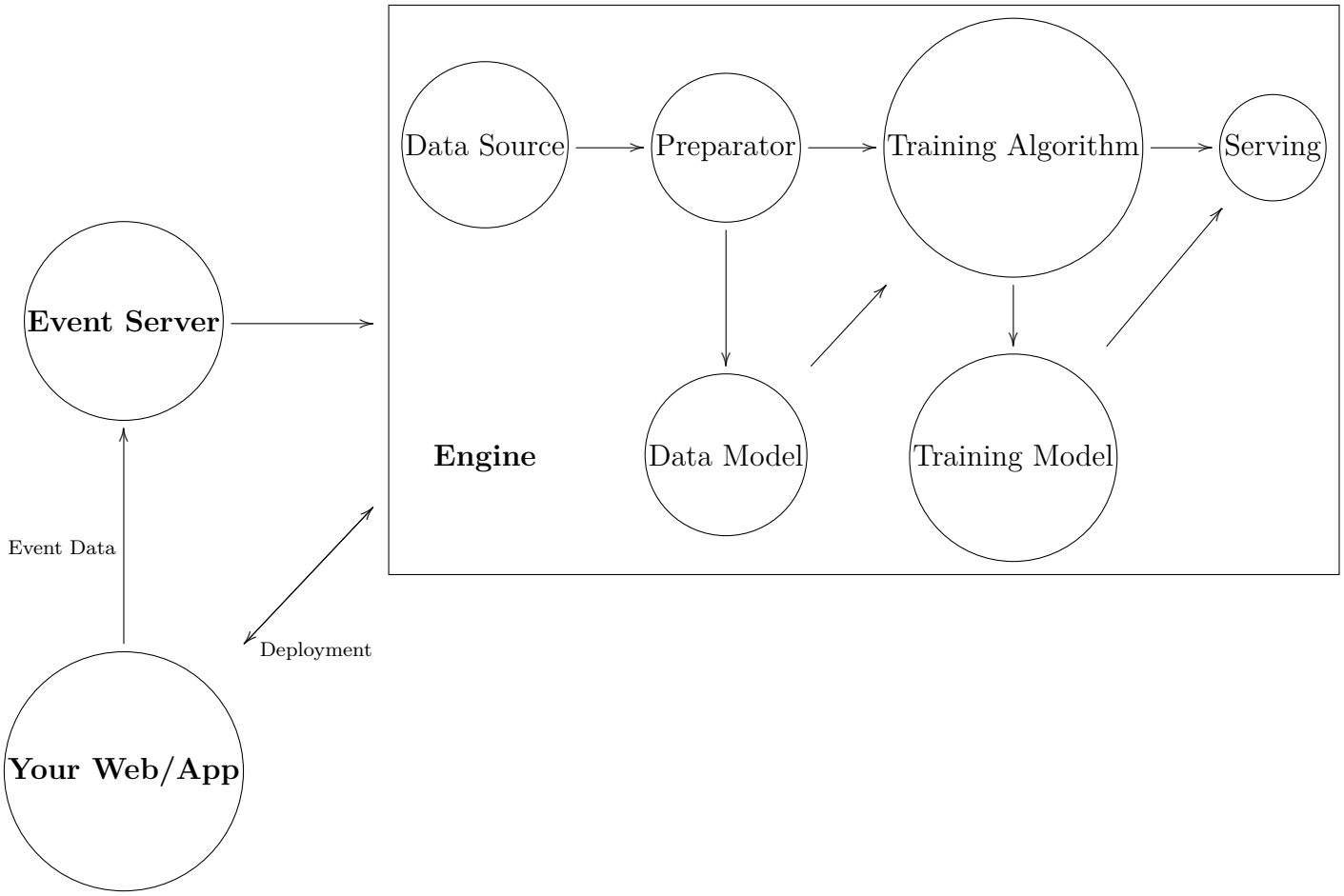
PredictionIO Engine Template: Modeling Text Data

Introduction

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 module. 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 an data entries. Type in the following line to see how we prepare the Scikit 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 Python SDK that we will use to import our data into the event server is the class `EventClient` and its method `create_event`. Now, in your data directory create a file named `import_eventserver.py`. Initialize the script by adding the following lines:

```
import predictionio
import argparse
```

We will now define a function `import_events` which will take as input an object of type `EventClient` and a string object providing the name of the data document, which is in our case `train.tsv`. We will skip the details of the document processing in Python, and refer to the reader to the following documentation:

`https://docs.python.org/2/library/strings.html`

We will process each line in the file and convert it to a list of strings. Once each line is processed, an event is created using the aforementioned client method. We will be assigning various properties to our data so that our engine can successfully read the corresponding event data parts that will be needed to train our predictive model. Your function should look similar to the following:

```
def import_events(client):
    train = ((float(twenty_train.target[k]), twenty_train.data[k])
             for k in range(len(twenty_train.data)))
    count = 0
    print('Importing data.....')
    for elem in train:
        count += 1
        client.create_event(
            event = "documents",
            entity_id = count,
            entity_type = "source",
            properties = {
                "label": elem[0],
                "text": elem[1]
            })
    print("Imported {0} events.".format(count))
```