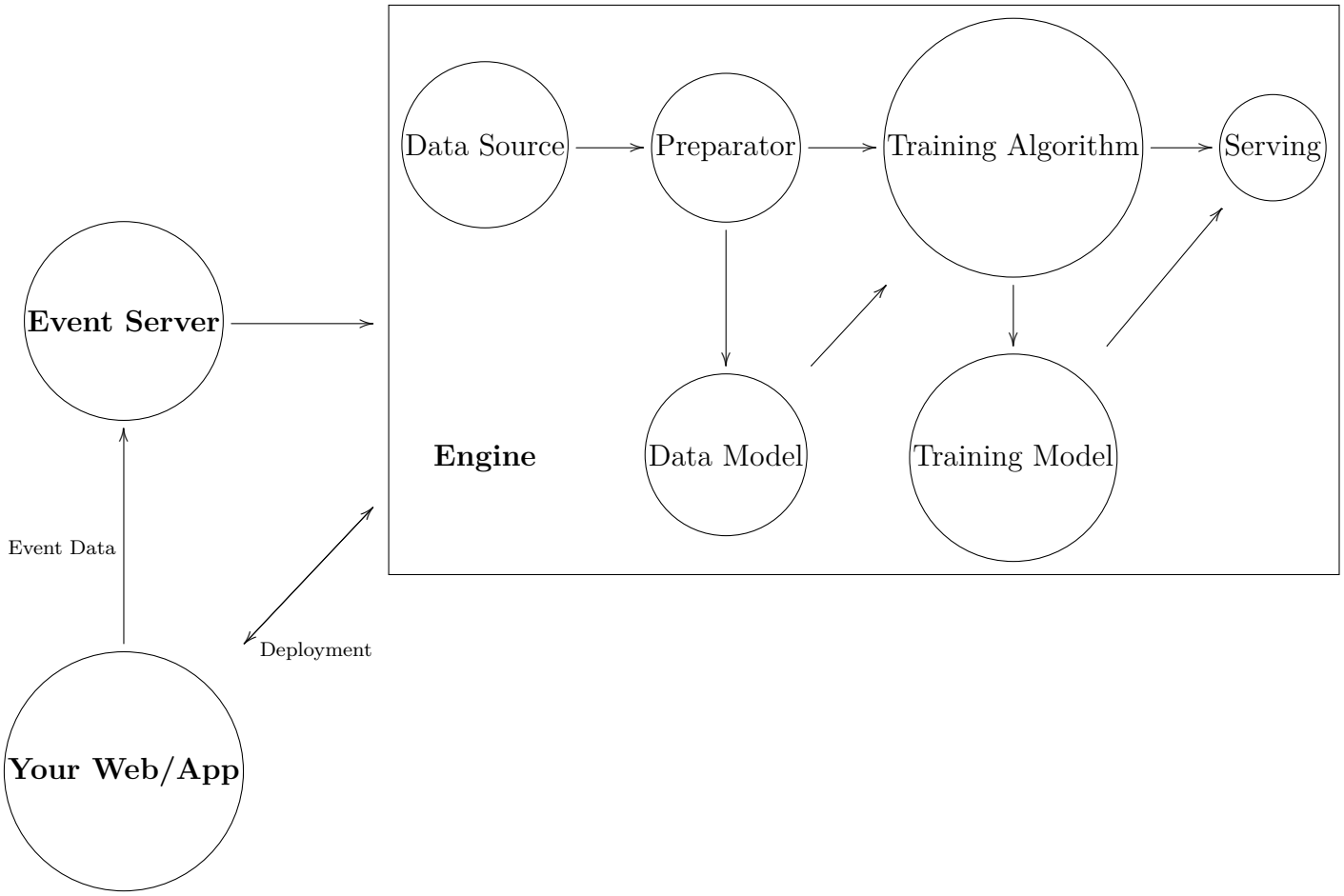
PredictionIO: Modeling Text Data Engine

The purpose of this tutorial is to illustrate how to model text data using PredictionIO's engine platform. We will be largely mirroring the tutorial provided for Scikit's Learn package title "Working With Text Data." However, we will be providing instructions for building the model using PredictionIO's engine platform. The advantages of using this framework include the usage of Apache Spark for distributed computation which is crucial for computation with large data sets, as well as the capacity to use a newly trained predictive model to respond to queries in real-time.
We will assume the user is running the PredictionIO version 0.9.2, and meets all minimum computing requirements. To download PredictionIO, follow the instructions on the Getting Started tutorial.

Engine Overview

As a user, we are charged with collecting data and importing it into an event server. The data is then read and processed by our engine via the `DataSource` and `Preparation` components. The `Algorithm` engine component then trains a predictive model using the processed, or prepared, data. Once we have trained a model, we are ready to deploy our engine and respond to real-time queries via the `Serving` component.
We are given our observations as a set of strings $\mathcal{S}$. In a general setting, there are two different kind of problems we can be given. In the first, each string s contained in $\mathcal{S}$ is also associated to a class label y taking values in $\{1, 2, ..., K\}$, where K is an integer, which we can think of as a categorical, or class, label.We may proceed by learning a predictive model f, a function which takes in a string s and outputs a class label, using the observed data pairs (y, s). Once we have learned f then given a new string s_{new}, we can predict a class label $f(s_{\text{new}})$. This is an example of a supervised learning problem. To exemplify the second kind of problem, suppose we are only given the set of strings $\mathcal{S}$, and are asked to group the string together into K different groups based on a set of characteristics which we extract from the strings. This is an example of an unsupervised learning problem.
In both cases, we are tasked with learning a model f. However, before this we must first represent our strings s in such a manner so that we may apply existing machine learning methods provided in the Spark MLLib library. We will be using the Apache OpenNLP tools library for the purpose of data representation.
Our engine includes all DASE components, as well as a Data Model implementation which deals with representing our text data as numeric feature vectors. It also incorporates a Training Model component which is more of a conceptual framework representing a set of Scala classes that can produce a supervised (or unsupervised) learning model f. For our supervised training model implementation, f will be a classifier, and for our unsupervised training model, an assignment of class labels for each string in $\mathcal{S}$. The following diagram shows the how the provided Modeling Text Data Engine template functionality is structured, and the interactions between your web/app, the provided Event Server, and our Modeling Text Data Engine.



We begin the data collection stage by importing a Scikit Learn dataset, `20newsgroups`, into PredictionIO's event server, as well as a set of English stop words that will be used in the Data Model implementation. We will delve more into the latter event data when we reach the Data Model section.

Importing Data

We will be using Scikit Learn's datasets and text modules, as well as the PredictionIO Python SDK. First, let's get a better feel for the data: initialize your Python interpreter and type the following lines:

```
>>> from sklearn.datasets import fetch_20newsgroups
>>> twenty_train = fetch_20newsgroups(subset = 'train',
...                                   shuffle = True,
...                                   random_state = 10)
```

This initializes a dictionary-like object as can be seen in the following output:

```
>>> twenty_train.keys()
dict_keys(['filenames', 'DESCR', 'target', 'data', 'target_names'])
```

What we are interested in here are the target and data keys. Type in the following line to see how we will be preparing the data prior to importing it to the PredictionIO Event Server.

```
>>> [(twenty_train.target[k], twenty_train.data[k])
...  for k in range(len(twenty_train.data))][0]
```

Our main tool from the PredictionIO Python SDK that we will use to import our data into the event server is the class `EventClient` and its method `create_event`. The function that implements this in the script MyTextEngine/data/import_eventserver.py with respect to the `20newsgroups` dataset is `import_events`. As fore mentioned, we will also be importing Scikit's library of English stop words into PredictionIO's Event Server. This is locally stored as a frozen set of strings, as can be verified by typing the following into your Python interpreter:

```
>>> from sklearn.feature_extraction import text
>>> text.ENGLISH_STOP_WORDS
```

Again, the class `EventClient` is the SDK component that allows us import the stop word data into the PredictionIO event server. The function used to import the stop word events is `import_stopwords` which is defined in the same script. With these two functions under our belt, we are ready to begin importing data. First, make sure PredictionIO is running, you can check this by typing `pio status` on your terminal. If it is not running, start it with the command `pio-start-all`. Now, let's create a new application named MyTextApp by entering the command `pio app new MyTextApp` in your shell. You should see some printed output which includes your newly created access key. At this point make sure you are in the template root directory, and import the data set using the following command:

```
python data/import_eventserver.py --access_key *****
```

where you will replace `*****` with your actual access key. If the data is successfully imported, you should see the following subsequent printed output

```
Importing data.....
Imported 11314 events.
Importing stop words.....
Imported 318 stop words.
```

Our data is now sitting in PredictionIO's Event Server and ready to be used by our engine. The following section will get you acquainted with the different engine components that are implemented in this engine template.

Engine Components

Data Source

Preparator

Data Model

Our data model implementation is actually just a Scala class taking in as parameters `td`, `nMin`, `nMax`, where `td` is an object of class `TrainingData`, and the other two parameters are the components of our n-gram window which we will define shortly. In this section, we give an overview of how we go about representing our document strings. It will be easier to explain this process with an example, so consider the document:

$$D := \texttt{"Hello, my name is Marco."}$$

The first thing we need to do is break up D into a list of "allowed tokens." You can think of a token as a terminating sequence of characters that exist in our document (think of a word in a sentence). For example, the list of tokens that appear in D is:

$$\texttt{Hello} \rightarrow \texttt{,} \rightarrow \texttt{my} \rightarrow \texttt{name} \rightarrow \texttt{is} \rightarrow \texttt{Marco} \rightarrow \texttt{.}$$

Now, recall that when we imported our data, we also imported a set of stop words. This set of stop words contains all the words (or tokens) that we do not want to include once we tokenize our documents. Hence, we will call the tokens that appear in D and are not contained in our set of stop words allowed tokens. So, if our set of stop words is $\{\texttt{my}, \texttt{is}\}$, then the list of allowed tokens appearing in D is:

$$\texttt{Hello} \rightarrow \texttt{,} \rightarrow \texttt{name} \rightarrow \texttt{Marco} \rightarrow \texttt{.}$$