# FORECASTING DE DEMANDA

UFRJ, Escola Politécnica, DEL

André Teixeira dos Santos

Big Data - Assis Bento - 2016/2

Relatório do Trabalho

# SUMÁRIO

Este documento descreve e apresenta o resultado do trabalho da disciplina de Big Data, ministrada por Assis Bento. O trabalho atingiu o objetivo geral de gerar previsões da demanda, consultadas através de um webapp, com um ETL e o modelo de previsão sendo feitos com Spark & Scala.

# VISÃO GERAL

O objetivo do trabalho era gerar uma previsão (forecast) da demanda dos componentes utilizados numa fábrica. O usuário, hipotético, poderia consultar a previsão da demanda, para um componente específico, através de um webapp. A previsão é gerada para 3 meses a frente, à partir do último dado disponível na série histórica. Na figura abaixo, o último dado é fevereiro de 2015, e a previsão gera para março, maio e abril de 2015.

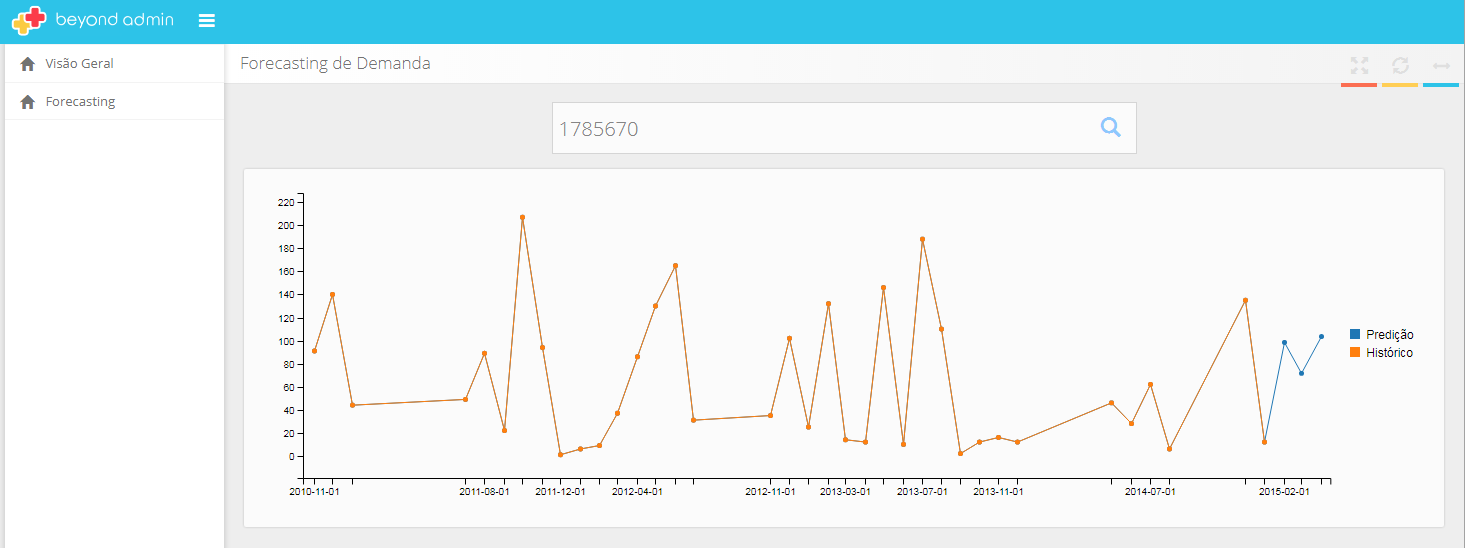


Figura - Tela do webapp para consulta de previsão de demanda. No gráfico, em laranja, a série história do componente 1785670, em azul, a previsão gerada.

Essa solução é composta por 3 componentes principais:

1. **Spark-Scala Scripts** - Conjunto de scripts em Scala, que utilizam as APIs do Spark de RDD, Dataframe & Machine Learning, para transformar os dados brutos, carregá-los no CouchDB e gerar o modelo de predição linear.
2. **CouchDB** - Banco NoSQL com API REST nativa. Os dados processados ficam disponíveis para consumo através da API REST. Além disso uma view também é instalada no CouchDB, para facilitar a comunicação com o WebApp.
3. **WebApp** - Applicação Web (HTML + Javascript) que consome os dados no CouchDB e renderiza para o usuário.

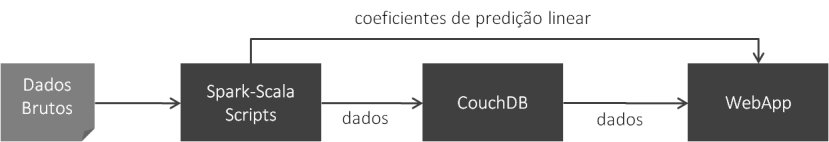


Figura Visão geral da arquitetura

Os compontes são mais detalhados nas próximas seções do relatório.

# DADOS

Os dados utilizados para análise e treinamento do modelo estavam em formato CSV, tinham cerca de 5 anos de histórico, 7GB e 5 milhões de linhas. Durante a exportação de Oracle para CSV, houve um problema com o separador de decimais e algumas linhas tiveram que ser removidas durante o processamento. As colunas principais dessa base, chamada de LINES, eram:

* **ID do Componente**: Identificador único de um componente
* **Data da Requisição**: Data em que o componente foi demandado
* **Quantidade**: Quantidade demandada

A granularidade temporal dos dados era dia/hora. Para fins de análise, isso foi agregado no mês e o modelo de previsão de demanda prevê a demanda no mês.

Por questão de confidencialidade, os dados não estão no repositório.

# SPARK-SCALA SCRIPTS

Os scripts para processamento dos dados (suposto ETL), foram feitos na linguagem Scala e utilizam as APIs do Spark de RDD, Dataframe e Machine Learning. É importante notar que não foi gerado um app do Scala/Spark, i.e., os arquivos (scripts) \*.scala são executados através do spark-shell utilizando o comando :load, conforme o exemplo abaixo. A motivação para essa abordagem foi a flexibilidade e velocidade para desenvolver o ETL. Porém seria fácil juntar os script num app do Scala/Spark.



Figura - Exemplo de excução de um script Scala

Existem no total 5 arquivos (scripts), que devem ser executados em ordem, para o processamento de dados ocorrer. Cada script é detalhado, em ordem, nas próximas seções

*1-setup.scala*

Esse script carrega os dados brutos num RDD, filtra linhas corrompidas, e extrai 10 colunas mais relevantes (do total de 340) e por último, salva o RDD final como um novo CSV. Além disso esse script também faz algumas operações de agregação no RDD final para extrair a série histórica de um componente específico e salva num CSV. Esse último passo foi para fins de análise e não faz parte do processo de ETL.

*2-data-cleansing.scala*

Esse script carrega os dados preparado no passo anterior e faz uma validação a mais nas linhas, além de reescrever o cabeçalho do csv e trocar o separador de vírgula para tabulção.

*3-transform-data.scala*

Script principal. Esse script carrega o arquivo CSV gerado no passo anterior como um Dataframe ao invés de RDD. A partir daí, diversas operações são feitas sobre esses dados, utilizando SQL, para gerar a versão final dos dados necessário para treinamento do modelo de previsão. Exemplo de operações feitas são: agregação dos dados no mês, formatação de datas, inclusão de colunas com valores do mês anterios para treinamento do modelo autoregressivo. No final do script o Dataframe final é salvo num arquivo no formato parquet.

*4-load-couchdb.scala*

Esse script carrega algumas colunas do arquivo parquet gerado no passo anterior para o CouchDB, através de um http client.



Figura - Amostra dos dados carregados no CouchDB, com uma coluna identificando o componente (item), uma com o volume e outras com a data em diferentes estruturas. A coluna date não foi utilizada pelo webapp.

Para esse script específico é necessário um passo a mais antes de carrega-lo no spark-shell. Isto é necessário por que ele utiliza uma biblioteca não-nativa do Spark/Scala, para comunicação HTTP (A carga no CouchDB é feita através de HTTP POST). Para instalar essa dependência no contexto do spark-shell é necessario o comando abaixo.



Figura - Comando para instalar uma dependência externa no contexto do spark-shell. Caso fosse um app Scala/Spark, essa dependência poderia estar no arquivo .sbt.

*5-model.scala*

Esse script gera o modelo ARIMA (2, 0, 1) de previsão de demanda. O modelo utilizado, ARIMA (2, 0, 1), foi decidido através de análises de séries temporais, feitas em R, em um subset dos dados originais. Os scripts R não estão no repositório, vão ser inclusos depois (estavam em outra máquina no momento da redação deste relatório). A motivação para fazer essa análise em R se deve ao fato que seleção de modelo é um processo exploratório e dinâmico e para isso, R é mais apropriado.

Uma vez descoberto qual modelo (linear) de previsão utilizar, este foi implementado utilizando as APIs de Machine Learning do Spark (SparkML). As APIs do SparkML, para o caso de regressão linear, quererem que os dados estejam numa estrutura específica. Essa conversão também é feita nesse script. Após isso, o modelo é treinado e os coeficiente são exibidos no shell. É importante notar que a idéia inicial era fazer um modelo para cada componente mas, por restrições de tempo de processamento & deadlines, foi-se gerado somente um modelo para todos componentes. Os coeficientes de predição linear calculados pelo SparkML foram manualmente configurados no WebApp.

# COUCHDB

O CouchDB foi instalado e duas coisas foram configuradas. Primeiro criou-se um banco, chamado de *timeseries*. Segundo, crio-se uma view de map/reduce (conceito do CouchDB) nesse banco, para simplificar a API para o WebApp. O código dessa view se encontra no repositório, mas segue snippet abaixo.

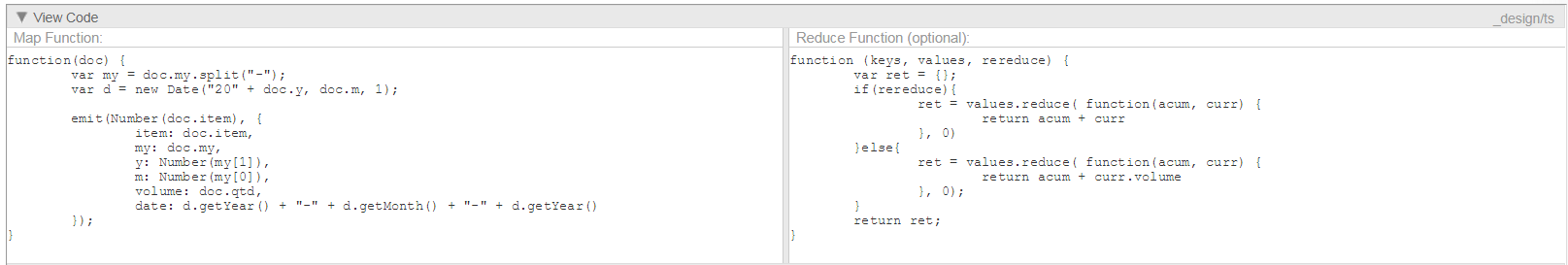


Figura - Snippet da view de map/reduce no CouchDB

# WEBAPP

Uma vez os dados carregado no CouchDB, é possível que o WebApp acesse os dados através de requisições Ajax na API REST do CouchDB. Para a construção do WebApp, utilizou-se um template de Bootstrap para a parte de design artístico, a bibliote jQuery de uso geral, C3js para renderização de gráfico e Moment.js para manipulação de datas. Na figura 1 pode-se ver o webapp resultante. O código do webapp está no repositório. Não é necessário nenhum processo de build para servi-lo, basta servir, por exemplo, direto do Apache (não é necessário PHP ou outra linguagem de backend para servi-lo, o webapp é feito com HTML e Javascript puros).

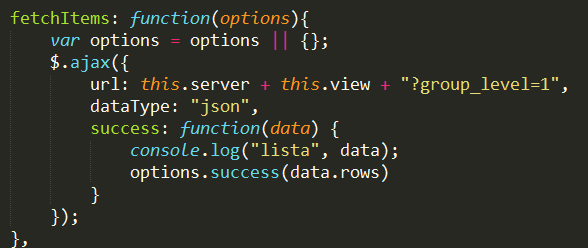


Figura - Snippet do código do webapp que faz a requisão no CouchDB para obter a lista de componentes disponíveis.

# CONCLUSÃO

O trabalho, juntamente com as aulas, foi um método excelente para aprendizagem do Spark de forma geral. Apesar de todos os requisitos/desejos inicias não terem sidos atendidos (e.g., um modelo por componente), o resultado final ficou interessante e usável. Algumas críticas e observações pessoais são sintetizadas abaixo.

1. **Falta de sinergia entre Dataframes e SparkML**. Foi necessário uma conversão de dados.
2. **CouchDB não foi a melhor escolha de banco de saída**. Apesar de ser interessante para o frontend um banco já com uma API REST, a carga Spark/Scala -> CouchDB foi extremamente lenta, pois o CouchDB não tem um método de carga massiva de dados, sendo que a carga teve que ser feita através de requisições HTTP.
3. **Necessário modelos de mais alto nível no SparkML**. O SparkML conta com modelos de aprendizagem de máquina muito gerais e de baixo nível de abstração. Por exemplo, o SparkML tem o modelo de regressão linear, mas não tem uma API para modelagem de versões autoregressivas do modelo linear, como o ARIMA. Isso significa que para produzir um modelo ARIMA, é necessário preparar os dados da série temporal de acordo com a versão do ARIMA que se deseja utilizar, o que inviabiliza a seleção de modelos através do SparkML, pois esse processo é exploratório e dinâmico.
4. **Boa API de Dataframes**. As APIs de Dataframes funcionaram muito bem na minha opnião e tornaram o desenvolvimento mais fluído.