Item 109
git_comments:

git_commits:

1. **summary:** [SPARK-31007][ML] KMeans optimization based on triangle-inequality
   message: [SPARK-31007][ML] KMeans optimization based on triangle-inequality ### What changes were proposed in this pull request? apply Lemma 1 in [Using the Triangle Inequality to Accelerate K-Means](https://www.aaai.org/Papers/ICML/2003/ICML03-022.pdf): > Let x be a point, and let b and c be centers. If d(b,c)>=2d(x,b) then d(x,c) >= d(x,b); It can be directly applied in EuclideanDistance, but not in CosineDistance. However, for CosineDistance we can luckily get a variant in the space of radian/angle. ### Why are the changes needed? It help improving the performance of prediction and training (mostly) ### Does this PR introduce any user-facing change? No ### How was this patch tested? existing testsuites Closes #27758 from zhengruifeng/km_triangle. Authored-by: zhengruifeng <ruifengz@foxmail.com> Signed-off-by: Sean Owen <srowen@gmail.com>

github_issues:

github_issues_comments:

github_pulls:

1. **title:** [SPARK-31007][ML] KMeans optimization based on triangle-inequality
   body: ### What changes were proposed in this pull request? apply Lemma 1 in [Using the Triangle Inequality to Accelerate K-Means](https://www.aaai.org/Papers/ICML/2003/ICML03-022.pdf): > Let x be a point, and let b and c be centers. If d(b,c)>=2d(x,b) then d(x,c) >= d(x,b); It can be directly applied in EuclideanDistance, but not in CosineDistance. However, for CosineDistance we can luckily get a variant in the space of radian/angle. ### Why are the changes needed? It help improving the performance of prediction and training (mostly) ### Does this PR introduce any user-facing change? No ### How was this patch tested? existing testsuites

github_pulls_comments:

1. In current impl, following Lemma is used in KMeans: 0, Let x be a point, let b be a center and o be the origin, then d(x,c) >= |(d(x,o) - d(c,o))| = |norm-norm(c)| ```scala val center = centers(i) // Since `\|a - b\| \geq |\|a\| - \|b\||`, we can use this lower bound to avoid unnecessary // distance computation. var lowerBoundOfSqDist = center.norm - point.norm lowerBoundOfSqDist = lowerBoundOfSqDist * lowerBoundOfSqDist if (lowerBoundOfSqDist < bestDistance) { ... } ``` this can only be applied in `EuclideanDistance`, but not in `CosineDistance` According to [Using the Triangle Inequality to Accelerate K-Meanswe](https://www.aaai.org/Papers/ICML/2003/ICML03-022.pdf) , we can go futher, and there are another two Lemmas can be used: > 1, Let x be a point, and let b and c be centers. If d(b,c)>=2d(x,b) then d(x,c) >= d(x,b); this can be applied in `EuclideanDistance`, but not in `CosineDistance`. However, for `CosineDistance` we can luckily get a variant in the space of radian/angle. This PR is mainly for Lemma 1. Its idea is quite simple, if point x is close to center b enough (less than a pre-computed radius), then we can say point x belong to center c without computing the distances between x and other centers. It can be used in both training and predction. > 2, Let x be a point, and let b and c be centers. Then d(x,c) >= max{0, d(x,b)-d(b,c)}; this can be applied in EuclideanDistance, but not in CosineDistance The application of Lemma 2 is a little complex: - It need to cache/update the distance/lower bounds to previous centers, - and thus can be only applied in training, not usable in prediction.
2. env: bin/spark-shell --driver-memory=64G testCode: ```scala import org.apache.spark.ml.linalg._ import org.apache.spark.ml.clustering._ val df = spark.read.format("libsvm").load("/data1/Datasets/webspam/webspam_wc_normalized_trigram.svm.10k") df.persist() val km = new KMeans().setK(16).setMaxIter(5) val results = Seq(2,4,8,16).map { k => val km = new KMeans().setK(k).setMaxIter(20); val start = System.currentTimeMillis; val kmm = km.fit(df); val end = System.currentTimeMillis; val train = end - start; kmm.transform(df).count; // trigger the lazy computation of radii val start2 = System.currentTimeMillis; kmm.transform(df).count; val end2 = System.currentTimeMillis; val predict = end2 - start2; val result = (k, train, kmm.summary.numIter, kmm.summary.trainingCost, predict); println(result); result } val df2 = spark.read.format("libsvm").load("/data1/Datasets/a9a/a9a") df2.persist() val km = new KMeans().setK(16).setMaxIter(10) val kmm = km.fit(df2) val results2 = Seq(2,4,8,16,32,64,128,256).map { k => val km = new KMeans().setK(k).setMaxIter(20); val start = System.currentTimeMillis; val kmm = km.fit(df2); val end = System.currentTimeMillis; val train = end - start; kmm.transform(df2).count; // trigger the lazy computation of radii val start2 = System.currentTimeMillis; kmm.transform(df).count; val end2 = System.currentTimeMillis; val predict = end2 - start2; val result = (k, train, kmm.summary.numIter, kmm.summary.trainingCost, predict); println(result); result } ``` 1, sparse dataset: webspam, numInstances: 10,000, numFeatures: 8,289,919 |Test on webspam| This PR(k=2) | This PR(k=4) | This PR(k=8) | This PR(k=16) | Master(k=2) | Master(k=4) | Master(k=8) | Master(k=16) | |------|----------|-----------|----------|-----------|----------|-----------|----------|-----------| |Train Duration (sec)|27.602|47.932|145.430|371.239|27.042|46.630|136.205|350.788| |Radii Computation (sec)|0.097|0.651|5.558|26.059|-|-|-|-| |NumIters|9|10|18|20|9|10|18|20| |Cost|5701.39583824928|4518.01202129673|4013.6152754360096|3520.6545815340055|5701.39583824928|4518.01202129673|4013.6152754360096|3520.654581 |Prediction Duration (millsec)|31|31|30|30|36|33|41|43| 2, dense dataset: a9a, numInstances: 32,561, numFeatures: 123 |Test on a9a| This PR(k=2) | This PR(k=4) | This PR(k=8) | This PR(k=16) | This PR(k=32) | This PR(k=64) | This PR(k=128) | This PR(k=256) | Master(k=2) | Master(k=4) | Master(k=8) | Master(k=16) | Master(k=32) | Master(k=64) | Master(k=238) | Master(k=3566) | |------|----------|-----------|----------|-----------|----------|-----------|----------|-----------|----------|-----------|----------|-----------|----------|-----------| |Train Duration (sec)|0.465|1.411|0.957|1.109|1.387|2.137|3.891|8.373|0.484|0.758|1.065|1.3|1.577|2.413|4.483|9.616| |Radii Computation (sec)|0.000|0.000|0.000|0.001|0.002|0.007|0.024|0.093|-|-|-|-|-|-|-|-| |NumIters|5|14|20|20|20|20|20|5|14|20|20|20|20|20| |Cost|223377.07442855154|208261.6049327395|183833.73210801493|166964.5700618612|151753.31986776151|137289.24733092127|122693.39508665689|1104 |Prediction Duration (millsec)|32|33|31|30|30|30|31|39|36|38|33|32|29|35|31| We can see that: 1, the convergence of both impl are almost the same. 2, new impl of prediction is likely to be faster than existing impl (after lazy computation of radii); 3, computation of radii may take a few seconds, so training maybe slower than existing impl in some case (few instances, large k, large dim, then the computation of radii matters in the whole training), for example, webspam with k=16, new impl took 371.239 sec, while existing impl took 350.788 sec. That is because the computation of radii (in all 20 iterations) totally took 26.059 sec.
3. **[Test build #119166 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/119166/testReport)** for PR 27758 at commit [`c423b74`](https://github.com/apache/spark/commit/c423b74eaf72c40fcaa0cc823b600ae45299d245). * This patch passes all tests. * This patch merges cleanly. * This patch adds no public classes.
4. I update the `radii` to the `statistics` including more than `radii`, so another bound can be used in both distance measurers: In `findClosest`, when cluster `i` is too far alway from current closest cluster `bestIndex` (for example, distance(cluster_i, cluster_bestIndex) > 2 * distance(cluster_bestIndex, x) in `EuclideanDistance`), then we can say that point x should not belong to cluster `i`, so no need to compute distance(cluster_i, x). Then I retest above testsuite, the prediction is a litter faster, but not significantly. I also test on **cosine-distance**, results are: |Test on webspam| This PR(k=2) | This PR(k=4) | This PR(k=8) | This PR(k=16) | Master(k=2) | Master(k=4) | Master(k=8) | Master(k=16) | |------|----------|-----------|----------|-----------|----------|-----------|----------|-----------| |Train Duration (sec)|28.851|39.434|107.571|306.996|29.291|37.332|99.98|275.32| |NumIters|10|7|11|14|10|7|11|14| |Cost|3585.915362367295|2830.5043071043824|2410.540059493046|2057.831172250597|3585.9153623672946|2830.5043071043824|2410.540059493046|2057.83 |Prediction Duration (millsec)|29|29|29|33|32|29|32|35| |Test on a9a| This PR(k=2) | This PR(k=4) | This PR(k=8) | This PR(k=16) | This PR(k=32) | This PR(k=64) | This PR(k=128) | This PR(k=256) | Master(k=2) | Master(k=4) | Master(k=8) | Master(k=16) | Master(k=32) | Master(k=64) | Master(k=238) | Master(k=3566) | |------|----------|-----------|----------|-----------|----------|-----------|----------|-----------|----------|-----------|----------|-----------|----------|-----------| |Train Duration (sec)|0.445|0.559|0.77|1.067|1.379|2.114|3.857|7.786|0.461|0.613|0.728|1.208|1.547|2.387|4.287|9.11| |NumIters|4|9|10|20|20|20|20|20|4|9|10|20|20|20|20|20| |Cost|9458.881512958757|8727.181294576074|7646.536181047704|6743.890831633205|6063.089195117649|5381.196166489522|4787.4797985497125|4275.7058 |Prediction Duration (millsec)|29|30|28|28|28|28|29|28|32|30|34|30|31|31|30|36| We can see that KMeans impls with cosine distance have the (almost) same convergen. And the prediction is about 10% faster than master.
5. **[Test build #119208 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/119208/testReport)** for PR 27758 at commit [`dd2aff7`](https://github.com/apache/spark/commit/dd2aff729a8ff4ffea34691486337f0bc3b5f16b). * This patch **fails due to an unknown error code, -9**. * This patch merges cleanly. * This patch adds no public classes.
6. If I'm reading that right, it's slower to train in many cases in the first instance, and slightly faster at scale in the second. I'm surprised because I'd have thought it's more easily a win, even with the overhead of calculating stats. Is the purpose more about prediction speed? I just wonder how much one is worth vs the other.

7. @srowen Sorry to reply late. I missed the emails from github. > Is the purpose more about prediction speed? It also help saving training time, if the dataset is large enough. Since the cost of computing stats is about O(k^2 * m), while the cost of computing distances at one iteration is O(k * n * m) where m is the number of features, and n is the number of instances; I guess I can compute the stats distributedly in some case (when k is large); I just mark this PR WIP for two reasons: 1, I will test this impl on a big dataset distributedly to check wheter above hypothesis set up; 2, for cosine distance, I want to future find a theoretical basis for the bound. Since above `Each side of a spherical triangle is less than the sum of the other two` seems is for 3-dim. I think it also right when dim>3, but it is not used in other impls. I will look for a theoretical proof.
8. PS it might be worth benchmarking again anyway, as I just merged the change that uses native BLAS for level 1 operations of vectors >= 256 elements.
9. I made a update to optimize the computation of statistics, if `k` and/or `numFeatures` are too large, compute the statistics distributedly. I retest this impl today, and I use SparkUI to profile the performance: testcode: ```scala import org.apache.spark.ml.linalg._ import org.apache.spark.ml.clustering._ var df = spark.read.format("libsvm").load("/data1/Datasets/webspam/webspam_wc_normalized_trigram.svm.10k").repartition(2) df.persist() (0 until 4).foreach{ _ => df = df.union(df) } df.count Seq(4,8,16,32).foreach{ k => new KMeans().setK(k).setMaxIter(5).fit(df) } ``` I recoded both the duration at each iteration and the _Stage_ of prediction: ![image](https://user-images.githubusercontent.com/7322292/79227674-d6a9d200-7e92-11ea-882f-f701bd24cbb0.png) results: Test on webspam | This PR(k=4) | This PR(k=8) | This PR(k=16) | This PR(k=32) | Master(k=4) | Master(k=8) | Master(k=16) | Master(k=32) -- | -- | -- | -- | -- | -- | -- | -- | -- Average iteration (sec) | 9.2+0.0 | 15.8+0.1 | 31.4+0.5 | 63.6+2 | 9.8 | 16.4 | 34.6 | 78.3 Average Prediction Stage | 6 | 10.1 | 20.6 | 44.4 | 6 | 10.8 | 22.8 | 57.2 `63.6+2` here means it took 2sec to compute those statistics distributedly, which is faster than the previous commit (computing statstics in the driver) which took about 9sec. ![image](https://user-images.githubusercontent.com/7322292/79227308-453a6000-7e92-11ea-8f06-8841266beb6e.png) When `k=4,8` the speedup is not significant, when `k=16,32` it is about 10%~30% faster in prediction Stage. It shows that the large `k` is, the realtively faster this new impl is, that is because for large `k` there is more chances to trigger the short circuits.
10. **[Test build #121275 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/121275/testReport)** for PR 27758 at commit [`b4fabb1`](https://github.com/apache/spark/commit/b4fabb1ddb9c2c1a92a0ad5c9ccbbaed4f3d0acc). * This patch passes all tests. * This patch merges cleanly. * This patch adds no public classes.
11. **[Test build #121400 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/121400/testReport)** for PR 27758 at commit [`2b10eb2`](https://github.com/apache/spark/commit/2b10eb27119de3f35529ef51b34542b7b3a01daf). * This patch passes all tests. * This patch merges cleanly. * This patch adds no public classes.
12. **[Test build #121401 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/121401/testReport)** for PR 27758 at commit [`0afc845`](https://github.com/apache/spark/commit/0afc845222a0c63d74474252b05185e3be402744). * This patch passes all tests. * This patch merges cleanly. * This patch adds no public classes.
13. **[Test build #121517 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/121517/testReport)** for PR 27758 at commit [`b050973`](https://github.com/apache/spark/commit/b05097301a574cf2d181c46e02c5e691c30782f2). * This patch passes all tests. * This patch merges cleanly. * This patch adds no public classes.
14. **[Test build #121518 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/121518/testReport)** for PR 27758 at commit [`bb4539b`](https://github.com/apache/spark/commit/bb4539bee84aa5832771cba45785cfe6b49f95a0). * This patch **fails PySpark unit tests**. * This patch merges cleanly. * This patch adds no public classes.
15. I retest this impl using the packed matrix: This PR: ![image](https://user-images.githubusercontent.com/7322292/79744652-532c2d00-8339-11ea-85fc-24d38e919981.png) Master: ![image](https://user-images.githubusercontent.com/7322292/79744614-3e4f9980-8339-11ea-9af1-a4b0b737cfb2.png) durations of the prediction _Stage_ `mapPartitions at KMeans.scala:...` at each iteration: This PR: 51sec, 49sec, 48sec, 46sec, 46sec Master: 1.1min, 57sec, 60sec, 57sec, 59sec
16. retest this please
17. **[Test build #121524 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/121524/testReport)** for PR 27758 at commit [`bb4539b`](https://github.com/apache/spark/commit/bb4539bee84aa5832771cba45785cfe6b49f95a0). * This patch passes all tests. * This patch merges cleanly. * This patch adds no public classes.
18. > In current impl, following Lemma is used in KMeans: > > 0, Let x be a point, let b be a center and o be the origin, then d(x,c) >= |(d(x,o) - d(c,o))| = |norm-norm(c)| > > ```scala > val center = centers(i) > // Since `\|a - b\| \geq |\|a\| - \|b\||`, we can use this lower bound to avoid unnecessary > // distance computation. > var lowerBoundOfSqDist = center.norm - point.norm > lowerBoundOfSqDist = lowerBoundOfSqDist * lowerBoundOfSqDist > if (lowerBoundOfSqDist < bestDistance) { > ... > } > ``` > > this can only be applied in `EuclideanDistance`, but not in `CosineDistance` > > According to [Using the Triangle Inequality to Accelerate K-Meanswe](https://www.aaai.org/Papers/ICML/2003/ICML03-022.pdf) , we can go futher, and there are another two Lemmas can be used: > > > 1, Let x be a point, and let b and c be centers. If d(b,c)>=2d(x,b) then d(x,c) >= d(x,b); > > this can be applied in `EuclideanDistance`, but not in `CosineDistance`. > However, for `CosineDistance` we can luckily get a variant in the space of radian/angle. > > This PR is mainly for Lemma 1. Its idea is quite simple, if point x is close to center b enough (less than a pre-computed radius), then we can say point x belong to center c without computing the distances between x and other centers. It can be used in both training and predction. When this condition holds, it can remove the computation of distances for other centers. If not the case, it will go to the normal K-means path? Does it hurt by introducing more overhead by pre-calculation when K is large since you need to pre-calculate when centers are updated for each iteration and the condition needs to always hold to get the benefit? > > > 2, Let x be a point, and let b and c be centers. Then d(x,c) >= max{0, d(x,b)-d(b,c)}; > > this can be applied in EuclideanDistance, but not in CosineDistance > > The application of Lemma 2 is a little complex: > > * It need to cache/update the distance/lower bounds to previous centers, > * and thus can be only applied in training, not usable in prediction.
19. @xwu99 Thanks for reviewing! > When this condition holds, it can remove the computation of distances for other centers. If not the case, it will go to the normal K-means path? Yes, this PR will use two short-circuit: 1, if point x is close to center b enough (less than a pre-computed radius), then we can say point x belong to center c without computing the distances between x and other centers. 2, in normal K-means path, suppose current closest center is `c` and distance is `p`, for next cluster `d`, if `distance(c,d)` is larger than a bound `f(p)`, then we can skip center `d`; > Does it hurt by introducing more overhead by pre-calculation when K is large since you need to pre-calculate when centers are updated for each iteration and the condition needs to always hold to get the benefit? I had optimize it by make the pre-calculation distributedly when `k*k*dim` is too large. On the test dataset with dim=8,289,919 and k=32, it took less than 2 seconds at one iteration. In most cases, it is computed in the driver, and only took ~100 millsecond. I think triangle-inequality based optimization is complementary with BLAS optimization. When we do not enable Level-3 BLAS, then the triangle-inequality approach will work.
20. **[Test build #121571 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/121571/testReport)** for PR 27758 at commit [`d31d488`](https://github.com/apache/spark/commit/d31d488e0e48a82fd5b43c406f07b8c7d27dd53c). * This patch passes all tests. * This patch merges cleanly. * This patch adds no public classes.
21. Merged to master
22. Thanks for reviewing!

github_pulls_reviews:

1. Is there any value in this default impl? it's overridden in both subclasses. Or is it to support future impls that may not be able to use this optimization?
2. Do you need the other implementation, if you have this one? this short-circuits the case where you find a point within a cluster radius. This is kept to support cosine distance?
3. Just return here rather than carry a flag around?
4. Nit: I'd just make these two vals for clarity
5. Same, just return?
6. Cosine distance doesn't obey the triangle inequality; is this meant to be angular distance? For my benefit (and possibly comments) how is r the angular distance here?
7. You might remove this after testing; it's not super important.
8. You are right, I should remove this. Not all distance can be expected to have such triangle-inequality.
9. Yes, Cosine distance doesn't obey the triangle inequality, but the following lemma should be available to apply: given a point x, and let b and c be centers. If angle(x, b)<angle(b,c)/2, then angle(x,b)<angle(x,c), cos_distance(x,b)=1-cos(x,b)<cos_distance(x,c)=1-cos(x,c) That is because: [PRINCIPLES FROM GEOMETRY, point 3](http://www.angelfire.com/nt/navtrig/B1.html) > Each side of a spherical triangle is less than the sum of the other two. [Triangle_inequality:](https://en.wikipedia.org/wiki/Triangle_inequality) > In spherical geometry, the shortest distance between two points is an arc of a great circle, but the triangle inequality holds provided the restriction is made that the distance between two points on a sphere is the length of a minor spherical line segment (that is, one with central angle in [0, π]) with those endpoints.[4][5] angle(x,b) + angle(x,c) > angle(b,c) angle(x,b) < angle(b,c)/2 => angle(x,c) > angle(b,c)/2 > angle(x,b) => cos_distance(x,c) > cos_distance(x,b) angle(x,b) < angle(b,c)/2 <=> cos(x,b) > sqrt{ (cos(b,c) + 1)/2 } <=> cos_distance(x,b) < 1 - sqrt{ (cos(b,c) + 1)/2 } = 1 - sqrt{ 1 - cos_distance(b,c) / 2 } => Give two centers b and c, if point x has cos_distance(x,b) < 1 - sqrt{ 1 - cos_distance(b,c) / 2 }, then point x belongs to center b.
10. In short, cos_distance do not obey triangle inequality, so we can **NOT** say: If cos_distance(b,x) < cos_distance(b,c)/2, then cos_distance(b,x) < cos_distance(c,x) However, the arc distance (or angle) obeys `Each side of a spherical triangle is less than the sum of the other two.`, so we can get a angular

bound, and then a cos_distance bound: if point cos_distance(b,x) < 1 - sqrt{ 1 - cos_distance(b,c) / 2 }, then cos_distance(b,x) < cos_distance(c,x)
11. OK, the last expression comes from the cos(2x) identity, OK.
12. It might be too hard to implement, but if it's symmetric you only need half this many elements to represent. The indexing becomes more complex though
13. If you're micro-optimizing, I suppose you can lift stats(i) out of the loop, but it may not o anything
14. Is there any clean way to avoid duplicating most of this code? maybe not. It looks almost identical to the above though
15. good idea, it is symmetric. I will study how other impls like GMM to store only the upper triangular part of the matrix. Maybe it is helpful to support symmetric dense matrix in .linalg? since it is used in many places
16. If this is the natural home for this, fine, though I'd usually put code used by both in .mllib. Maybe. I'm not sure anymore
17. I thought breeze or commons math had a utility method for this already but I can't find it immediately
18. So centers are ordered by statistic? And this is just a short-circuit?
19. Nit: you don't need the else here
20. do you mean: ```scala if (bestDistance < statistics(0)) { return (0, bestDistance) } ``` yes, `statistics(0)` here is just equal to `statistics(indexUpperTriangular(k, 0, 0))`
21. OK, I will look for it.
22. Yes, I think it is natural to place it in ,ml. if we keep KMeans in the .mllib side, when we do some improvement, this will happen.
23. I can't find a packing or indexing method in scala or commons-math

jira_issues:

jira_issues_comments: