For practical purposes, it is always advisable to apply and compare various models and their performance. There is no general superior model and the result depends on the data and objective. Note also that some results, e.g., from crossvalidation, depend on stochastic mechanisms and are therefore subject to randomness. The following table provides an overview of out-of-sample results of the fitted methods in this chapter and is sorted by descending AUC.

```
result_table = pd.DataFrame([['Logistic Regression', AUC_logreg, sqrt_brier_logreg],
        ['Logistic Regression (L1)', AUC_logreg_reg1, sqrt_brier_logreg_reg1],
        ['Logistic Regression (CV 1)', AUC_logreg_reg2, sqrt_brier_logreg_reg2],
        ['Logistic Regression (CV 2)', AUC_logreg_reg3, sqrt_brier_logreg_reg3],
        ['KNN', AUC_knn, sqrt_brier_knn],
        ['KNN (CV)', AUC_knn_best, sqrt_brier_knn_best],
        ['Naive Bayes', AUC_nb, sqrt_brier_nb],
        ['Decision Tree', AUC_dt, sqrt_brier_dt],
        ['Decision Tree (CV)', AUC_dt_best, sqrt_brier_dt_best],
        ['SVM', AUC_svm, sqrt_brier_svm]
                    ]).rename(columns={0:'Method', 1:'AUC', 2:'RMSE SQR(Brier score)'})
pd.set_option('display.max_rows', None)

print(result_table.sort_values(by='AUC', axis=0, ascending=False))
```

```
                        Method     AUC  RMSE SQR(Brier score)
1    Logistic Regression (L1)  0.6737                 0.1871
0         Logistic Regression  0.6732                 0.1871
2  Logistic Regression (CV 1)  0.6712                 0.1870
5                   KNN (CV)  0.6657                 0.1867
8          Decision Tree (CV)  0.6614                 0.2543
6                Naive Bayes  0.6439                 0.5654
7               Decision Tree  0.6426                 0.2986
4                         KNN  0.6364                 0.1855
3  Logistic Regression (CV 2)  0.6321                 0.1866
9                         SVM  0.4638                 0.1887
```

13.9 Sandbox Problems

- Estimate a Logistic Regression model with `l1` regularization. Choose an appropriate feature space.
- Build a decision tree model for the three features FICO, current LTV and interest rate at loan origination.
- Run a randomized grid search for decision trees where you randomize the maximum number of features and the maximum depth.
- Create a visualization of the best tree.
- Run a systematic grid search over the penalty parameter for a support vector machine.

14 Neural Networks and Deep Learning

We now import standard packages and functions calling the `dcr` module using `from dcr import *` and ignore warning messages.

```
import warnings; warnings.simplefilter('ignore')
from dcr import *
```

We run `%matplotlib inline` to see the full outputs, and specify the resolution of the figures.

```
%matplotlib inline
plt.rcParams['figure.dpi'] = 300
plt.rcParams['figure.figsize'] = (16, 9)
plt.rcParams.update({'font.size': 16})
```

14.1 Synopsis

In this chapter we introduce Artificial Neural Networks (ANN, or simply Neural Networks). They are often compared to the human brain, with multi-dimensional layers of neurons. Neural networks are extensions of regression models. Our networks are from the network class called Multi-Layer-Perceptron (MLP). At the end of the section, we will discuss an outlook into current developments with more complex structures. For the MLP class we use the `sklearn.neural_network` module and class `MLPClassifier` for PD modeling.

14.2 Neural Networks and Deep Learning

14.2.1 Idea

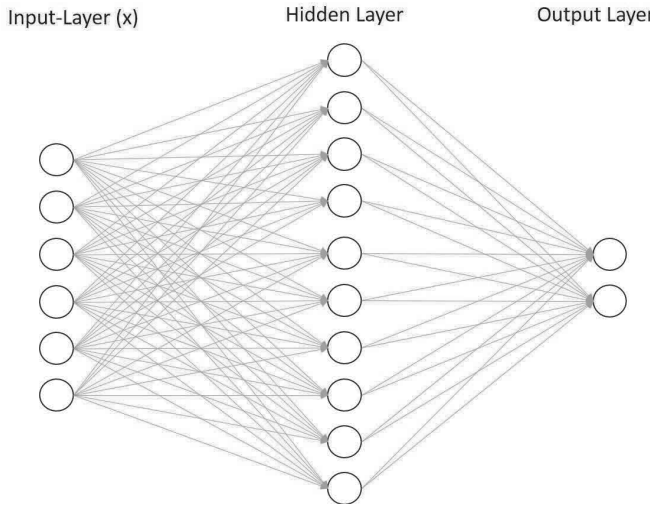
Neural networks are non-linear statistical models for regression or classification. Special simple forms are feed-forward Multi-Layer Perceptrons with backpropagation.

A schematic example of a **Single-Layer Perceptron** is shown in the figure below. It consists of the usual inputs (x), called input layer, and the outputs (y) as in a classical regression model. In addition, it has a so-called hidden layer. The values in the hidden layer are also called derived features and are computed as (non-) linear functions of the inputs or features (x). The fitted values for targets/outputs (y) are computed as (non-) linear functions of the derived features. These (non-) linear functions are called **Activation Functions**.

For categorical targets or outcomes (e.g., default vs. non-default) typically a sigmoid (logistic) function is used, which we know from Logistic Regressions:

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

A network structure or architecture with more than one hidden layer is often called **Deep Learning**.

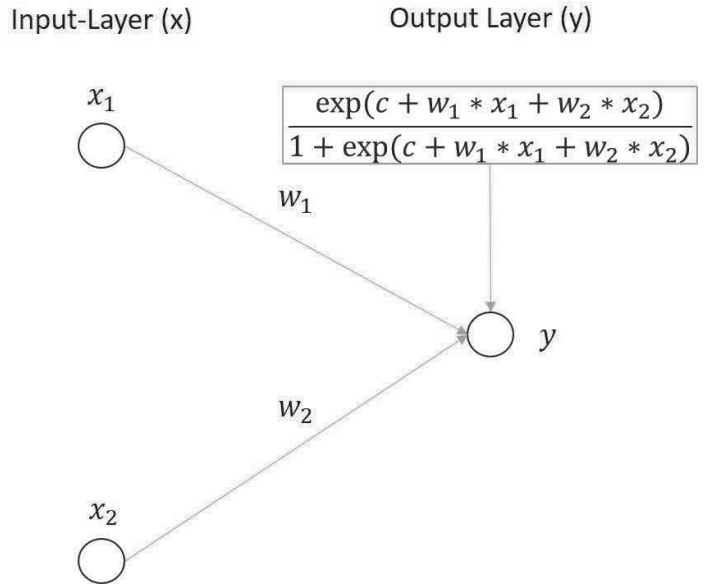


A neural network is fitted using the concept of **Backpropagation** by the following steps. We define a loss function. We typically use crossentropy for default risk. Subsequently, we apply regularization and early stopping criterion of the fitting to reduce overfitting. Thereafter, we apply the gradient descent approach from output back to the inputs (called backpropagation) in the following way: In a forward step, we compute predicted values using fixed weights (coefficients). In a backward step, we compute the 'errors' for each layer. After that, we adjust the weights and repeat. A training epoch is a sweep over the training set. The learning rate can be constant or variable (decreasing). The regularization parameter is typically estimated using cross-validation.

14.2.2 Simple Network without Hidden Layer

Let us consider an example that shows the analogy of regressions and neural networks. We look at a network with an input layer consisting of two features x and an output layer consisting of defaults y.

We first create two standard normally distributed and independent random variables x_1 and x_2 with $n = 2000$ observations as training data (and also $n = 2000$ as test data). This is for convenience, so we no longer have to standardize the inputs. Next, we specify some weights w_1 and w_2 and a bias. We compute a linear predictor and train and test PDs after defining the logistic function (`inv_logit`). We randomly create Bernoulli defaults from these. The x-variables are our inputs and the defaults (`y_train` and `y_test`, respectively) are the outputs. The figure shows the simple network architecture.

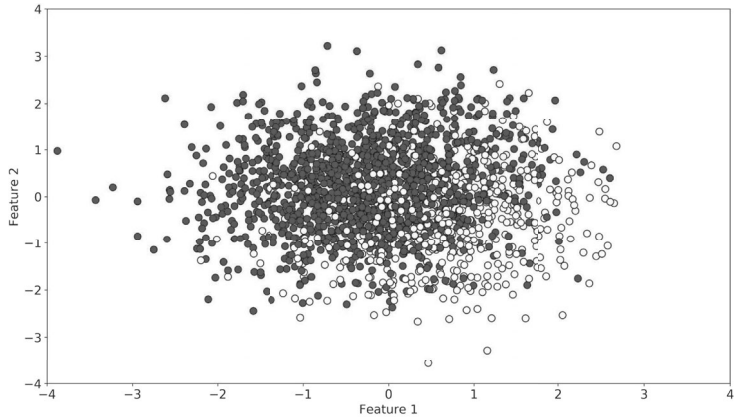



```
np.random.seed(1234)
n = 2000
sigma = 1
mean_X = [0, 0]
cov_X = [[1, 0], [0, 1]]

X_train = np.random.multivariate_normal(mean_X, cov_X, n)
X_test = np.random.multivariate_normal(mean_X, cov_X, n)

bias = -1
```

```
w1 = 1
w2 = -0.8

def inv_logit(eta):
    return np.exp(eta) / (1 + np.exp(eta))

lin_pred_train = bias + w1 * X_train[:,0] +w2 * X_train[:,1]
lin_pred_test = bias + w1 * X_test[:,0] +w2 * X_test[:,1]

PD_train = inv_logit(lin_pred_train)
PD_test = inv_logit(lin_pred_test)

y_train = np.random.binomial(1, PD_train)
y_test = np.random.binomial(1, PD_test)
```

We then plot both training features and mark the output category (default/non-default) separately. We subsample them to be able to plot classes with different colors. The defaults are the unfilled circles with the tendency to lie in the lower right.

```
from matplotlib.colors import ListedColormap
cmap_bold = ListedColormap(['#0000FF', '#00FF00', '#FFFFFF'])
plt.figure()
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cmap_bold,
edgecolor='k', s=80)
plt.xlim(-4, 4)
plt.ylim(-4, 4)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

Subsequently, we fit a Logistic Regression. In our case, this is actually the right specification for the model, because the data were generated this way. We check the performance for the test data. We see that the fitted parameters are close to the parameters of the data generating process.

```
model_lr = LogisticRegression(penalty='none', fit_intercept=True, solver='saga', n_jobs=2,
     tol =1e-15, max_iter=2000)
model_lr.fit(X_train, y_train)

print('Coefficients:', model_lr.coef_.round(decimals=4))
print('Intercept:', model_lr.intercept_.round(decimals=4))

predictions = model_lr.predict_proba(X_test)[:,1].T
predictions_cat = model_lr.predict(X_test)

print(classification_report(y_test,predictions_cat))
print(confusion_matrix(y_test,predictions_cat))
print("Log Loss:", log_loss(y_test, predictions))
print("AUC:", roc_auc_score(y_test, predictions))
print("Brier score: (the smaller the better):", brier_score_loss(y_test, predictions))
```

```
Coefficients: [[ 1.0895 -0.7983]]
Intercept: [-0.9784]
              precision    recall  f1-score   support

           0       0.78      0.89      0.83      1341
           1       0.68      0.48      0.56       659

    accuracy                           0.75      2000
```