PKA-KMEANS

December 12, 2023

[1]:
```
import pyspark
import os
import sys
from pyspark import SparkContext
os.environ['PYSPARK_PYTHON'] = sys.executable
os.environ['PYSPARK_DRIVER_PYTHON'] = sys.executable

from pyspark.sql import SparkSession
```

[2]:
```
spark = SparkSession.builder.config("spark.driver.memory", "16g").
 ↪appName('chapter_5').getOrCreate()
```

0.0.1 A First Take on Clustering

[3]:
```
data_without_header = spark.read.option("inferSchema", True).\
                                 option("header", False).\
                                 csv("data/kddcup.data_10_percent_corrected")

column_names = [  "duration", "protocol_type", "service", "flag",
  "src_bytes", "dst_bytes", "land", "wrong_fragment", "urgent",
  "hot", "num_failed_logins", "logged_in", "num_compromised",
  "root_shell", "su_attempted", "num_root", "num_file_creations",
  "num_shells", "num_access_files", "num_outbound_cmds",
  "is_host_login", "is_guest_login", "count", "srv_count",
  "serror_rate", "srv_serror_rate", "rerror_rate", "srv_rerror_rate",
  "same_srv_rate", "diff_srv_rate", "srv_diff_host_rate",
  "dst_host_count", "dst_host_srv_count",
  "dst_host_same_srv_rate", "dst_host_diff_srv_rate",
  "dst_host_same_src_port_rate", "dst_host_srv_diff_host_rate",
  "dst_host_serror_rate", "dst_host_srv_serror_rate",
  "dst_host_rerror_rate", "dst_host_srv_rerror_rate",
  "label"]

data = data_without_header.toDF(*column_names)
```

[4]:
```
from pyspark.sql.functions import col
data.select("label").groupBy("label").count().\
      orderBy(col("count").desc()).show(25)
```

```
+----------------+------+
|           label| count|
+----------------+------+
|          smurf.|280790|
|        neptune.|107201|
|         normal.| 97278|
|           back.|  2203|
|          satan.|  1589|
|        ipsweep.|  1247|
|      portsweep.|  1040|
|    warezclient.|  1020|
|       teardrop.|   979|
|            pod.|   264|
|           nmap.|   231|
|   guess_passwd.|    53|
|buffer_overflow.|    30|
|           land.|    21|
|    warezmaster.|    20|
|           imap.|    12|
|        rootkit.|    10|
|     loadmodule.|     9|
|      ftp_write.|     8|
|       multihop.|     7|
|            phf.|     4|
|           perl.|     3|
|            spy.|     2|
+----------------+------+
```

[5]:
```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.clustering import KMeans, KMeansModel
from pyspark.ml import Pipeline

numeric_only = data.drop("protocol_type", "service", "flag").cache()

assembler = VectorAssembler().setInputCols(numeric_only.columns[:-1]).\
                              setOutputCol("featureVector")

kmeans = KMeans().setPredictionCol("cluster").setFeaturesCol("featureVector")

pipeline = Pipeline().setStages([assembler, kmeans])
pipeline_model = pipeline.fit(numeric_only)
kmeans_model = pipeline_model.stages[1]

from pprint import pprint
pprint(kmeans_model.clusterCenters())
```

```
[array([4.79793956e+01, 1.62207883e+03, 8.68534183e+02, 4.45326100e-05,
```

```
       6.43293794e-03, 1.41694668e-05, 3.45168212e-02, 1.51815716e-04,
       1.48247035e-01, 1.02121372e-02, 1.11331525e-04, 3.64357718e-05,
       1.13517671e-02, 1.08295211e-03, 1.09307315e-04, 1.00805635e-03,
       0.00000000e+00, 0.00000000e+00, 1.38658354e-03, 3.32286248e+02,
       2.92907143e+02, 1.76685418e-01, 1.76607809e-01, 5.74330999e-02,
       5.77183920e-02, 7.91548844e-01, 2.09816404e-02, 2.89968625e-02,
       2.32470732e+02, 1.88666046e+02, 7.53781203e-01, 3.09056111e-02,
       6.01935529e-01, 6.68351484e-03, 1.76753957e-01, 1.76441622e-01,
       5.81176268e-02, 5.74111170e-02]),
 array([2.0000000e+00, 6.9337564e+08, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 1.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 5.7000000e+01,
       3.0000000e+00, 7.9000000e-01, 6.7000000e-01, 2.1000000e-01,
       3.3000000e-01, 5.0000000e-02, 3.9000000e-01, 0.0000000e+00,
       2.5500000e+02, 3.0000000e+00, 1.0000000e-02, 9.0000000e-02,
       2.2000000e-01, 0.0000000e+00, 1.8000000e-01, 6.7000000e-01,
       5.0000000e-02, 3.3000000e-01])]
```

[6]:
```
with_cluster = pipeline_model.transform(numeric_only)

with_cluster.select("cluster", "label").groupBy("cluster", "label").count().\
             orderBy(col("cluster"), col("count").desc()).show(25)
```

```
+-------+----------------+------+
|cluster|           label| count|
+-------+----------------+------+
|      0|          smurf.|280790|
|      0|        neptune.|107201|
|      0|         normal.| 97278|
|      0|           back.|  2203|
|      0|          satan.|  1589|
|      0|        ipsweep.|  1247|
|      0|      portsweep.|  1039|
|      0|    warezclient.|  1020|
|      0|       teardrop.|   979|
|      0|            pod.|   264|
|      0|           nmap.|   231|
|      0|   guess_passwd.|    53|
|      0|buffer_overflow.|    30|
|      0|           land.|    21|
|      0|    warezmaster.|    20|
|      0|           imap.|    12|
|      0|        rootkit.|    10|
|      0|     loadmodule.|     9|
|      0|      ftp_write.|     8|
|      0|       multihop.|     7|
```

```
|      0|             phf.|     4|
|      0|            perl.|     3|
|      0|             spy.|     2|
|      1|      portsweep.|     1|
+-------+----------------+------+
```

0.0.2 Choosing k

[7]:
```
from pyspark.sql import DataFrame
from random import randint

def clustering_score(input_data, k):
    input_numeric_only = input_data.drop("protocol_type", "service", "flag")
    assembler = VectorAssembler().setInputCols(input_numeric_only.columns[:-1]).
 ↪setOutputCol("featureVector")
    kmeans = KMeans().setSeed(randint(100,100000)).setK(k).
 ↪setPredictionCol("cluster").setFeaturesCol("featureVector")
    pipeline = Pipeline().setStages([assembler, kmeans])
    pipeline_model = pipeline.fit(input_numeric_only)
    kmeans_model = pipeline_model.stages[-1]
    training_cost = kmeans_model.summary.trainingCost
    return training_cost

for k in list(range(20,100, 20)):
    print(clustering_score(numeric_only, k))
```

```
24051452514440.62
6083481956357.812
5731139803832.49
20114467690413.004
```

[8]:
```
def clustering_score_1(input_data, k):
    input_numeric_only = input_data.drop("protocol_type", "service", "flag")
    assembler = VectorAssembler().\
                  setInputCols(input_numeric_only.columns[:-1]).\
                  setOutputCol("featureVector")
    kmeans = KMeans().setSeed(randint(100,100000)).setK(k).setMaxIter(40).\
      setTol(1.0e-5).\
      setPredictionCol("cluster").setFeaturesCol("featureVector")
    pipeline = Pipeline().setStages([assembler, kmeans])
    pipeline_model = pipeline.fit(input_numeric_only)
    kmeans_model = pipeline_model.stages[-1]
    training_cost = kmeans_model.summary.trainingCost
    return training_cost


for k in list(range(20,101, 20)):
```

```
    print(k, clustering_score_1(numeric_only, k))
```

```
20 24349409615712.145
40 11603516338846.06
60 9270386137063.135
80 2067945253790.8936
100 1894872748589.5337
```

0.0.3 Feature Normalization

[9]:
```
from pyspark.ml.feature import StandardScaler

def clustering_score_2(input_data, k):
    input_numeric_only = input_data.drop("protocol_type", "service", "flag")
    assembler = VectorAssembler().\
                setInputCols(input_numeric_only.columns[:-1]).\
                setOutputCol("featureVector")
    scaler = StandardScaler().setInputCol("featureVector").\
                              setOutputCol("scaledFeatureVector").\
                              setWithStd(True).setWithMean(False)
    kmeans = KMeans().setSeed(randint(100,100000)).\
                      setK(k).setMaxIter(40).\
                      setTol(1.0e-5).setPredictionCol("cluster").\
                      setFeaturesCol("scaledFeatureVector")
    pipeline = Pipeline().setStages([assembler, scaler, kmeans])
    pipeline_model = pipeline.fit(input_numeric_only)
    kmeans_model = pipeline_model.stages[-1]
    training_cost = kmeans_model.summary.trainingCost
    return training_cost

for k in list(range(60, 271, 30)):
    print(k, clustering_score_2(numeric_only, k))
```

```
60 532540.2067919375
90 342335.3184153998
120 237574.75170821618
150 180676.05740041216
180 151328.0931732155
210 132322.00848148682
240 110804.28065716835
270 100864.63100581066
```

0.0.4 Categorical Variables

[10]:
```
from pyspark.ml.feature import OneHotEncoder, StringIndexer

def one_hot_pipeline(input_col):
```

```
    indexer = StringIndexer().setInputCol(input_col).setOutputCol(input_col +␣
↪"_indexed")
    encoder = OneHotEncoder().setInputCol(input_col + "_indexed").
↪setOutputCol(input_col + "_vec")
    pipeline = Pipeline().setStages([indexer, encoder])
    return pipeline, input_col + "_vec"
```

[11]:
```
def clustering_score_3(input_data, k):
    proto_type_pipeline, proto_type_vec_col = one_hot_pipeline("protocol_type")
    service_pipeline, service_vec_col = one_hot_pipeline("service")
    flag_pipeline, flag_vec_col = one_hot_pipeline("flag")

    assemble_cols = set(input_data.columns) - \
                    {"label", "protocol_type", "service", "flag"} | \
                    {proto_type_vec_col, service_vec_col, flag_vec_col}

    assembler = VectorAssembler().setInputCols(list(assemble_cols)).
↪setOutputCol("featureVector")
    scaler = StandardScaler().setInputCol("featureVector").
↪setOutputCol("scaledFeatureVector").setWithStd(True).setWithMean(False)
    kmeans = KMeans().setSeed(randint(100,100000)).setK(k).setMaxIter(40).
↪setTol(1.0e-5).setPredictionCol("cluster").
↪setFeaturesCol("scaledFeatureVector")
    pipeline = Pipeline().setStages([proto_type_pipeline, service_pipeline,␣
↪flag_pipeline, assembler, scaler, kmeans])
    pipeline_model = pipeline.fit(input_data)

    kmeans_model = pipeline_model.stages[-1]
    training_cost = kmeans_model.summary.trainingCost
    return training_cost

for k in list(range(60, 271, 30)):
    print(k, clustering_score_3(data, k))
```

```
60 16406958.580078132
90 6170124.858751168
120 1457907.1809100988
150 1017264.6449497421
180 767171.1009468844
210 570168.2782300282
240 493899.3613133822
270 400056.55190308514
```

0.0.5 Using Labels with Entropy

[12]:
```
from math import log

def entropy(counts):
    values = [c for c in counts if (c > 0)]
    n = sum(values)
    p = [v/n for v in values]
    return sum([-1*(p_v) * log(p_v) for p_v in p])
```

[13]:
```
from pyspark.sql import functions as fun
from pyspark.sql import Window

cluster_label = pipeline_model.\
                    transform(data).\
                    select("cluster", "label")

df = cluster_label.\
        groupBy("cluster", "label").\
        count().orderBy("cluster")

w = Window.partitionBy("cluster")

p_col = df['count'] / fun.sum(df['count']).over(w)
with_p_col = df.withColumn("p_col", p_col)

result = with_p_col.groupBy("cluster").\
            agg((-fun.sum(col("p_col") * fun.log2(col("p_col"))))
                    .alias("entropy"),
            fun.sum(col("count"))
                    .alias("cluster_size"))

result = result.withColumn('weightedClusterEntropy',fun.col('entropy') * fun.
 ↪col('cluster_size'))

weighted_cluster_entropy_avg = result.\
                            agg(fun.sum(
                              col('weightedClusterEntropy'))).\
                            collect()
weighted_cluster_entropy_avg[0][0]/data.count()
```

[13]:
```
1.5576050390165843
```

[14]:
```
def fit_pipeline_4(data, k):
    (proto_type_pipeline, proto_type_vec_col) = one_hot_pipeline("protocol_type")
    (service_pipeline, service_vec_col) = one_hot_pipeline("service")
    (flag_pipeline, flag_vec_col) = one_hot_pipeline("flag")
```

```
    assemble_cols = set(data.columns) - {"label", "protocol_type", "service",␣
↪"flag"} | {proto_type_vec_col, service_vec_col, flag_vec_col}
    assembler = VectorAssembler(inputCols=list(assemble_cols),␣
↪outputCol="featureVector")

    scaler = StandardScaler(inputCol="featureVector",␣
↪outputCol="scaledFeatureVector", withStd=True, withMean=False)

    kmeans = KMeans(seed=randint(100, 100000), k=k, predictionCol="cluster",␣
↪featuresCol="scaledFeatureVector", maxIter=40, tol=1.0e-5)

    pipeline = Pipeline(stages=[proto_type_pipeline, service_pipeline,␣
↪flag_pipeline, assembler, scaler, kmeans])
    return pipeline.fit(data)


def clustering_score_4(input_data, k):
    pipeline_model = fit_pipeline_4(input_data, k)
    cluster_label = pipeline_model.transform(input_data).select("cluster",␣
↪"label")

    df = cluster_label.groupBy("cluster", "label").count().orderBy("cluster")

    w = Window.partitionBy("cluster")

    p_col = df['count'] / fun.sum(df['count']).over(w)
    with_p_col = df.withColumn("p_col", p_col)

    result = with_p_col.groupBy("cluster").agg(-fun.sum(col("p_col") * fun.
↪log2(col("p_col"))).alias("entropy"),
                                               fun.sum(col("count")).
↪alias("cluster_size"))

    result = result.withColumn('weightedClusterEntropy', col('entropy') *␣
↪col('cluster_size'))

    weighted_cluster_entropy_avg = result.agg(fun.
↪sum(col('weightedClusterEntropy'))).collect()
    return weighted_cluster_entropy_avg[0][0] / input_data.count()
```

0.0.6 Clustering in Action

[15]:
```
pipeline_model = fit_pipeline_4(data, 180)
count_by_cluster_label = pipeline_model.transform(data).\
                                        select("cluster", "label").\
                                        groupBy("cluster", "label").\
                                        count().orderBy("cluster", "label")
count_by_cluster_label.show()
```

```
+-------+----------------+------+
|cluster|           label| count|
+-------+----------------+------+
|      0|        neptune.| 35772|
|      0|      portsweep.|     2|
|      1|          smurf.|263762|
|      2|      portsweep.|   603|
|      3|        neptune.|    99|
|      4|        neptune.|    82|
|      5|        neptune.|   101|
|      6|           back.|     2|
|      6|           imap.|     1|
|      6|         normal.|    50|
|      7|        neptune.|    21|
|      8|         normal.|  1073|
|      8|      portsweep.|     2|
|      8|          satan.|     2|
|      8|    warezclient.|     1|
|      9|buffer_overflow.|     3|
|      9|   guess_passwd.|     1|
|      9|        ipsweep.|     1|
|      9|     loadmodule.|     3|
|      9|         normal.|   139|
+-------+----------------+------+
only showing top 20 rows
```