SPARK-9953

ML Vector, Matrix semantic equality + hashcode

Grupo 05

Nome (RA):

Lucas Alves Racoci (156331)

Luiz Fernando Rodrigues da Fonseca (156475)

Rafael Zane (157079)

Rodrigo Noronha Máximo (157209)

Informações Gerais

Details:

- Tipo: Umbrella
- Status: OPEN
- Priority: Major
- Resolution: Não Resolvido
- Affects Version/s: Não Identificado
- Fix Version/s: None
- Component/s: MLlib, PySpark
- Labels: None
- Target Version/s: Não Identificado

People

- Reporter: josephkb Joseph K. Bradley
- Votes: 0 Vote for this issue
- Watchers: 3 Start watching this issue

Dates

- Criada: 13/Aug/15 23:02
- Atualizada: 06/Nov/15 18:04

Informações Gerais

Descrição:

- **MLlib** é uma biblioteca para fazer **Machine Learning** de forma escalável no Apache Spark.
- **Umbrella** para alterar a implementação dos tipos de álgebra linear local da biblioteca MLlib (Vector, Matrix, DenseVector, SparseVector, DenseMatrix, SparseMatrix) para implementar **igualdade semântica**, levando em consideração a **transposta, zeros e esparsidade**.

Igualdade e Hash

- A documentação do Python recomenda várias regras para que as classes fiquem consistentes:
  - Se a classe não define __cmp__() ou __eq__(), então não deveria definir __hash__().
  - Se a classe define objetos mutáveis e implementa __cmp__() ou __eq__(), então não deveria implementar __hash__(), pois implementações de hashable collections requerem que o hash do objeto seja imutável (se hash mudar, objeto estará no bucket errado).
- A documentação de Scala e Java descreve técnicas para sobrescrever o método equals para preservar o contrato de equals, mesmo quando subclasses adicionam novos campos. São descritas também armadilhas:
  - Definir equals em termos de campos mutáveis.
  - Falhar em definir equals como uma relação de equivalência.

Umbrella Contendo

- Bug SPARK-9792 PySpark DenseMatrix, SparseMatrix deveriam sobrescrever __eq__
  - Crítico - Quebras, perda de dados, leak de memória severo. EM PROGRESSO
- Bug SPARK-9919 Matrizes devem respeitar as convenções de igualdade e de hashCode do Java
  - Crítico - Quebras, perda de dados, leak de memória severo. EM PROGRESSO
- Bug SPARK-9793 DenseVector e SparseVector do PySpark, devem sobrescrever __eq__ e __hash__
  - Crítico - Quebras, perda de dados, leak de memória severo. RESOLVIDO
- Improvement SPARK-9940 Implementar __hash__ em DenseVector e SparseVector do PySpark
  - Major - Major loss of function. FECHADO

Bug SPARK-9792

- Antes era possível o seguinte código:

```
A = SparseMatrix(2, 2, [0, 2, 3], [0], [2])
B = DenseMatrix(2, 2, [2, 0, 0, 0])

A == B  # True
B == A  # False
```

- A primeira comparação resulta em True, já que SparseMatrix checa para igualdade semântica.
- O pull request #17968 modifica DenseMatrix para que a igualdade seja semântica também.

Bug SPARK-9792 - Pull Request #17968

Arquivo modificado python/pyspark/ml(lib)/linalg/__init__.py

```
class DenseMatrix(Matrix):
  """
  Column-major dense matrix.
  """

...
def __eq__(self, other):
-       if (not isinstance(other, DenseMatrix) or

-               self.numRows != other.numRows or
-               self.numCols != other.numCols):
          return False
        self_values = np.ravel(self.toArray(), order='F')
        other_values = np.ravel(other.toArray(), order='F')
-       return all(self_values == other_values)
```

```
class DenseMatrix(Matrix):
  """
  Column-major dense matrix.
  """

...
def __eq__(self, other):
+       if isinstance(other, SparseMatrix):
+           return np.all(self.toArray() == other.toArray())
+       if (self.numRows != other.numRows or
+           self.numCols  !=  other.numCols):
          return False
        self_values = np.ravel(  self.toArray(), order='F')
        other_values = np.ravel(other.toArray(), order='F')
+       return np.all(self_values == other_values)
```

Bug SPARK-9750 relacionado com SPARK-9792

- Mesmo problema que o anterior, só que em Scala
- Resolvida pelo pull request 8042.
- Os testes ao lado, adicionados ao arquivo MatricesSuite.scala, mostram as novas comparações que devem ser verificadas

```
+   val dm1 = Matrices.dense(2, 2, Array(0.0, 1.0, 2.0, 3.0))
+   assert(dm1 === dm1)
+   assert(dm1 !== dm1.transpose)
+
+   val dm2 = Matrices.dense(2, 2, Array(0.0, 2.0, 1.0, 3.0))
+   assert(dm1 === dm2.transpose)
+
+   val sm1 = dm1.asInstanceOf[DenseMatrix].toSparse
+   assert(sm1 === sm1)
+   assert(sm1 === dm1)
+   assert(sm1 !== sm1.transpose)
+
+   val sm2 = dm2.asInstanceOf[DenseMatrix].toSparse
+   assert(sm1 === sm2.transpose)
+   assert(sm1 === dm2.transpose)
+ }
+
```

Bug SPARK-9750

- **SparseMatrix deveria sobrescrever equals.**
- Isso garantiria que duas instâncias da mesma matriz são iguais.
- A implementação deve levar em conta a flag **isTransposed** e que os valores podem não estar na mesma ordem.

Bug SPARK-9750 - Pull Request #8042

Arquivo modificado

mllib/src/main/scala/org/apache/spark/mllib/linalg/Matrices.scala

```
@
private[spark] class MatrixUDT extends UserDefinedType[Matrix] {
...
        override def equals(o: Any): Boolean = o match {
-               case m: DenseMatrix => m.numRows == numRows && m.numCols == numCols && Arrays.equals(toArray, m.toArray)
+               case m: Matrix => toBreeze == m.toBreeze
                case _ => false
        }
...
+       override def equals(o: Any): Boolean = o match {
+               case m: Matrix => toBreeze == m.toBreeze
+               case _ => false
+       }
+
...
}
```

Bug SPARK-9919

- Testar igualdade em Java: a.equals(b) implica a.hashCode() = b.hashCode().
- Logo a e b devem implementar o hashCode.
- Problema: hashCode não deveria ser computado levando em conta todos os valores, pode ser caro.
- Ideia: usar Vector.hashCode() como base.

Bug SPARK-9919 - Pull Request #8278

68 mllib/src/main/scala/org/apache/spark/mllib/linalg/Matrices.scala

```
@@ -264,7 +264,41 @@ class DenseMatrix(
264 264   }
265 265
266 266   override def hashCode: Int = {
267     -   com.google.common.base.Objects.hashCode(numRows : Integer, numCols: Integer, toArray)
    267 +   var result: Int = 31 + numRows
    268 +   result = 31 * result + numCols
    269 +
    270 +   var i = 0
    271 +   var nnz = 0
    272 +   while (i < values.size && nnz < 64) {
    273 +     val rowInd = i % numRows
    274 +     val colInd = i / numRows
```

Bug SPARK-9919 - Pull Request #8278

```
291 +       val value = if (isTransposed) apply(rowInd, colInd) else values(i)
292 +       if (value != 0.0) {
293 +         result = 31 * result + rowInd
294 +         result = 31 * result + colInd
295 +         val bits = java.lang.Double.doubleToLongBits(value)
296 +         result = 31 * result + (bits ^ (bits >>> 32)).toInt
297 +         nnz += 1
298 +       }
299 +       i += 1
300 +     }
301 +     result
302     }
```

Bug SPARK-9919 - Pull Request #8278

```
694 +  override def hashCode: Int = {
695 +
696 +    var result: Int = 31 + numRows
697 +    result = 31 * result + numCols
698 +
699 +    var i = 0
700 +    var nnz = 0
701 +    var col_ind = 0
702 +    var ptr = 0
```

```
703 +    while (i < values.size && nnz < 64) {
704 +      while (col_ind < numCols) {
705 +        val startptr = colPtrs(col_ind)
706 +        val endptr = colPtrs(col_ind + 1)
707 +        ptr = startptr
708 +        while (ptr < endptr) {
709 +          val row_ind = rowIndices(ptr)
710 +          val value = values(ptr)
711 +          ptr += 1
712 +          if (value != 0.0) {
713 +            result = 31 * result + row_ind
714 +            result = 31 * result + col_ind
715 +            val bits = java.lang.Double.doubleToLongBits(value)
716 +            result = 31 * result + (bits ^ (bits >>> 32)).toInt
717 +            nnz += 1
718 +          }
719 +        }
720 +        col_ind += 1
721 +      }
722 +      i += 1
723 +    }
724 +    result
725 +  }
726   }
```

Bug SPARK-9793

- DenseVector e SparseVector do PySpark, devem sobrescrever __eq__ e __hash__
- Usar semântica e não representação para comparação.
- Isso deixaria o comportamento do PySpark similar ao do Scala.

Bug SPARK-9793 - Pull Request #8166

- DenseVector

```
133  class VectorUDT(UserDefinedType):
134      """
135      SQL user-defined type (UDT) for Vector.
     @@ -404,11 +412,31 @@ def __repr__(self):
412          return "DenseVector([%s])" % (', '.join(_format_float(i) for i in self.array))
413
414      def __eq__(self, other):
-         return isinstance(other, DenseVector) and np.array_equal(self.array, other.array)
415 +       if isinstance(other, DenseVector):
416 +           return np.array_equal(self.array, other.array)
417 +       elif isinstance(other, SparseVector):
418 +           if len(self) != other.size:
419 +               return False
420 +           return Vectors._equals(list(xrange(len(self))), self.array, other.indices, other.values)
421 +       return False
```

```
422
423      def __ne__(self, other):
424          return not self == other
425
426 +    def __hash__(self):
427 +        size = len(self)
428 +        result = 31 + size
429 +        nnz = 0
430 +        i = 0
431 +        while i < size and nnz < 128:
432 +            if self.array[i] != 0:
433 +                result = 31 * result + i
434 +                bits = _double_to_long_bits(self.array[i])
435 +                result = 31 * result + (bits ^ (bits >> 32))
436 +                nnz += 1
437 +            i += 1
438 +        return result
439 +
```

Bug SPARK-9940

- Implementar __hash__ em DenseVector e SparseVector do PySpark
- CLOSED
- Resolução: Duplicado
- Idêntico ao bug SPARK-9793 já resolvido
- Pull Request #8166

Conclusão

- A partir das issues analisadas, foi possível perceber que mesmo em casos envolvendo uma Umbrella, é possível que **pequenas modificações** corrijam problemas pontuais.
- Essas pequenas modificações também podem possuir um **grande impacto em execuções** de códigos, como o caso dos objetos do bug 9792, que eram iguais fazendo A == B, mas não eram iguais fazendo B == A.
- E também temos **inconsistências** entre códigos **Python e Java**, dessa vez envolvendo igualdade e hash.