



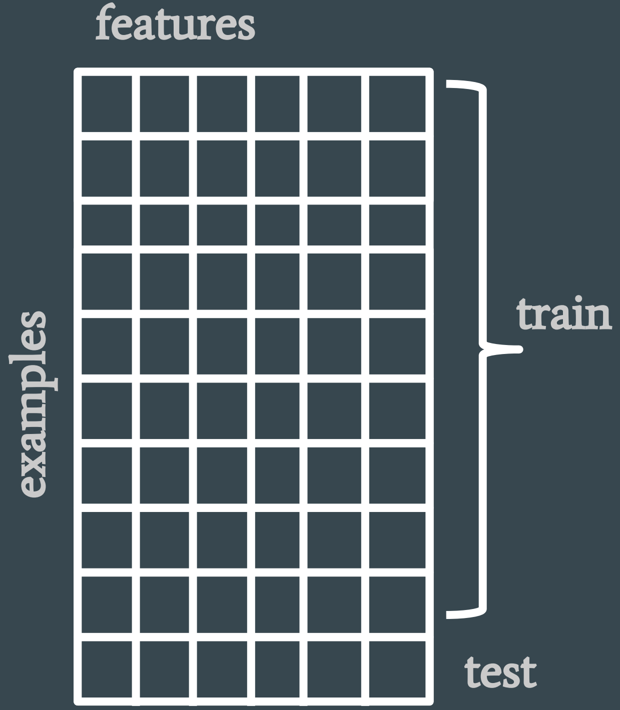
[1]

Error Estimation and Validation

- The goal behind developing a classifier is for use in predicting the true class of new samples, so determining the classification accuracy of the model using the same samples that were trained us will not be informative
- A better method of evaluating the effectiveness of ***C(x)*** would be **hold out** a portion of the given labelled data, and to use this to compose a **testing / validation set**
- This is a very effective testing procedure, though to achieve the best possible classifier we must use all of the provided data

Error Estimation and Validation

- Since we wish to use all possible data for building the model, but also need to evaluate how well the model will do on new data, we can implement **n-fold cross validation**
  - In this method, the data set is divided into **n-folds** with training and testing occurring n times
  - On each run, ***C(x)*** will be trained with n-1 folds and tested on the held out data
  - During each of the n runs, performance measures are calculated and <u>at the end these performance measures are averaged</u> to estimate how well the model will generalize when train with all n folds



Error Estimation and Validation

- In some situations the data may even be too scarce for n-fold cross validation and **leave-one-out (LOO) cross validation** must be implement
    - This is nothing but n fold cross validation with n being equal to the number of data points
    - The error obtained from LOO cross validation has a low bias but may have a large variance

Error Estimation and Validation

- For some problems, the number of samples may be very different between the ***K*** classes and the data is said to be **unbalanced**
- In these situations it is best to make sure that the training set is balanced as many classifiers will favor the richer data set
    - If the ratio of examples in each category is different between the training set and the testing set, the training and testing sets are said to be **stratified**

Feature Selection

- A common mistake particularly to this field is to try to build a model with too many features and not enough examples. In general there should be many more examples than features
  - About 10x more examples than features is a generally a desired ratio
- Thus in microarray experiments with thousands of features and tens of examples, **feature selection** is necessary
  - Wrapper methods: training on different sets of features and determining the best set
  - Filter Methods: Test for mutual information using correlation coefficient or statistical significance tests

Quadratic and Linear Discriminants

- **Quadratic Discriminants** is a standard classification approach applicable to continuous features and assumes that for each class ***c***, ***x*** follows a multivariate normal distribution with mean $\boldsymbol{\mu}_c$ and variance $\boldsymbol{\sigma}_c$
- Using the multivariate normal probability density function and replacing the true class mean and covariance matrices with sample derived estimates ($\boldsymbol{m_c}$ and $\boldsymbol{s_c^2}$ respectively), the discriminant function can be computed as

$$g_c(x) = -(x - m_c)\hat{\sigma_c^{-1}}(x - m_c)^T - log(|\hat{\sigma_c}|)$$

$$m_c = \frac{1}{n_c}\sum_{i=1}^{n_c} x_i \qquad \hat{\sigma_c} = \frac{1}{n_c}(x_i - m_c)(x_i - m_c)^T$$

Quadratic and Linear Discriminants

- The discriminant functions $g_c(x)$ are monotonically related to the densities $p(x|y = c)$, yielding higher values for larger densities
- The values of the discriminant functions will differ from one class to another only on the basis of the estimation of the class mean and covariance matrix
- After the formation of the discriminant functions, a new object z will be classified to the class for which the discriminant function is largest
- This classification approach produces nonlinear (quadratic) class boundaries, giving the name **quadratic discriminant rule** or **Gaussian classifier**

Quadratic and Linear Discriminants

- An alternative to the quadratic classifier is to assume that the class covariance matrices $\boldsymbol{\sigma}_{\mathbf{c}}$ for $\boldsymbol{c = 1, \ldots, K}$ are all the same
- In this case, a single pooled covariance matrix is used and is especially useful when the number of samples in each class is too low to produce a reliable estimate
- The resulting classifier uses hyperplanes as class boundaries, endowing the name **normal-based linear discriminant**

Quadratic and Linear Discriminants

- To further cope with situations where the number of features is comparable to the number of samples, a further simplification of the covariance matrix can be used by setting all off-diagonal elements to be 0
    - This method neglects co-variation between features and was found to outperform other types of classifiers on a variety of microarray analyses

K-Nearest Neighbor Classifier

- The KNN classifier can be seen as a nonparametric method of density estimation and makes no assumptions about the underlying distribution of the data besides the continuity of features
- There is no training involved in the KNN classifier at all. When a new object ***z*** must be classified, the distance between the new object and all other objects in the training set are calculated
- The samples are then ordered closest to furthest from the new object and the ***k*** closest samples are retained

K-Nearest Neighbor Classifier

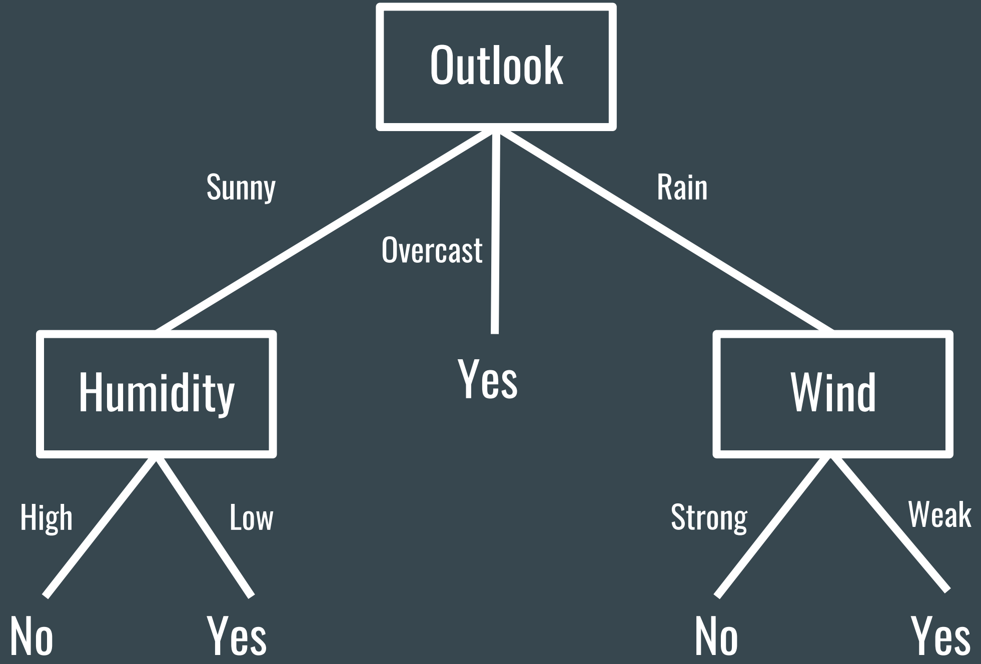
- The number of objects in each distinct class in the $\boldsymbol{k}$ remaining objects is then calculated ($\boldsymbol{n_c}$), and the class that is most represented is chosen to be the class of the new object
- The KNN discriminant function can be written as

$$g_c(x) = n_c$$

- Note for the KNN classifier all of the computation is shifted to the classification phase, and none is needed for the training phase
  - This is very computationally and memory inefficient as the entire training set must be maintained and a large amount of computations must be performed for every classification

Decision Trees

- A **Decision Tree** is constructed by an iterative selection of individual features that are most salient at each node
- At each level, the input space $\boldsymbol{X}$ is repeatedly split into descendant subsets starting with $\boldsymbol{X}$ itself



Decision Trees

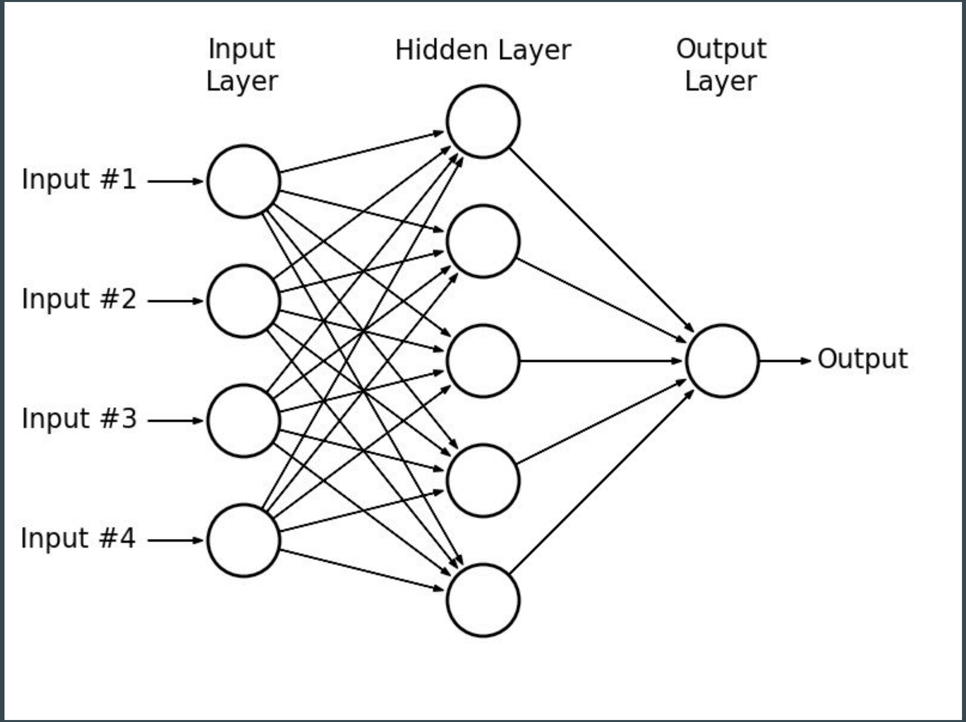
- Normally decision trees are constructed top-down, beginning at the root node and successively partitioning the feature space. This partitioning involves 3 main steps
    - Selecting a splitting rule for each internal node. This involves determining the feature together with the threshold value to split at
    - Determining which nodes are terminal nodes (this means that for each node we must determine to continue to split or to stop and assign a class label)
    - Assign a class label to terminal nodes by minimizing the estimated error rate

Decision Trees

- The most common implementation of decision trees is the binary implementation
  - A single feature is used for each node, resulting in decision boundaries that are parallel to features axes
  - Even though this implementation is suboptimal, they are extremely easy to interpret the set of rules leading to the chosen class label
- The creation of decision trees can be very computationally expensive, as finding the threshold value that creates the best split requires testing many different thresholds for a single feature
  - When performing this for all features in a high dimensional space, the number of computations is very large

Neural Networks

- The most common architecture used in classification problems is a three layered structure of **nodes** in which the signals are propagated from the **input layer** to the **output layer** via the **hidden layer**



[2]

Neural Networks

- The **hidden layer** is called "hidden" because it has no connections outside of the model
- Each hidden unit weights differently all output of the input layer, adds a bias term, and transforms the result using a nonlinear function
  - A common nonlinear function used is the logistic sigmoid function

$$\sigma(z) = \frac{1}{1 + exp(z)}$$

Neural Networks

- Similarly to the hidden layer, the **output layer** processes the output of the hidden layer
- A simple architecture uses one output unit for each class. The discriminant function implemented by the *k*-th output unit of such a network can be written as

$$g_k(x) = \sigma[\sum_j \alpha_{j,k}\sigma(\sum_i x_i w_{i,j} + b_j^h) + b_k^0]$$

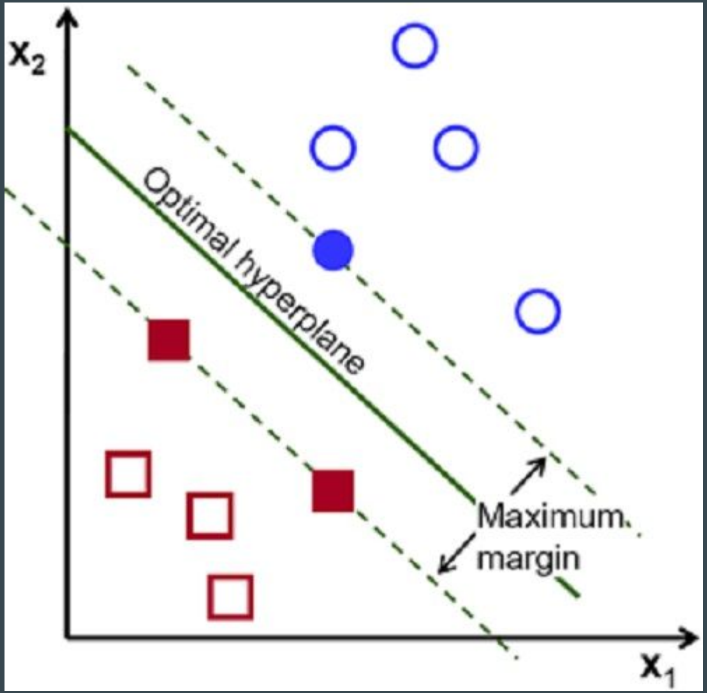
where $\alpha_{j,k}$ is the weight from the j-th hidden unti to the kth output node, b_j^k is the bias term of the h-th hidden unit, b_k^0 is the bias term of the k-th output unit

Neural Networks

- The error of the neural network on the training set can be computed as the sum of the error over all samples
- When a sample belongs to class k, it is desired that the output unit fires 1, while all other units fire 0
- Training is usually performed via **back-propagation** in which the weights are adjusted by the error vector of the network times the partial derivative of the weight with respect to the value of the output nodes

Support Vector Machines

- A classification problem is said to be **linearly separable** if it can be separated by a line (2d) plane (3d) or hyperplane(4d+)
- If a classification boundaries can be made this way, a natural question may be if all of the ways of forming the classification boundary are equally good, or if there is an optimal class boundary
- **Support Vector Machines** find the decision boundary that achieves the maximum margin between classes



[3]

Support Vector Machines

- In the case of an m-class classification problem, the procedure of finding the optimal decision boundary between two classes is repeated several times
- Using the labelled points, the SVM finds a function $f : R^n \rightarrow R^n$ such that for a given input ***x***, $f(x) \geq 0$ if x belongs to the class denotes by 1, otherwise $f(x) < 0$
- The equation $\mathbf{f(x)} = 0$ defines a hyperplane that is used for classification of unknown samples
- When the input consists of linearly separable classes, it is easy to find such a hyperplane

$$f(x) = \langle w \cdot x \rangle + b = \sum_{k=1}^{n} w_k x_k + b$$

Support Vector Machines

$$f(x) = \langle w \cdot x \rangle + b = \sum_{k=1}^{n} w_k x_k + b$$

where ***w*** is the normal vector of the hyperplane defined by ***f(x)=0*** and ***b*** is the offset from the origin. If ***b=0*** , the hyperplane passes through the origin

- Finding such a function is more complex in the case of non linearly separable data and will be covered in future research

Application

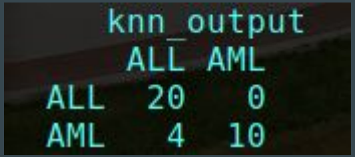
- I have begun to apply these concepts to the data from Golub (1999)
- For preprocessing and feature selection :
  - Use SAM to select around the top 500 differentially expressed genes
  - Perform PCA on these top DE genes
  - The top principal components can be used as features

Application: KNN

k = 3 k = 5 k = 10

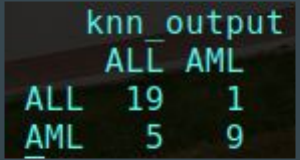
all genes

k = 3



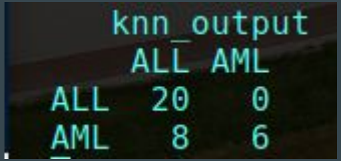
```
          knn_output
           ALL AML
  ALL   20    0
  AML    4   10
```

k = 5



```
          knn_output
           ALL AML
  ALL   19    1
  AML    5    9
```

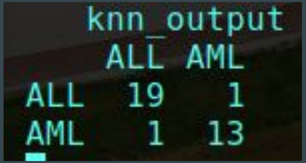
k = 10



```
          knn_output
           ALL AML
  ALL   20    0
  AML    8    6
```

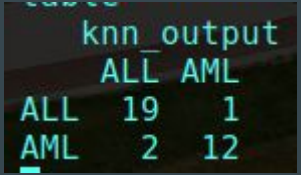
de genes

k = 3



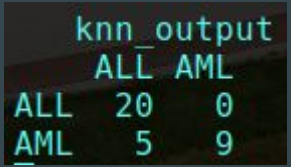
```
          knn_output
           ALL AML
  ALL   19    1
  AML    1   13
```

k = 5



```
          knn_output
           ALL AML
  ALL   19    1
  AML    2   12
```

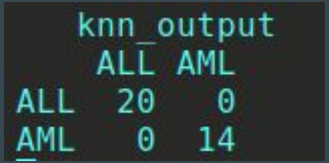
k = 10



```
          knn_output
           ALL AML
  ALL   20    0
  AML    5    9
```

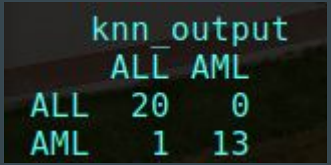
PCs

k = 3



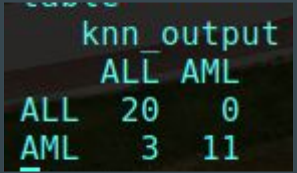
```
          knn_output
           ALL AML
  ALL   20    0
  AML    0   14
```

k = 5



```
          knn_output
           ALL AML
  ALL   20    0
  AML    1   13
```

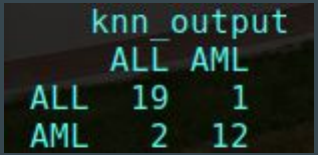
k = 10



```
          knn_output
           ALL AML
  ALL   20    0
  AML    3   11
```

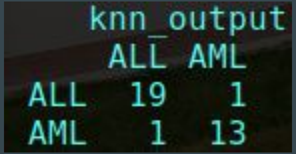
top 5 PCs

k = 3



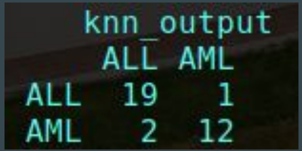
```
          knn_output
           ALL AML
  ALL   19    1
  AML    2   12
```

k = 5



```
          knn_output
           ALL AML
  ALL   19    1
  AML    1   13
```

k = 10



```
          knn_output
           ALL AML
  ALL   19    1
  AML    2   12
```

References

[1] unutbu. “How Can I Make My Confusion Matrix Plot Only 1 Decimal, in Python?” Stack Overflow, 2016, stackoverflow.com/questions/40264763/how-can-i-make-my-confusion-matrix-plot-only-1-decimal-in-python.

[2] Minnaar, Alex. Deep Learning Basics: Neural Networks, Backpropagation and Stochastic Gradient Descent, Machine Learning at University College London, 2015, alexminnaar.com/implementing-the-distbelief-deep-neural-network-training-framework-with-akka.html.

[3] Eliot, Lance. “Support Vector Machines (SVM) for AI Self-Driving Cars.” AI Trends, 19 Jan. 2018, aitrends.com/ai-insider/support-vector-machines-svm-ai-self-driving-cars/.

[4] Drăghici Sorin. Statistics and Data Analysis for Microarrays: Using R and Bioconductor. Chapman and Hall, 2012.