

Open Source Analytics

# NBA Real Time Prediction using Spark

Dustin VanStee  
@dustinvanstee

---



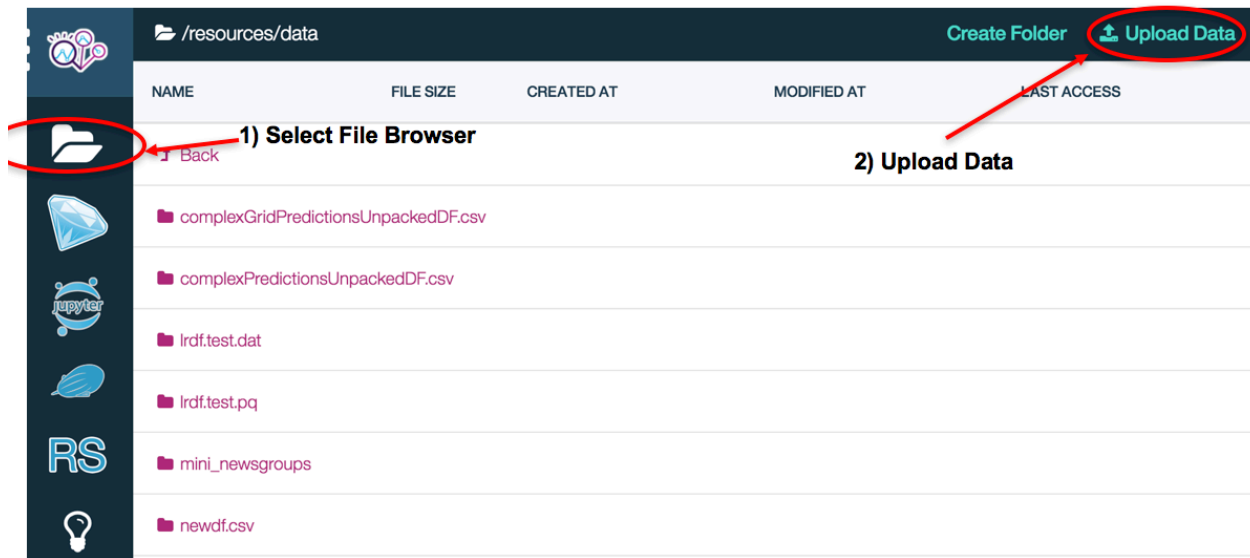
This document is written so that the user can reproduce the NBA real time machine learning demo. The demo consists of 2 parts

- datascientistworkbench (**DSWB**) zeppelin notebook for the data scientist
- Web application desired for user interaction on Bluemix

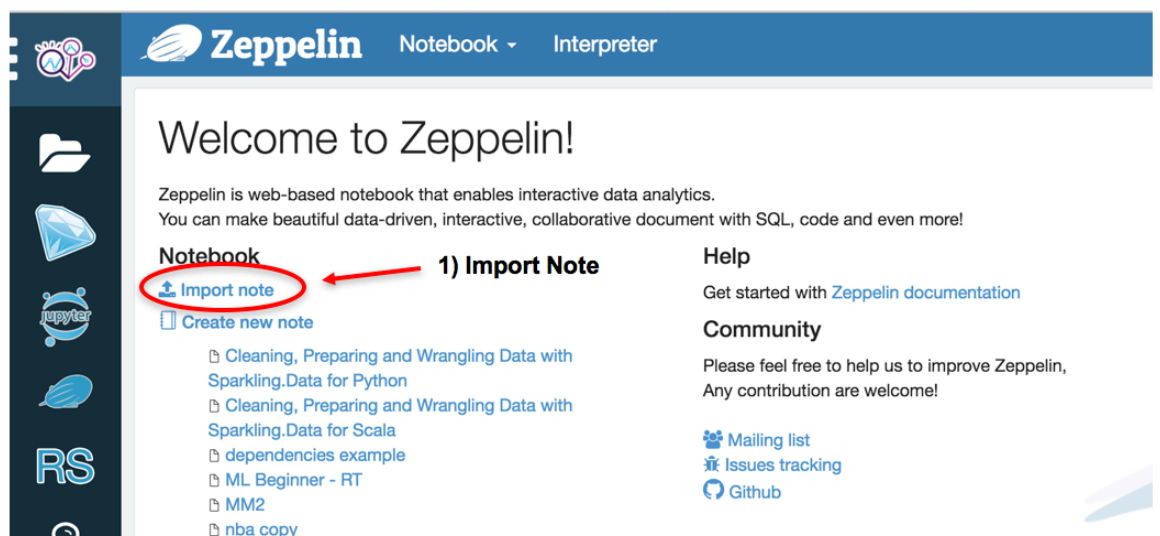
After the environment is setup, there will be a brief description of the Logistic Regression model that was built, and how it was 'wired' into the Bluemix environment. Upon completion, you will have a full demo environment setup, with a live website.

## Reproduce the DSWB Environment

1. download <https://github.com/dustinvanstee/nba-rt-prediction> project
2. Reproduce DSWB environment
  - a. For the zeppelin notebook in datascientistworkbench, you will need 3 files.
    - i. Copy nbaodds.xml /data/resources
    - ii. Copy scores\_nba.test.dat to /data/resources



iii. Import NBA-Predictions.json into your zeppelin Environment



Congratulations, you can now run the rest of the demo from within the notebook!

**INFO:** This completes the Analytics Setup portion of the document

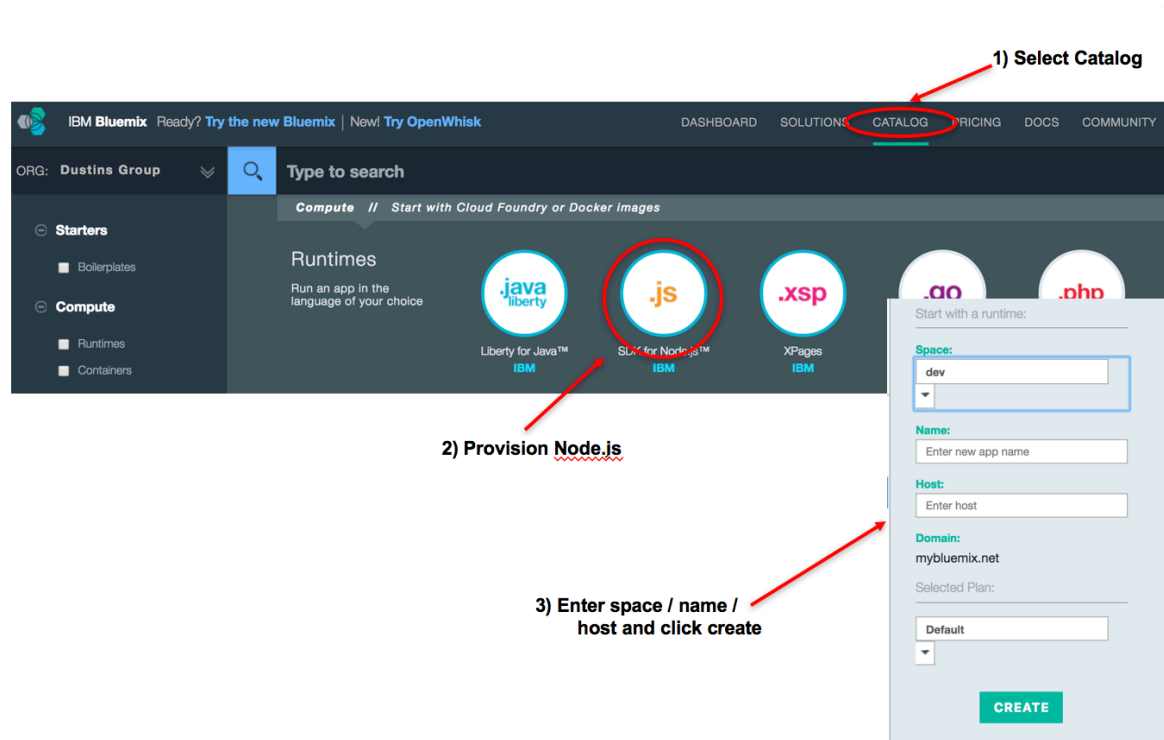
## Reproduce the Bluemix Web Application Environment

### Software pre-requisites

- Install Node Package Manager (npm) - <https://docs.npmjs.com/getting-started/installing-node>
- Setup a Bluemix account - <https://console.ng.bluemix.net/registration>

# Bluemix Configuration and Setup

To setup the web application portion of your demo, we will use the node.js framework in Bluemix. This framework provides convenient method to develop and test our application on your local machine, and then deploy the finished product to Bluemix. The first step will be to deploy the Node.js framework from within Bluemix as shown below



10

While the service is being deployed, your screen will display the instructions for the next steps. You will need to install the following tools so that you can push your web application to Bluemix. Follow the instructions to install those 2 applications to your computer.

Before you begin, install the IBM® Bluemix® and Cloud Foundry command line interfaces.



From here, we will deviate slightly from the directions in Step 1 of the getting started. Instead of downloading the starter app, you will use the app downloaded earlier from github.

Unzip the Node.js web application and move the directory to the desired location on your computer

```
tar -zxvf nodejs.tar.gz
mv nodejs/ <directoryOfYourChoice>
```

Next, we will follow the directions as laid out in the getting started guide for node.js in Bluemix. Here are the commands I typed to get my temporary node.js environment working....

```
bluemix api https://api.ng.bluemix.net
```

```
Invoke 'cf api https://api.ng.bluemix.net'...
Setting api endpoint to https://api.ng.bluemix.net...
OK

API endpoint:  https://api.ng.bluemix.net (API version: 2.44.0)
User:          vanstee@us.ibm.com
Org:          vanstee@us.ibm.com
Space:        dev
```

```
cd /tmp/nodejs/
# use cloud foundry to push my application to my Node.js
# instance name in bluemix (named temp-temp-temp)
cf push temp-temp-temp
```

```
Updating app temp-temp-temp in org Dustins Group / space dev as vanstee@us.ibm.com... cf
push temp-temp-temp

OK

Uploading temp-temp-temp...
Uploading app files from: /private/tmp/nodejs
Uploading 4.8M, 1271 files
Done uploading
OK

Stopping app temp-temp-temp in org Dustins Group / space dev as vanstee@us.ibm.com...
OK

Starting app temp-temp-temp in org Dustins Group / space dev as vanstee@us.ibm.com...
-----> Downloaded app package (7.2M)
-----> Downloaded app buildpack cache (456K)

-----> IBM SDK for Node.js Buildpack v3.3-20160428-1409
        Based on Cloud Foundry Node.js Bui
ldpack v1.5.4
-----> Creating runtime environment
```

Once this is complete, you have now deployed your web application !

Express HTML Home About Sitemap

# NBA Real Time Predictions

Simple driver for NBA scoring service

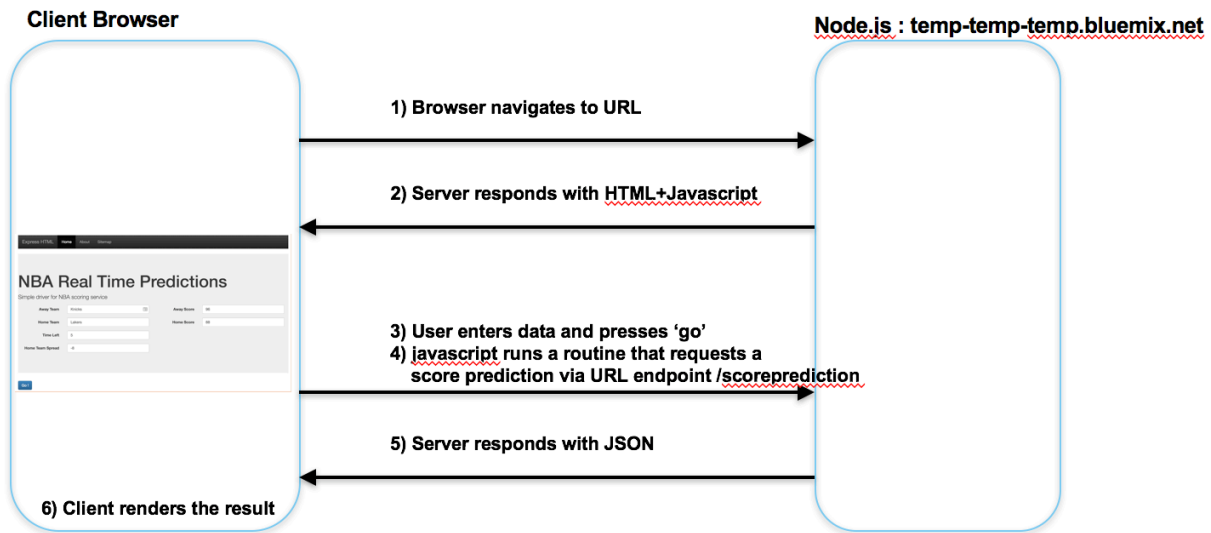
Away Team	<input type="text" value="Knicks"/>	Away Score	<input type="text" value="96"/>
Home Team	<input type="text" value="Lakers"/>	Home Score	<input type="text" value="88"/>
Time Left	<input type="text" value="5"/>		
Home Team Spread	<input type="text" value="-8"/>		

Go !

When you point your browser to this URL, the node.js server that is listening at that URL responds with this webpage (coded with Bootstrap). You can enter some data into the cells and click ‘go’.

Once you click to, some javascript is run on the browser client to send a request to the node.js service to score this outcome based on the point spread, time left, and home/away score. Some JSON is returned from the node.js service and formatted in a simple tabular view. Here is a diagram the represents what is going on ‘under the hood’

## Node.js Implementation Diagram



## Code layout

There are 3 main files that contain most of the custom code for this project

app.js : node.js file that uses express library to create a rest API  
public/index.html : NBA real time prediction web page  
public/rtp.js : javascript function that is invoked by clicking 'go'

I will briefly cover the the app.js file. This file make use of a few libraries, with the main library being express. ExpressJs provides all the web server functions you need to make a web server without have to know all the details of the underlying framework. The custom part of the code is this code below. Here you can see that I have 2 endpoints under my base URL defined. When a browser navigates to the base URL + / , the main index.html page is return. If a user were to browse to the base URL + /scoreprediction/ , the scoreprediction service is run with 3 inputs that are passed as part of the URL. This code executes the logistic regression scoring model (more details on that in the next section), and returns an answer back to the client in JSON format.

```

app.get('/', function(req, res) {
  res.sendFile(path.join(__dirname+'/index.html'));
});

app.get('/scoreprediction/:sd/:tl/:spr', function(req, res) {
  var intercept = 0.03437454446866164
  var weights = [0.07777566101325575,0.0061780161634745045,-0.10528919257886181,0.09145774467717606]
  var sdt = parseFloat(req.params.sd) / Math.pow( (parseFloat(req.params.tl) / 8) + 0.1, 0.5)
  var e0X = Math.exp(intercept + weights[0]*parseFloat(req.params.sd)
    + weights[1]*parseFloat(req.params.tl)
    + weights[2]*parseFloat(req.params.spr)
    + weights[3]*sdt
  );

  var probability = 1 / (1 + e0X)

  var data = []

  data.push({
    e0X : e0X,
    probability : probability,
    sdt : sdt
  })
  res.json(data);
});

```

The webpage was built using a nice web framework called bootstrap. I like it because the formatting and look of the pages is very moderation and fairly easy to manipulate. I won't go into the gory detail for the webpage and javascript, but will just highlight the one line of code that is activated when the 'go' button is pushed

```

<form class="form-horizontal container" role="form"
onSubmit="return
handleClick(this.homescorertp.value,this.awayscorertp.value,this.t
imelefttrtp.value,this.spreadrtp.value,this.homertp.value,this.away
rtp.value)">

```

What happens is that a handleClick function gets called with the values that are in the form of the webpage. This function is coded in the rtp.js. The rtp.js:handleClick function calls the /scoreprediction service on the server side and then waits (asynchronously) for the JSON data to be return. The result values are shown in the results table.

## Code Modification

There is one dependency that needs to get updated in the public/rtp.js file so that you can use your own score prediction service instead of the one I setup (I have a hardcoded URL in the code). To fix this modify this line of code to point to your web URL (this is the value you typed in for the name of your Node.js project when you created it in Bluemix).

```

9   var mytestquery = jQuery.ajax({
10     //url: 'http://localhost:6001/scoreprediction/' + sd + '/' + tl + '/' + spr,
11     url: 'http://nba-rt-prediction.mybluemix.net/scoreprediction/' + sd + '/' + tl + '/' + spr,

```

Once you have done this, you can push this change to Bluemix by running this command in your node.js project directory.

```
cf push nba-rt-prediction
```

This will copy the entire project back up to Bluemix and redeploy your app. You can also run the app locally for debugging purposes if you type this

```
node app.js
```

This will setup the webserver locally on your laptop and you can point your browser to localhost:6001 for local debugging.

## How to ‘Operationalize’ Machine Learning Insights

So far, we have been able to reproduce the environment, but let’s have a deeper look at what is happening in this demo environment. In the Zeppelin workbook, we went through a number of data refinement steps to produce an input data set. We then took that input data set and split it into a training and test partition that we fed into the logistic regression modelling functions within Spark MLLIB. Simply stated, the logistic regression model returns a set of weights that transform the inputs into the predicted output. In our test we had just 4 ‘features’. Here is what the logistic regression output summary looks like in the zeppelin notebook.

```
Tranining Samples = 1923
Test      Samples = 1304
Cross Val Samples = 27
Reg Parameter:    =0.01
lrModel.intercept = -0.237840249165419
lrModel.weights   = [0.08305696167072295,0.014121472627424681,-0.10720674340147919,0.09154865436260184]
Test Error = 0.15567484662576692
complexPredictionDF: org.apache.spark.sql.DataFrame = [label: double, features: vector, rawPrediction: ve
complexModel: org.apache.spark.ml.classification.LogisticRegressionModel = logreg_5c4c5fda5cba
```

For the model, the important values to focus on are the intercept and the weights. For this specific example the weights will scale to the following input variables [score\_difference, time\_left, spread, custom\_score\_diff].

To implement simple logistic regression in our Node.js web application, we just need to know how to apply these weights to the logistic regression formula. We implement the formula using this equation

$$f(X;\theta) = \text{sigmoid}(\theta^T X)$$

where

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$



Explicitly in this example the inputs from the web form (X) are multiplied by the weights (theta) as follows

$$\theta^T X = -0.23 + 0.08 * X_1 + 0.014 * X_2 + -0.107 * X_3 + 0.09 * X_4$$

We then take the number apply the sigmoid function, and out comes a probability. The higher the number the more likely the home team will win. A value near 0.50 means that our model has no more predictive value than flipping a coin. The zeppelin notebook has some commentary about the weights, so I will omit the discussion here.

In practice, a pattern like this works well enough to get the job done when the number of inputs to the model are small, but for models that are more complex(neural nets, svm) or updated frequently this manual coding approach doesn't work so well. We benefitted from the fact that the logistic equation can be coded in a few lines of code, but we should not always expect that to be the case. There are some other alternatives I will discuss in the next section ....

## Extensions to this demo ...

Extensions to what has been shown here include methods to connect our Node.js framework directly to spark for execution of the scoring. Frameworks like EclairJS look very interesting from this perspective. For the next revision of this demo I plan try to implement something like that. Here is a brief list of some other improvements ....

- Implement machine learning pipeline to select best hyper parameters for model
- Experiment with linear regression to predict final score
- Assess prediction sensitivity and margin of error (variance)