Tarefa Final

https://github.com/lzanotto/hmr2020

Nome: Marcos Roberto Andreasi.

1. O que é GPU e para que é usada.

A NVIDIA líder na produção de hardware para o mundo dos games percebeu que as GPU's (Graphical Processing Units ou Unidades de Processamento Gráfico), normalmente utilizadas para renderização de imagens e vídeos, poderiam ser usadas para resolução de problemas matemáticos que necessitavam de maior poder computacional. Percebendo seu potencial, a NVIDIA desenvolveu as GPU's para uso geral e a linguagem de programação CUDA (Compute Unified Device Architecture), contribuindo para que os poderosos processadores gráficos sejam usados para processar dados. Desde sistemas embarcados até a utilização por usuários domésticos ou mesmo supercomputadores, seu código pode interagir diretamente com a GPU.

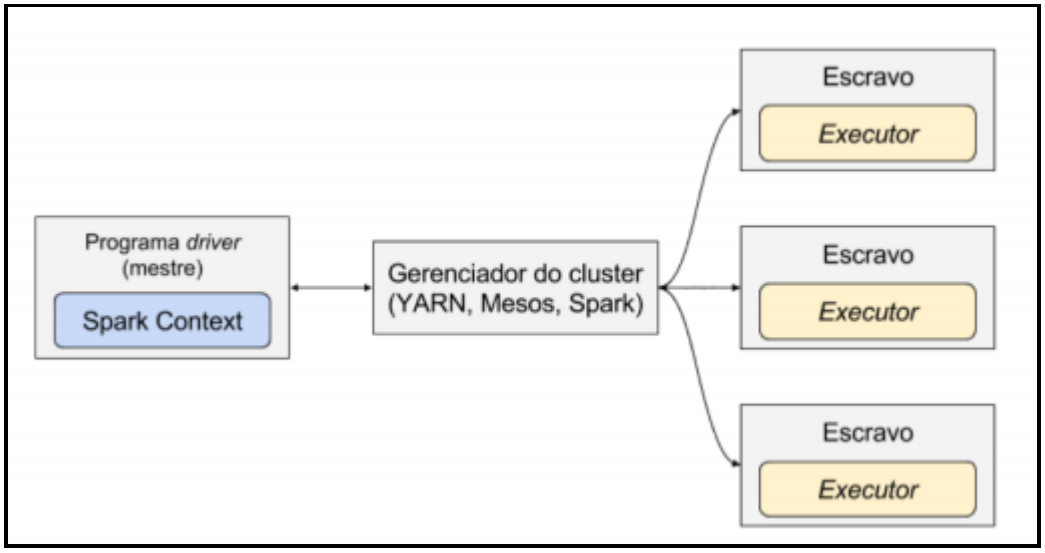
Suas aplicações possuem: restauração de imagens, segmentação, filtragem, interpolação e reconstrução. Literalmente as GPU's são chips de computador que realizam cálculos matemáticos de forma veloz e paralelizada.

Podemos dizer que a computação otimizada via GPU é o uso de unidades de processamento gráfico ou seja a (GPU's) em conjunto com as CPU's, a fim de resolver com maior velocidade análise científica, analytics, engenharia e aplicações. Resumidamente o que faz uma GPU, é receber parte da aplicação no qual o processamento requer maior uso intensivo de computação, enquanto a CPU recebe o restante. Com isso ela consegue paralelizar a execução, fazendo com que as aplicações fiquem mais velozes.

2. Como o Spark Utiliza a GPU e mostre sua arquitetura.

A arquitetura do Spark é basicamente proposta no conceito mestre e escravo, sendo o mestre responsável pelo gerenciamento das execuções de aplicações no cluster. Ao enviar uma aplicação, o mestre executa o programa driver (Driver Program) o qual contém uma instância do Spark Context e os demais códigos referentes às aplicações (RDDs, transformações e ações). Ele

tem a função de converter as aplicações em unidades denominadas tasks e organizar a execução dessas unidades nas máquinas escravas, que são denominadas executores (trabalhadores). Os executores são responsáveis por realizarem as tarefas que lhe foram determinadas e reportar os resultados para o programa driver em um nó mestre. O Spark por sua vez depende de um gerenciador de cluster (Cluster Manager) para inicializar os executores, o qual pode ser o Hadoop YARN, Mesos ou seu próprio gerenciador de cluster.



O Spark utiliza-se da GPU uma vez que permite que os usuários escrevam programas em um sistema de computação de cluster, os quais podem executar operações sobre dados em paralelo. O Spark, entretanto, representa grandes conjuntos de dados como RDDs - coleções de objetos distribuídos e imutáveis - que são armazenados nos executores (ou nós escravos). Os objetos que compõem RDDs são chamados de partições (chunk) e podem ser ou não computados em diferentes nós de um sistema distribuído.

3. O que é Stream e como o Spark utiliza o Stream.

E´ uma extensão do normalmente utilizada para processamento de dados em tempo real, com recursos interessantes como escalabilidade e a tolerância a falhas, também possui processamento único e a integração entre os processos batch em tempo de real. Seu funcionamento parte do princípio de dividir os dados em lotes podendo ser de n milissegundos, segundos ou minutos e processando cada um destes lotes como um RDD. O resultado também é retornado em lotes. O interessante é que a quantidade lotes que será processado pode ser

totalmente parametrizável. Na criação de Stream de dados isso se baseia no contexto de D Stream (Discretized Stream). Utilizando-se destes objetos podem ser criados por meio de uma Unix Sockets ou de outras fontes, como por meio do Flume, Kafka, Twitter etc. Observamos dois tipos de operações nas D Stream a de transformação e a de saída, normalmente utilizada para salvar os resultados e funcionando de maneira parecida com as de ação para as RDD's. Outro conceito interessante são as de operações em sliding window, por exemplo, countByWindow, que irá contar a quantidade de observações sempre de tempos em tempos, definido como parâmetro. O exemplo abaixo apresenta contagem em 5 em 5 segundos, de todo o post do Twitter que contém a palavra Spark.

```
TwitterUtils.createStream(...)
    .filter(_.getText.contains("Spark"))
    .countByWindow(Seconds(5))
```

Scala Streaming, contagem em sliding window.

Para prevenção as falhas, automaticamente, por meio do recurso default, ele persiste e replica os dados duas vezes. Com isso, se algum computador cair ou falhar, todo o processamento poderá ser refeito.

4. **O que é Machine Learning cite como o Spark utiliza sua biblioteca Mlib para acelerar o processamento**.

Machine Learning (ML) é uma das mais importantes áreas da inteligência artificial com a criação de algoritmos específicos podemos ensinar uma determinada máquina a desempenhar tarefas. Estes por sua vez possibilita pegar um conjunto de dados de entrada e com base em determinados padrões encontrados gerar as saídas. Cada entrada desse conjunto de dados possui suas próprias features, ter um conjunto de padrões é o ponto inicial para qualquer aprendizado de máquina.

O aprendizado é baseado na observação dos dados como: exemplos podemos citar: uma experiência direta ou uma instrução. Uma vez que tenham aprendido a reconhecer os padrões, são capazes de executar tarefas complexas e dinâmicas, prever com mais precisão, reagir em situações diversas e comportar-se de forma inteligente.

Podemos dividir os principais métodos de aprendizados que são:

Aprendizado supervisionado: no qual os exemplos são rotulados. O algoritmo analisa um conjunto de entradas juntamente com as saídas corretas correspondentes, e ele aprende através da comparação das saídas real com as corretas Em seguida, ele modifica o modelo de acordo.

Aprendizado não supervisionado: Ao contrário do aprendizado supervisionado os dados não possuem rótulos históricos. Com isso, o sistema não sabe a “resposta certa” nesse caso. Acarretando que o algoritmo deve descobrir o que está sendo mostrado e o objetivo é explorar os dados e assim encontrar alguma estrutura neles.

aprendizado semi-supervisionado: pode ser utilizado para as mesmas aplicações que o aprendizado supervisionado, entretanto os rótulos podem ser marcados ou não para o treinamento. Geralmente uma pequena quantidade de dados rotulados com uma grande quantidade de dados não rotulados (dados não rotulados são mais baratos). Esse tipo de aprendizagem pode ser usado com métodos estatísticos: classificação, regressão e previsão.

aprendizado por reforço: este tipo de aprendizado normalmente é utilizado para a robótica, jogos e navegação. Com o aprendizado por reforço, o algoritmo descobre por meio de tentativa e erro quais ações geram as melhores recompensas.

Em se tratando de processos iterativos, estamos nos referindo a algoritmos de aprendizado de máquina, que literalmente apresentam processos de otimização iterativos. MLLib está disponível em linguagem de programação em Python, Scala e Java. No qual consiste em uma biblioteca de códigos de Machine Learning prontos e disponíveis para uso, funcionando de forma semelhantes aos pacotes do R, Numpy, Scikit-learn do Python. Estudos revelam que estes algoritmos são cerca de 100 vezes mais rápidos do que suas respectivas versões em Map Reduce no Hadoop. Dentre os principais conjuntos de algoritmos podemos destacar o SVM, regressão logística, regressão linear, árvores de

decisão, sistemas de recomendação, agrupamento por diversas técnicas, SVD e Naive Bayes. Com o exemplo abaixo, podemos observar seu uso por meio da construção de um método de agrupamento usando a técnica K-Means com 10 grupos.

```
points = spark.textFile("hdfs://...")
               .map(parsePoint)

model = KMeans.train(points, k=10)
```

Clustering usando K-Means com o MLLib.

5. **Reproduza um código da Web utilizando SQL com Python (pyspark). Submeta o código no github. Deixe seus comentários sobre o desenvolvimento aqui.**

Inicializar o SQL:

```
from pyspark.sql import SparkSession
spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .getOrCreate()
```

Criar RDD:

```
rdd = sc.textFile('C:\\Users\\marco\\Downloads\\sql-mlib')
```

Lê o arquivo cvs com sqlContext.read.csv. Usar inferSchema definido como True para dizer ao Spark para adivinhar automaticamente o tipo de dados.

```
sqlContext.read.csv (SparkFiles.get ("adult_data.csv"), header = True, inferSchema = True)
```

Tipo de dados:

```
printSchema ()
```

```
| - idade: inteiro (anulável = verdadeiro)
| - classe de trabalho: string (nullable = true)
| - fnlwgt: integer (nullable = true)
| - educação: string (nullable = true)
| - education_num: integer (nullable = true)
| - marital: string (nullable = true)
| - ocupação: string (nullable = true)
| - relacionamento: string (nullable = true)
| - raça: string (nullable = true)
| - sexo: string (nullable = true)
| - capital_gain: integer (nullable = true)
| - capital_loss: integer (nullable = true)
| - hours_week: integer (nullable = true)
| - native_country: string (nullable = true)
| - rótulo: string (nullable = true)
```

Ver os dados com show.

show (20)

Selecionar e mostrar as linhas com selecionar e os nomes dos recursos. Como, ‘idade’ e ‘fnlwgt’ como no exemplo.

select ('idade', 'fnlwgt'). show (5)

```
+ --- + ------ +
| idade | fnlwgt |
+ --- + ------ +
| 39 77516 |
| 50 83311 |
| 38 | 215646 |
| 53 | 234721 |
| 28 | 338409 |
+ --- + ------ +
mostrando apenas as 5 primeiras linhas
```

Se quiser contar o número de ocorrências por grupo, podemos utilizar os comandos:

- groupBy ()
- contagem()

juntos. No exemplo em questão, conta o número de linhas por nível de educação.

groupBy ("education"). count (). sort ("count", ascending = True) .show ()

```
+ ------------ + ----- +
| educação | contagem |
+ ------------ + ----- +
| Pré-escolar | 51
| 1o ao 4o | 168
| 5º a 6º | 333 |
| Doutorado | 413
| 12º | 433
| 9º | 514
| Prof-escola | 576
| 7º a 8º | 646 |
| 10º | 933 |
| Assoc-acdm | 1067 |
| 11º | 1175 |
| Assoc-voc | 1382 |
| Masters | 1723 |
| Solteiros | 5355 |
| Alguma faculdade | 7291 |
| HS-grad | 10501 |
+ ------------ + ----- +
```

Resumo das estatísticas dos dados, podemos usar o comando describe (). Ele irá calcular:

- contagem
- significar
- desvio padrão
- min
- max

describe (). show ()

Estatísticas de apenas uma coluna, adicione o nome da coluna dentro do describe ()

describe ('capital_gain'). show ()

```
+ ------- + ------------------ +
| resumo | capital_gain |
+ ------- + ------------------ +
| contagem | 32561 |
| média | 1077,6488437087312 |
| stddev | 7385.292084840354 |
| min | 0 |
| max | 99999 |
+ ------- + ------------------ +
```

Outro exemplo é agrupar os dados por grupo e calcular operações estatísticas como a média.

groupby ('marital'). agg ({'capital_gain': 'mean'}). show ()

```
+ -------------------- + ------------------ +
| conjugal | média (ganho_capital) |
+ -------------------- + ------------------ +
| Separado | 535.5687804878049 |
| Nunca se casou | 376.58831788823363 |
| Casado-cônjuge-ab ... | 653.9832535885167 |
| Divorciado | 728.4148098131893 |
| Viúvo | 571.0715005035247 |
| Casado-AF-cônjuge | 432.6521739130435 |
| Cônjuge casada | 1764.8595085470085 |
+ -------------------- + ------------------ +
```

Mensagem de erro: "Exception: Java gateway process exited before sending its port number

Referência:

<https://www.guru99.com/pyspark-tutorial.html>

6. Reproduza um código da Web utilizando Stream com Python (pyspark). Submeta o código no github. Deixe seus comentários sobre o desenvolvimento aqui.

Exemplo: Contar o número de palavras em determinados dados de texto recebidos de um servidor monitorado em uma rede TCP.

Devemos importar o StreamingContext , para criar as principais funcionalidades do streaming. Em seguida criamos um StreamingContext local com dois threads de execução e intervalo de lote de 1 segundo.

```
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

sc = SparkContext("local[2]", "NetworkWordCount")
ssc = StreamingContext(sc, 1)
```

Utilizando o context, criamos um DStream que representa os dados de streaming de uma fonte TCP, especificada como nome do host (por exemplo `localhost`) e porta (por exemplo `9999`).

```
lines = ssc.socketTextStream("localhost", 9999)
```

Este comando `lines`DStream representa o fluxo de dados que será recebido do servidor de dados. Cada registro neste DStream é uma linha de texto. Em seguida, podemos dividir as linhas por espaço em palavras.

```
words = lines.flatMap(lambda line: line.split(" "))
```

`flatMap` é uma operação DStream de normalização (um para muitos), que cria um DStream gerando vários novos registros de cada registro no DStream de origem. Nesta operação, cada linha será dividida em várias palavras e o fluxo de palavras será representado como `words`DStream. Em seguida, conseguimos contar essas palavras.

```
pairs = words.map(lambda word: (word, 1))
wordCounts = pairs.reduceByKey(lambda x, y: x + y)

wordCounts.pprint()
```

O `words`DStream é posteriormente mapeado (transformação um-para-um) para um DStream de `(word, 1)` pares, que é então reduzido para obter a frequência de palavras em cada lote de dados. Finalmente, `wordCounts.pprint()` irá imprimir algumas das contagens geradas a cada segundo.

O Spark Streaming apenas configura o cálculo que executará quando for iniciado, para isso todas as transformações devem ser configuradas (código abaixo). Após todas as transformações terem sido configuradas, finalizamos com estes comandos.

```
ssc.start()
ssc.awaitTermination()
```

Se tudo estiver correto devemos observar uma tela parecida com a debaixo do exemplo:

```
# TERMINAL 1:
# Running Netcat

$ nc -lk 9999

hello world



...
```

```
# TERMINAL 2: RUNNING network_wordcount.py

$ ./bin/spark-submit examples/src/main/python/streaming/network_wordcount.py localhost 9999
...
-------------------------------------------
Time: 2014-10-14 15:25:21
-------------------------------------------
(hello,1)
(world,1)
...
```

código: network_wordcount.py <hostname> <port>

Onde: <hostname> e <port> descrevem o servidor TCP ao qual o Spark Streaming se conecta para receber dados.

Para executar este comando, primeiramente você precisa executar um servidor Netcat.

`$ nc -lk 9999`

Em seguida execute o comando:

`$ bin / spark-submit examples / src / main / python / streaming / network_wordcount.py localhost 9999`

```
from __future__ import print_function

import sys

from pyspark import SparkContext
do pyspark . importação de streaming StreamingContext

if __name__ == "__main__" :
    se len ( sys . argv ) ! = 3 :
        print ( "Uso: network_wordcount.py <hostname> <port>" , file = sys . stderr )
        sys . saída ( - 1 )
    sc = SparkContext ( appName = "PythonStreamingNetworkWordCount" )
    ssc = StreamingContext ( sc , 1 )

    linhas = ssc . socketTextStream ( sys . argv [ 1 ], int ( sys . argv [ 2 ]))
```

```
                contagens  =  linhas . flatMap ( linha lambda  : linha . split (
    "" )) \
                                . mapa ( palavra lambda  : ( palavra , 1 )) \
                                . reduzByKey ( lambda  a , b : a + b )
                contagens . pprint ()

                ssc . start ()
                ssc . awaitTermination ()
```

Mensagem de erro: "`Exception: Java gateway process exited before sending its port number`"

Referência:

< https://www.toptal.com/apache/apache-spark-streaming-twitter>

7. Reproduza um código da Web utilizando Machine Learning com Python (pyspark). Submeta o código no github. Deixe seus comentários sobre o desenvolvimento aqui. faça um gráfico mostrando a análise do resultado, envie junto no github.

Uso do MLLib para aprendizagem não supervisionada. O MLLib permite lidar com grande volume de dados, distribuindo a análise para vários clusters.

Configure contextos Spark e Spark SQL

```
In [2]:
from pyspark import SparkContext
sc = SparkContext('local[*]')
```

```
In [2]:
from pyspark.sql import SQLContext
sqlc = SQLContext(sc)
```

Importar funções do Spark MLLib

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import StandardScaler
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import PCA
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
```

```
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.clustering import GaussianMixture
from pyspark.mllib.classification import LogisticRegressionWithLBFGS,
LogisticRegressionModel
```

Obtenção dos dados

Em [4]:

```
sqlc . read . format ( 'com.databricks.spark.csv' )
     . options ( header = 'false' ,  inferschema = 'true' )
     . load ( 'data / sonar.all-data.txt' ))
```

Em [5]:

```
printSchema ()
```

```
raiz
 | - C0: duplo (anulável = verdadeiro)
 | - C1: duplo (anulável = verdadeiro)
 | - C2: duplo (anulável = verdadeiro)
 | - C3: duplo (anulável = verdadeiro)
 | - C4: duplo (anulável = verdadeiro)
 | - C5: duplo (anulável = verdadeiro)
 | - C6: duplo (anulável = verdadeiro)
 | - C7: duplo (anulável = verdadeiro)
 | - C8: duplo (anulável = verdadeiro)
 | - C9: duplo (anulável = verdadeiro)
 | - C10: duplo (anulável = verdadeiro)
 | - C11: duplo (anulável = verdadeiro)
 | - C12: duplo (anulável = verdadeiro)
 | - C13: duplo (anulável = verdadeiro)
 | - C14: duplo (anulável = verdadeiro)
 | - C15: duplo (anulável = verdadeiro)
 | - C16: duplo (anulável = verdadeiro)
 | - C17: duplo (anulável = verdadeiro)
 | - C18: duplo (anulável = verdadeiro)
 | - C19: duplo (anulável = verdadeiro)
 | - C20: duplo (anulável = verdadeiro)
 | - C21: duplo (anulável = verdadeiro)
 | - C22: duplo (anulável = verdadeiro)
 | - C23: duplo (anulável = verdadeiro)
 | - C24: duplo (anulável = verdadeiro)
 | - C25: duplo (anulável = verdadeiro)
 | - C26: duplo (anulável = verdadeiro)
 | - C27: duplo (anulável = verdadeiro)
 | - C28: duplo (anulável = verdadeiro)
 | - C29: duplo (anulável = verdadeiro)
 | - C30: duplo (anulável = verdadeiro)
 | - C31: duplo (anulável = verdadeiro)
 | - C32: duplo (anulável = verdadeiro)
 | - C33: duplo (anulável = verdadeiro)
 | - C34: duplo (anulável = verdadeiro)
 | - C35: duplo (anulável = verdadeiro)
 | - C36: duplo (anulável = verdadeiro)
 | - C37: duplo (anulável = verdadeiro)
 | - C38: duplo (anulável = verdadeiro)
 | - C39: duplo (anulável = verdadeiro)
```

```
 | - C40: duplo (anulável = verdadeiro)
 | - C41: duplo (anulável = verdadeiro)
 | - C42: duplo (anulável = verdadeiro)
 | - C43: duplo (anulável = verdadeiro)
 | - C44: duplo (anulável = verdadeiro)
 | - C45: duplo (anulável = verdadeiro)
 | - C46: duplo (anulável = verdadeiro)
 | - C47: duplo (anulável = verdadeiro)
 | - C48: duplo (anulável = verdadeiro)
 | - C49: duplo (anulável = verdadeiro)
 | - C50: duplo (anulável = verdadeiro)
 | - C51: duplo (anulável = verdadeiro)
 | - C52: duplo (anulável = verdadeiro)
 | - C53: duplo (anulável = verdadeiro)
 | - C54: duplo (anulável = verdadeiro)
 | - C55: duplo (anulável = verdadeiro)
 | - C56: duplo (anulável = verdadeiro)
 | - C57: duplo (anulável = verdadeiro)
 | - C58: duplo (anulável = verdadeiro)
 | - C59: duplo (anulável = verdadeiro)
 | - C60: string (nullable = true)
```

Pré-processamento dos dados

Em [6]:

```
withColumnRenamed ( "C60" , "rótulo" )
```

Transforma os 60 recursos em vetores

Em [7]:

```
assembler = VectorAssembler (
    inputCols = [ 'C %d ' % i para i no intervalo ( 60 )],
    outputCol = "recursos" )
output = assembler . transformar ()
```

Utiliza recursos para zerar a média e calcular o desvio padrão da unidade

Em [8]:

```
standardizer = StandardScaler ( withMean = True , withStd = True ,
                                inputCol = 'features' ,
                                outputCol = 'std_features' )
model = padronizador . ajuste ( saída )
saída = modelo . transformar ( saída )
```

Converte a ‘label’ em índice numérico

Em [9]:

```
indexer = StringIndexer ( inputCol = "label" , outputCol =
"label_idx" )
indexed = indexer . ajuste ( saída ) . transformar ( saída )
```

Extrai apenas as colunas de interesse

Em [10]:

```
sonar = indexado . select ([ 'std_features' , 'label' ,
'label_idx' ])
```

Em [11]:

```
sonar . mostrar ( n = 3 )
```

```
+ -------------------- + ----- + --------- +
| std_features | label | label_idx |
+ -------------------- + ----- + --------- +
| [-0,3985897356694 ... | R | 1,0 |
| [0,70184498705605 ... | R | 1,0 |
| [-0.1289179854363 ... | R | 1,0 |
+ -------------------- + ----- + --------- +
mostrando apenas as 3 primeiras linhas
```

Conversão dos dados

Primeiro devemos ajustar o nosso modelo Gaussian com 2 componentes aos 2 primeiros componentes principais dos dados como exemplo de aprendizado não supervisionado. O modelo GaussianMixture requer um RDD de vetores, não um DataFrame. Observamos que `pyspark`converte `numpy`arrays em vetores Spark.

Em [12]:

```
pca = PCA ( k = 2 , inputCol = "std_features" , outputCol = "pca"
)
model = pca . ajuste ( sonar )
transformado = modelo . transformar ( sonar )
```

Em [13]:

```
recursos = transformado . selecione ( 'pca' ) . rdd . map ( lambda
x : np . array ( x ))
```

Modelo de construção

Em [14]:

```
gmm = mistura gaussiana . train ( recursos , k = 2 )
```

Otimização e ajuste do modelo aos dados

Onde devemos observar que nosso modelo contém erros devido a falha no processo de limpeza da amostra para análise.

Em [15]:

```
prever  =  gmm . prever ( recursos ) . coletar ()
```

Em [16]:

```
rótulos  =  sonar . select ( 'label_idx' ) . rdd . mapa ( lambda  r :
r [ 0 ]) . coletar ()
```

Avaliação do modelo

Em [17]:

```
np . corrcoef ( prever ,  rótulos )
```

Fora [17]:

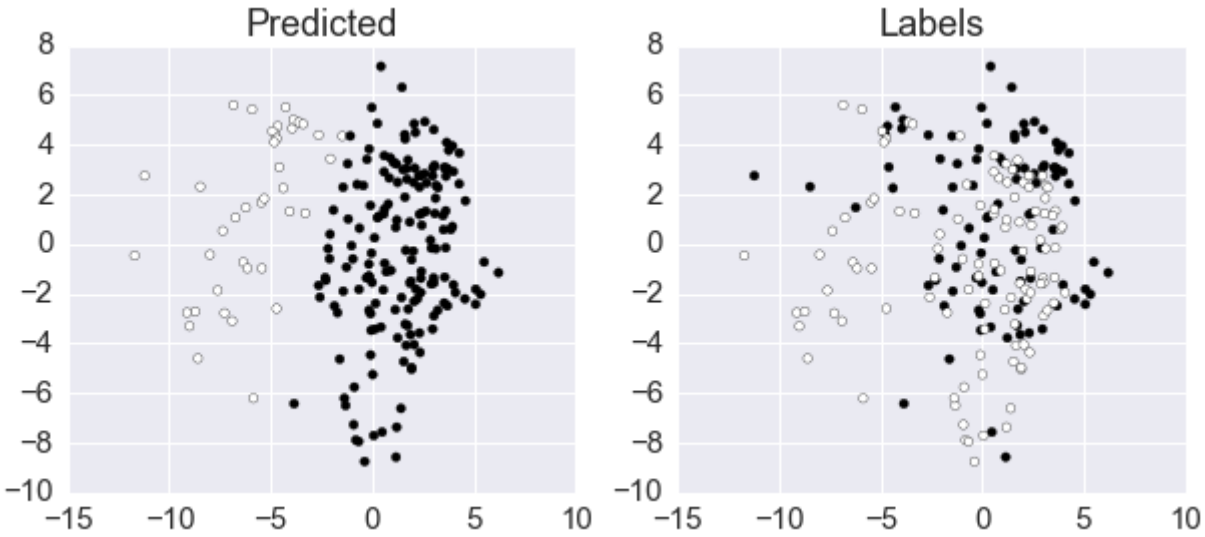
```
array ([[1., 0.13825324],
       [0,13825324, 1.]])
```

Discrepância entre os dados previsto e os rótulos

Em [18]:

```
xs  =  np . array ( features . collect ()) . squeeze ()


fig ,  eixos  =  plt . subtramas ( 1 ,  2 ,  figsize = ( 10 ,  4 ))
eixos [ 0 ] . espalhar ( xs [:,  0 ],  xs [:, 1 ],  c = prever )
eixos [ 0 ] . set_title ( 'Predicted' )
axes [ 1 ] . dispersão ( xs[:,  0 ],  xs [:, 1 ],  c = rótulos )
eixos [ 1 ] . set_title ( 'Labels' )
pass
```



Mensagem de erro: "`Exception: Java gateway process exited before sending its port number`"

Referência:

<http://people.duke.edu/~ccc14/sta-663-2016/21D_Spark_MLib.html>

Tutorial Gib + Git Hub

Referência:

<https://codefc.com.br/git-o-que-voce-precisa-saber/>

PS.: Independente da mensagem de erro, a ideia era utilizar esses tutoriais para seguir os passos e verificar seu funcionamento. A princípio não posso afirmar que tudo ocorreria conforme o proposto. Agradeço a disponibilidade e atenção do senhor em ajudar e compartilhar seus conhecimentos, no meu caso foi de grande valia. Foi bastante interessante o processo de aprendizagem no meu modo de pensar até melhor do que seria em um "mundo prefeito", utilizando o laboratório da Faculdade pois assim tivemos de pesquisar sair da curva, quebrar a cabeça para resolver determinadas situações que não ocorreriam se não estivéssemos no modo virtual devido a pandemia. Por outro lado, deixamos de ter o contato pessoal que também faz falta no processo de aprendizagem. Espero poder ter o senhor como professor em outra disciplina até o final do curso, é claro em outra circunstância não como a deste ano de 2020. Mais uma vez agradeço. Abraço.