Learning (supervised) algorithm and training data features and corresponding labels. This model will take features from new data samples and output predicted labels in the prediction phase.

Unsupervised Machine Learning Pipeline

Unsupervised Machine Learning is all about extracting patterns, relationships, associations, and clusters from data. The processes related to feature engineering, scaling and selection are similar to supervised learning. However there is no concept of pre-labeled data here. Hence the unsupervised Machine Learning pipeline would be slightly different in contrast to the supervised pipeline. Figure 1-25 depicts a standard unsupervised Machine Learning pipeline.

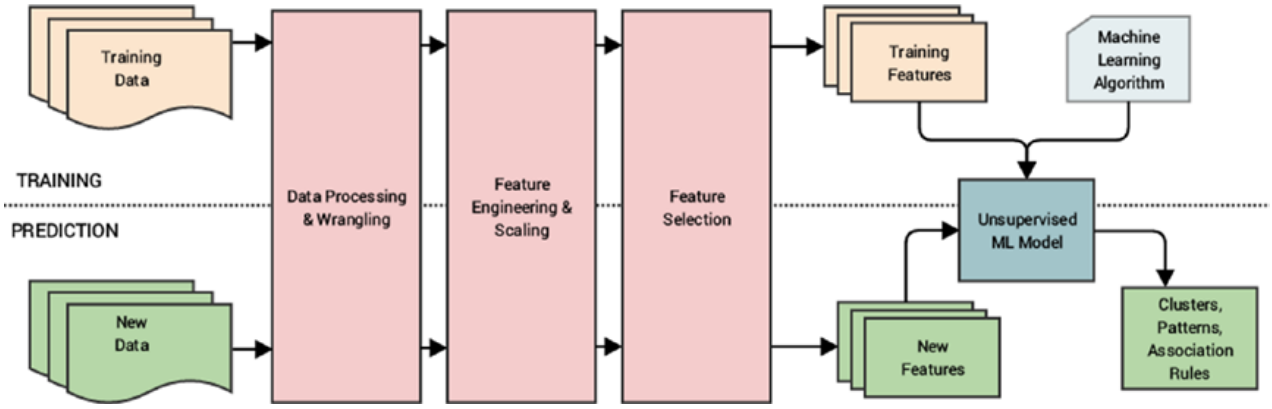

Figure 1-25. *Unsupervised Machine Learning pipeline*

Figure 1-25 clearly depicts that no supervised labeled data is used for training the model. With the absence of labels, we just have training data that goes through the same data preparation phase as in the supervised learning pipeline and we build our unsupervised model with an unsupervised Machine Learning algorithm and training features. In the prediction phase, we extract features from new data samples and pass them through the model which gives relevant results according to the type of Machine Learning task we are trying to perform, which can be clustering, pattern detection, association rules, or dimensionality reduction.

Real-World Case Study: Predicting Student Grant Recommendations

Let's take a step back from what we have learned so far! The main objective here was to gain a solid grasp over the entire Machine Learning landscape, understand crucial concepts, build on the basic foundations, and understand how to execute Machine Learning projects with the help of Machine Learning pipelines with the CRISP-DM process model being the source of all inspiration. Let's put all this together to take a very basic real-world case study by building a supervised Machine Learning pipeline on a toy dataset. Our major objective is as follows. Given that you have several students with multiple attributes like grades, performance, and scores, can you build a model based on past historical data to predict the chance of the student getting a recommendation grant for a research project?

This will be a quick walkthrough with the main intent of depicting how to build and deploy a real-world Machine Learning pipeline and perform predictions. This will also give you a good hands-on experience to get started with Machine Learning. Do not worry too much if you don't understand the details of each and every line of code; the subsequent chapters cover all the tools, techniques, and frameworks used here in

detail. We will be using Python 3.5 in this book; you can refer to Chapter 2, "The Python Machine Learning Ecosystem" to understand more about Python and the various tools and frameworks used in Machine Learning. You can follow along with the code snippets in this section or open the `Predicting Student Recommendation Machine Learning Pipeline.ipynb` jupyter notebook by running **`jupyter notebook`** in the command line/terminal in the same directory as this notebook. You can then run the relevant code snippets in the notebook from your browser. Chapter 2 covers jupyter notebooks in detail.

Objective

You have historical student performance data and their grant recommendation outcomes in the form of a comma separated value file named `student_records.csv`. Each data sample consists of the following attributes.

- `Name` (the student name)
- `OverallGrade` (overall grade obtained)
- `Obedient` (whether they were diligent during their course of stay)
- `ResearchScore` (marks obtained in their research work)
- `ProjectScore` (marks obtained in the project)
- `Recommend` (whether they got the grant recommendation)

You main objective is to build a predictive model based on this data such that you can predict for any future student whether they will be recommended for the grant based on their performance attributes.

Data Retrieval

Here, we will leverage the `pandas` framework to retrieve the data from the CSV file. The following snippet shows us how to retrieve the data and view it.

```
In [1]: import pandas as pd
   ...: # turn of warning messages
   ...: pd.options.mode.chained_assignment = None  # default='warn'
   ...:
   ...: # get data
   ...: df = pd.read_csv('student_records.csv')
   ...: df
```

Out[1]:

	Name	OverallGrade	Obedient	ResearchScore	ProjectScore	Recommend
0	Henry	A	Y	90	85	Yes
1	John	C	N	85	51	Yes
2	David	F	N	10	17	No
3	Holmes	B	Y	75	71	No
4	Marvin	E	N	20	30	No
5	Simon	A	Y	92	79	Yes
6	Robert	B	Y	60	59	No
7	Trent	C	Y	75	33	No

Figure 1-26. *Raw data depicting student records and their recommendations*

Now that we can see data samples showing records for each student and their corresponding recommendation outcomes in Figure 1-26, we will perform necessary tasks relevant to data preparation.

Data Preparation

Based on the dataset we saw earlier, we do not have any data errors or missing values, hence we will mainly focus on feature engineering and scaling in this section.

Feature Extraction and Engineering

Let's start by extracting the existing features from the dataset and the outcomes in separate variables. The following snippet shows this process. See Figures 1-27 and 1-28.

```
In [2]: # get features and corresponding outcomes
   ...: feature_names = ['OverallGrade', 'Obedient', 'ResearchScore',
                         'ProjectScore']
   ...: training_features = df[feature_names]
   ...:
   ...: outcome_name = ['Recommend']
   ...: outcome_labels = df[outcome_name]

In [3]: # view features
   ...: training_features
```

Out[3]:

	OverallGrade	Obedient	ResearchScore	ProjectScore
0	A	Y	90	85
1	C	N	85	51
2	F	N	10	17
3	B	Y	75	71
4	E	N	20	30
5	A	Y	92	79
6	B	Y	60	59
7	C	Y	75	33

Figure 1-27. *Dataset features*

```
In [4]: # view outcome labels
   ...: outcome_labels
```

Out[4]:

	Recommend
0	Yes
1	Yes
2	No
3	No
4	No
5	Yes
6	No
7	No

Figure 1-28. *Dataset recommendation outcome labels for each student*

Now that we have extracted our initial available features from the data and their corresponding outcome labels, let's separate out our available features based on their type (numerical and categorical). Types of feature variables are covered in more detail in Chapter 3, "Processing, Wrangling, and Visualizing Data".

```
In [5]: # list down features based on type
   ...: numeric_feature_names = ['ResearchScore', 'ProjectScore']
   ...: categoricial_feature_names = ['OverallGrade', 'Obedient']
```

We will now use a standard scalar from `scikit-learn` to scale or normalize our two numeric score-based attributes using the following code.

```
In [6]: from sklearn.preprocessing import StandardScaler
   ...: ss = StandardScaler()
   ...:
   ...: # fit scaler on numeric features
   ...: ss.fit(training_features[numeric_feature_names])
   ...:
   ...: # scale numeric features now
   ...: training_features[numeric_feature_names] =
                                    ss.transform(training_features[numeric_feature_names])
   ...:
   ...: # view updated featureset
   ...: training_features
```

Out[6]:

	OverallGrade	Obedient	ResearchScore	ProjectScore
0	A	Y	0.899583	1.376650
1	C	N	0.730648	-0.091777
2	F	N	-1.803390	-1.560203
3	B	Y	0.392776	0.772004
4	E	N	-1.465519	-0.998746
5	A	Y	0.967158	1.117516
6	B	Y	-0.114032	0.253735
7	C	Y	0.392776	-0.869179

Figure 1-29. *Feature set with scaled numeric attributes*

Now that we have successfully scaled our numeric features (see Figure 1-29), let's handle our categorical features and carry out the necessary feature engineering needed based on the following code.

```
In [7]: training_features = pd.get_dummies(training_features,
                                           columns=categoricial_feature_names)
   ...: # view newly engineering features
   ...: training_features
```

Out[7]:

ResearchScore	ProjectScore	OverallGrade_A	OverallGrade_B	OverallGrade_C	OverallGrade_E	OverallGrade_F	Obedient_N	Obedient_Y
0.899583	1.376650	1	0	0	0	0	0	1
0.730648	-0.091777	0	0	1	0	0	1	0
-1.803390	-1.560203	0	0	0	0	1	1	0
0.392776	0.772004	0	1	0	0	0	0	1
-1.465519	-0.998746	0	0	0	1	0	1	0
0.967158	1.117516	1	0	0	0	0	0	1
-0.114032	0.253735	0	1	0	0	0	0	1
0.392776	-0.869179	0	0	1	0	0	0	1

Figure 1-30. *Feature set with engineered categorical variables*

```
In [8]: # get list of new categorical features
   ...: categorical_engineered_features = list(set(training_features.columns) -
                                               set(numeric_feature_names))
```

Figure 1-30 shows us the updated feature set with the newly engineered categorical variables. This process is also known as one hot encoding.

Modeling

We will now build a simple classification (supervised) model based on our feature set by using the logistic regression algorithm. The following code depicts how to build the supervised model.

```
In [9]: from sklearn.linear_model import LogisticRegression
   ...: import numpy as np
   ...:
   ...: # fit the model
   ...: lr = LogisticRegression()
   ...: model = lr.fit(training_features,
                       np.array(outcome_labels['Recommend']))
   ...: # view model parameters
   ...: model
Out[9]: LogisticRegression(C=1.0, class_weight=None, dual=False,
                           fit_intercept=True, intercept_scaling=1, max_iter=100,
                           multi_class='ovr', n_jobs=1, penalty='l2',
                           random_state=None, solver='liblinear', tol=0.0001,
                           verbose=0, warm_start=False)
```

Thus, we now have our supervised learning model based on the logistic regression model with L2 regularization, as you can see from the parameters in the previous output.

Model Evaluation

Typically model evaluation is done based on some holdout or validation dataset that is different from the training dataset to prevent overfitting or biasing the model. Since this is an example on a toy dataset, let's evaluate the performance of our model on the training data using the following snippet.

```
In [10]: # simple evaluation on training data
    ...: pred_labels = model.predict(training_features)
    ...: actual_labels = np.array(outcome_labels['Recommend'])
    ...:
    ...: # evaluate model performance
    ...: from sklearn.metrics import accuracy_score
    ...: from sklearn.metrics import classification_report
    ...:
    ...: print('Accuracy:', float(accuracy_score(actual_labels,
                pred_labels))*100, '%')
    ...: print('Classification Stats:')
    ...: print(classification_report(actual_labels, pred_labels))

Accuracy: 100.0 %
Classification Stats:
             precision    recall  f1-score   support

         No       1.00      1.00      1.00         5
        Yes       1.00      1.00      1.00         3

avg / total       1.00      1.00      1.00         8
```

Thus you can see the various metrics that we had mentioned earlier, like accuracy, precision, recall, and F1 score depicting the model performance. We talk about these metrics in detail in Chapter 5, "Building, Tuning, and Deploying Models".

Model Deployment

We built our first supervised learning model, and to deploy this model typically in a system or server, we need to persist the model. We also need to save the scalar object we used to scale the numerical features since we use it to transform the numeric features of new data samples. The following snippet depicts a way to store the model and scalar objects.

```
In [11]: from sklearn.externals import joblib
    ...: import os
    ...: # save models to be deployed on your server
    ...: if not os.path.exists('Model'):
    ...:     os.mkdir('Model')
    ...: if not os.path.exists('Scaler'):
    ...:     os.mkdir('Scaler')
    ...:
    ...: joblib.dump(model, r'Model/model.pickle')
    ...: joblib.dump(ss, r'Scaler/scaler.pickle')
```

These files can be easily deployed on a server with necessary code to reload the model and predict new data samples, which we will see in the upcoming sections.

Prediction in Action

We are now ready to start predicting with our newly built and deployed model! To start predictions, we need to load our model and scalar objects into memory. The following code helps us do this.

```
In [12]: # load model and scaler objects
    ...: model = joblib.load(r'Model/model.pickle')
    ...: scaler = joblib.load(r'Scaler/scaler.pickle')
```

We have some sample new student records (for two students) for which we want our model to predict if they will get the grant recommendation. Let's retrieve and view this data using the following code.

```
In [13]: ## data retrieval
    ...: new_data = pd.DataFrame([{'Name': 'Nathan', 'OverallGrade': 'F',
                'Obedient': 'N', 'ResearchScore': 30, 'ProjectScore': 20},
    ...:                          {'Name': 'Thomas', 'OverallGrade': 'A',
                'Obedient': 'Y', 'ResearchScore': 78, 'ProjectScore': 80}])
    ...: new_data = new_data[['Name', 'OverallGrade', 'Obedient',
                              'ResearchScore', 'ProjectScore']]
    ...: new_data
```

Out[13]:

	Name	OverallGrade	Obedient	ResearchScore	ProjectScore
0	Nathan	F	N	30	20
1	Thomas	A	Y	78	80

Figure 1-31. *New student records*

We will now carry out the tasks relevant to data preparation—feature extraction, engineering, and scaling—in the following code snippet.

```
In [14]: ## data preparation
    ...: prediction_features = new_data[feature_names]
    ...:
    ...: # scaling
    ...: prediction_features[numeric_feature_names] =
                scaler.transform(prediction_features[numeric_feature_names])
    ...:
    ...: # engineering categorical variables
    ...: prediction_features = pd.get_dummies(prediction_features,
                                    columns=categoricial_feature_names)
    ...:
    ...: # view feature set
    ...: prediction_features
```

Out[14]:

	ResearchScore	ProjectScore	OverallGrade_A	OverallGrade_F	Obedient_N	Obedient_Y
0	-1.127647	-1.430636	0	1	1	0
1	0.494137	1.160705	1	0	0	1

Figure 1-32. *Updated feature set for new students*

We now have the relevant features for the new students! However you can see that some of the categorical features are missing based on some grades like B, C, and E. This is because none of these students obtained those grades but we still need those attributes because the model was trained on all attributes including these. The following snippet helps us identify and add the missing categorical features. We add the value for each of those features as 0 for each student since they did not obtain those grades.

```
In [15]: # add missing categorical feature columns
    ...: current_categorical_engineered_features =
             set(prediction_features.columns) - set(numeric_feature_names)
    ...: missing_features = set(categorical_engineered_features) -
                            current_categorical_engineered_features
    ...: for feature in missing_features:
    ...:     # add zeros since feature is absent in these data samples
    ...:     prediction_features[feature] = [0] * len(prediction_features)
    ...:
    ...: # view final feature set
    ...: prediction_features
```

Out[15]:

	ResearchScore	ProjectScore	OverallGrade_A	OverallGrade_F	Obedient_N	Obedient_Y	OverallGrade_C	OverallGrade_B	OverallGrade_E
0	-1.127647	-1.430636	0	1	1	0	0	0	0
1	0.494137	1.160705	1	0	0	1	0	0	0

Figure 1-33. *Final feature set for new students*

We have our complete feature set ready for both the new students. Let's put our model to the test and get the predictions with regard to grant recommendations!

```
In [16]: ## predict using model
    ...: predictions = model.predict(prediction_features)
    ...:
    ...: ## display results
    ...: new_data['Recommend'] = predictions
    ...: new_data
```

Out[16]:

	Name	OverallGrade	Obedient	ResearchScore	ProjectScore	Recommend
0	Nathan	F	N	30	20	No
1	Thomas	A	Y	78	80	Yes

Figure 1-34. *New student records with model predictions for grant recommendations*

We can clearly see from Figure 1-34 that our model has predicted grant recommendation labels for both the new students. Thomas clearly being diligent, having a straight A average and decent scores, is most likely to get the grant recommendation as compared to Nathan. Thus you can see that our model has learned how to predict grant recommendation outcomes based on past historical student data. This should whet your appetite on getting started with Machine Learning. We are about to deep dive into more complex real-world problems in the upcoming chapters!

Challenges in Machine Learning

Machine Learning is a rapidly evolving, fast-paced, and exciting field with a lot of prospect, opportunity, and scope. However it comes with its own set of challenges, due to the complex nature of Machine Learning methods, its dependency on data, and not being one of the more traditional computing paradigms. The following points cover some of the main challenges in Machine Learning.

- Data quality issues lead to problems, especially with regard to data processing and feature extraction.
- Data acquisition, extraction, and retrieval is an extremely tedious and time consuming process.
- Lack of good quality and sufficient training data in many scenarios.
- Formulating business problems clearly with well-defined goals and objectives.
- Feature extraction and engineering, especially hand-crafting features, is one of the most difficult yet important tasks in Machine Learning. Deep Learning seems to have gained some advantage in this area recently.
- Overfitting or underfitting models can lead to the model learning poor representations and relationships from the training data leading to detrimental performance.
- The curse of dimensionality: too many features can be a real hindrance.
- Complex models can be difficult to deploy in the real world.

This is not an exhaustive list of challenges faced in Machine Learning today, but it is definitely a list of the top problems data scientists or analysts usually face in Machine Learning projects and tasks. We will cover dealing with these issues in detail when we discuss more about the various stages in the Machine Learning pipeline as well as solve real-world problems in subsequent chapters.

Real-World Applications of Machine Learning

Machine Learning is widely being applied and used in the real world today to solve complex problems that would otherwise have been impossible to solve based on traditional approaches and rule-based systems. The following list depicts some of the real-world applications of Machine Learning.