SparkContext and SparkSession

In [1]:
```
import pyspark

sc = pyspark.SparkContext(appName="kmeans")
```

In [2]:
```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
        .master("local").appName("kmeans").getOrCreate()
```

In [3]:
```
dataWithoutHeader = spark.read.option('inferSchema', 'true') \
                              .option('header', 'false') \
                              .csv('../Dropbox/pj_ss/kdd/corrected')
```

In [4]:
```
dataWithoutHeader
```

Out[4]:
```
DataFrame[_c0: int, _c1: string, _c2: string, _c3: string, _c4: int, _c
5: int, _c6: int, _c7: int, _c8: int, _c9: int, _c10: int, _c11: int, _
c12: int, _c13: int, _c14: int, _c15: int, _c16: int, _c17: int, _c18:
int, _c19: int, _c20: int, _c21: int, _c22: int, _c23: int, _c24: doubl
e, _c25: double, _c26: double, _c27: double, _c28: double, _c29: doubl
e, _c30: double, _c31: int, _c32: int, _c33: double, _c34: double, _c3
5: double, _c36: double, _c37: double, _c38: double, _c39: double, _c4
0: double, _c41: string]
```

In [5]:
```
data = dataWithoutHeader.toDF(
"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")
```

In [6]:
```
data
```

Out[6]:
```
DataFrame[duration: int, protocol_type: string, service: string, flag:
string, src_bytes: int, dst_bytes: int, land: int, wrong_fragment: int,
urgent: int, hot: int, num_failed_logins: int, logged_in: int, num_comp
romised: int, root_shell: int, su_attempted: int, num_root: int, num_fi
le_creations: int, num_shells: int, num_access_files: int, num_outbound
_cmds: int, is_host_login: int, is_guest_login: int, count: int, srv_co
unt: int, serror_rate: double, srv_serror_rate: double, rerror_rate: do
uble, srv_rerror_rate: double, same_srv_rate: double, diff_srv_rate: do
uble, srv_diff_host_rate: double, dst_host_count: int, dst_host_srv_cou
nt: int, dst_host_same_srv_rate: double, dst_host_diff_srv_rate: doubl
e, dst_host_same_src_port_rate: double, dst_host_srv_diff_host_rate: do
uble, dst_host_serror_rate: double, dst_host_srv_serror_rate: double, d
st_host_rerror_rate: double, dst_host_srv_rerror_rate: double, label: s
tring]
```

In [7]:
```
data.count()
```

Out[7]:
```
311029
```

In [8]:
```
data.select("label").groupBy("label").count().orderBy("count", ascending
=False).show(25)
```

```
+----------------+------+
|           label| count|
+----------------+------+
|          smurf.|164091|
|         normal.| 60593|
|        neptune.| 58001|
|  snmpgetattack.|  7741|
|       mailbomb.|  5000|
|   guess_passwd.|  4367|
|      snmpguess.|  2406|
|          satan.|  1633|
|    warezmaster.|  1602|
|           back.|  1098|
|          mscan.|  1053|
|        apache2.|   794|
|   processtable.|   759|
|          saint.|   736|
|      portsweep.|   354|
|        ipsweep.|   306|
|     httptunnel.|   158|
|            pod.|    87|
|           nmap.|    84|
|buffer_overflow.|    22|
|       multihop.|    18|
|          named.|    17|
|       sendmail.|    17|
|             ps.|    16|
|          xterm.|    13|
+----------------+------+
only showing top 25 rows
```

KMeans

In [9]:
```
from pyspark.ml import Pipeline
from pyspark.ml.clustering import KMeans, KMeansModel
from pyspark.ml.feature import VectorAssembler
```

In [10]:
```
# numericOnly = data.drop("protocol_type", "service", "flag").dropna().cache()
numericOnly = data.drop("protocol_type", "service", "flag").cache()
```

In [11]:
```
inputCols = numericOnly.columns
inputCols.remove('label')
```

In [12]:
```
assembler = VectorAssembler() \
    .setInputCols(inputCols)\
    .setOutputCol("featureVector")
```

In [13]:
```
kmeans = KMeans() \
    .setPredictionCol("cluster") \
    .setFeaturesCol("featureVector")
```

In [14]:
```
pipeline = Pipeline().setStages([assembler, kmeans])
```

In [15]:
```
pipelineModel = pipeline.fit(numericOnly)
```

In [16]:
```
kmeansModel = pipelineModel.stages[-1]
```

In [17]:
```
kmeansModel.clusterCenters()
```

Out[17]:
```
[array([1.79004395e+01, 1.42797360e+03, 7.47039505e+02, 2.89363946e-05,
        7.61991724e-04, 5.14424793e-05, 1.46771824e-02, 2.36313889e-03,
        1.72476988e-01, 1.12433969e-02, 1.99339607e-04, 2.25060847e-05,
        8.35940288e-03, 9.58116176e-04, 8.35940288e-05, 7.71637189e-04,
        0.00000000e+00, 3.85818595e-05, 2.42422684e-03, 2.69248744e+02,
        2.35581548e+02, 5.92156629e-02, 5.91936070e-02, 1.42586174e-01,
        1.42248551e-01, 8.15652339e-01, 2.44467201e-02, 2.53492141e-02,
        2.35282885e+02, 1.99195186e+02, 7.93499246e-01, 2.49525604e-02,
        5.47922078e-01, 4.56625952e-03, 5.87645767e-02, 5.87915519e-02,
        1.42659769e-01, 1.41694258e-01]),
 array([3.7500000e+02, 4.7235628e+07, 1.4913600e+05, 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, 0.0000000e+00,
        0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 1.0000000e+00,
        1.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
        0.0000000e+00, 1.0000000e+00, 0.0000000e+00, 0.0000000e+00,
        2.0350000e+02, 1.5000000e+00, 1.0000000e-02, 3.5000000e-02,
        5.0000000e-03, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
        0.0000000e+00, 0.0000000e+00])]
```

In [18]:
```
withCluster = pipelineModel.transform(numericOnly)
```

In [19]:
```
withCluster.select("cluster", "label") \
    .groupBy("cluster", "label").count() \
    .orderBy(["cluster", "count"], ascending=[1, 0]) \
    .show(25)
```

```
+-------+----------------+------+
|cluster|           label| count|
+-------+----------------+------+
|      0|          smurf.|164091|
|      0|         normal.| 60593|
|      0|        neptune.| 58001|
|      0|  snmpgetattack.|  7741|
|      0|       mailbomb.|  5000|
|      0|   guess_passwd.|  4367|
|      0|      snmpguess.|  2406|
|      0|          satan.|  1633|
|      0|    warezmaster.|  1602|
|      0|           back.|  1098|
|      0|          mscan.|  1053|
|      0|        apache2.|   794|
|      0|   processtable.|   759|
|      0|          saint.|   736|
|      0|      portsweep.|   354|
|      0|        ipsweep.|   306|
|      0|     httptunnel.|   158|
|      0|            pod.|    87|
|      0|           nmap.|    84|
|      0|buffer_overflow.|    22|
|      0|       multihop.|    18|
|      0|       sendmail.|    17|
|      0|          named.|    17|
|      0|             ps.|    16|
|      0|          xterm.|    13|
+-------+----------------+------+
only showing top 25 rows
```

Choosing k

In [20]:
```
import random
```

In [21]:
```
cols = numericOnly.columns.copy()
cols.remove("label")

assembler = VectorAssembler() \
        .setInputCols(cols) \
        .setOutputCol("featureVector")

kmeans = KMeans() \
        .setSeed(random.randint(0,1000)) \
        .setK(10) \
        .setPredictionCol("cluster") \
        .setFeaturesCol("featureVector")
```

In [22]:
```
pipeline = Pipeline().setStages([assembler, kmeans])
```

In [23]:
```
kmeansModel = pipeline.fit(numericOnly).stages[-1]
```

In [24]:
```
kmeansModel.computeCost(assembler.transform(numericOnly)) / data.count()
```

Out[24]:
```
51921335.40145438
```

In [25]:
```
def clusteringScore0(data, k): # (data: DataFrame, k: Int): Double
    cols = data.columns.copy()
    cols.remove("label")

    assembler = VectorAssembler() \
        .setInputCols(cols) \
        .setOutputCol("featureVector")

    kmeans = KMeans() \
        .setSeed(random.randint(0,1000)) \
        .setK(k) \
        .setPredictionCol("cluster") \
        .setFeaturesCol("featureVector")

    pipeline = Pipeline().setStages([assembler, kmeans])
    kmeansModel = pipeline.fit(data).stages[-1]
    return kmeansModel.computeCost(assembler.transform(data)) / data.cou
nt()
```

In [26]:
```
k=10

kmeans = KMeans() \
        .setSeed(random.randint(0,1000)) \
        .setK(k) \
        .setMaxIter(40) \
        .setTol(1.0e-5) \
        .setPredictionCol("cluster") \
        .setFeaturesCol("featureVector")
```

In [27]:
```
def clusteringScore1(data, k): # (data: DataFrame, k: Int): Double
    cols = data.columns.copy()
    cols.remove("label")

    assembler = VectorAssembler() \
        .setInputCols(cols) \
        .setOutputCol("featureVector")

    kmeans = KMeans() \
        .setSeed(random.randint(0,1000)) \
        .setK(k) \
        .setMaxIter(40) \
        .setTol(1.0e-5) \
        .setPredictionCol("cluster") \
        .setFeaturesCol("featureVector")

    pipeline = Pipeline().setStages([assembler, kmeans])
    kmeansModel = pipeline.fit(data).stages[-1]
    return kmeansModel.computeCost(assembler.transform(data)) / data.cou
nt()
```

In [28]:
```
scores1 = map(lambda x: (x, clusteringScore1(numericOnly, x)) ,range(20,
 101, 20))
```

In [29]:
```
list(scores1)
```

Out[29]:
```
[(20, 9034696.219611958),
 (40, 2756705.918325848),
 (60, 1329020.198590687),
 (80, 926983.917915916),
 (100, 691309.8840558736)]
```

Visualization with R

Feature Normalization

In [37]:
```
from pyspark.ml.feature import StandardScaler
```

In [38]:
```
def clusteringScore2(data, k): #def clusteringScore2(data: DataFrame, k:
 Int): Double = {
    inputCols = data.columns.copy()
    inputCols.remove("label")

    assembler = VectorAssembler() \
        .setInputCols(inputCols) \
        .setOutputCol("featureVector")

    scaler = StandardScaler() \
        .setInputCol("featureVector") \
        .setOutputCol("scaledFeatureVector") \
        .setWithStd(True) \
        .setWithMean(False)

    kmeans = KMeans() \
        .setSeed(42) \
        .setK(k) \
        .setMaxIter(40) \
        .setTol(1.0e-5) \
        .setPredictionCol("cluster") \
        .setFeaturesCol("scaledFeatureVector")

    pipeline = Pipeline().setStages([assembler, scaler, kmeans])
    pipelineModel = pipeline.fit(data)
    kmeansModel = pipelineModel.stages[-1]
    return kmeansModel.computeCost(pipelineModel.transform(data)) / data
.count()
```

In [39]:
```
scores2 = map(lambda x: (x, clusteringScore2(numericOnly, x)) ,range(60,
 271, 30))
```

In [40]:
```
list(scores2)
```

Out[40]:
```
[(60, 1.3694771413258962),
 (90, 0.7050760568516177),
 (120, 0.4869118613988665),
 (150, 0.3841574418069433),
 (180, 0.29687668048640176),
 (210, 0.253320736380156),
 (240, 0.22233164988285115),
 (270, 0.18663344645614047)]
```

Categorical Variables

In [41]:
```
from pyspark.ml.feature import OneHotEncoder, StringIndexer

def oneHotPipeline(inputCol): # (inputCol: String): (Pipeline, String)
    indexer = StringIndexer(inputCol=inputCol, outputCol=inputCol+"_inde
xed")
    encoder = OneHotEncoder(inputCol=inputCol+"_indexed", outputCol=inpu
tCol+"_vec")

    pipeline = Pipeline().setStages([indexer, encoder])
    return (pipeline, inputCol + "_vec")
```

In [42]:
```
def clusteringScore3(data, k): # data: DataFrame, k: Int): Double = {
    (protoTypeEncoder, protoTypeVecCol) = oneHotPipeline("protocol_type"
)
    (serviceEncoder, serviceVecCol) = oneHotPipeline("service")
    (flagEncoder, flagVecCol) = oneHotPipeline("flag")

    inputCols = data.columns.copy()
    for c in ["protocol_type", "service", "flag", "label"]:
        inputCols.remove(c)
    inputCols.extend(["protocol_type_vec", "service_vec", "flag_vec"])

    assembler = VectorAssembler() \
        .setInputCols(inputCols) \
        .setOutputCol("featureVector")

    scaler = StandardScaler() \
        .setInputCol("featureVector") \
        .setOutputCol("scaledFeatureVector") \
        .setWithStd(True) \
        .setWithMean(False)

    kmeans = KMeans() \
        .setSeed(42) \
        .setK(k) \
        .setMaxIter(40) \
        .setTol(1.0e-5) \
        .setPredictionCol("cluster") \
        .setFeaturesCol("scaledFeatureVector")

    pipeline = Pipeline().setStages([protoTypeEncoder, serviceEncoder, f
lagEncoder, assembler, scaler, kmeans])
    pipelineModel = pipeline.fit(data)
    kmeansModel = pipelineModel.stages[-1]
    return kmeansModel.computeCost(pipelineModel.transform(data)) / data
.count()
```

In [43]:
```
scores3 = map(lambda x: (x, clusteringScore3(data, x)), range(60, 271, 30))
list(scores3)
```

Out[43]:
```
[(60, 33.198858240424926),
 (90, 13.799994290846644),
 (120, 3.0079089572099984),
 (150, 1.9903696794018468),
 (180, 1.4556981741961816),
 (210, 1.250157570950619),
 (240, 0.950222243269795),
 (270, 0.7478185031900026)]
```

Using Labels with Entrophy

In [44]:
```
import math
#  Calc entropy
#  파이썬 map은 제너레이터
def calc_each_entropy(v, n):
    p = v/n
    return -p*math.log(p)

def entropy(counts): # (counts: iterable[int]): Double
    values = [x for x in counts if x > 0]
    n = sum(map(float, values))
    entropys = map(lambda v: calc_each_entropy(v, n), values)
    return sum(entropys)
```

In [45]:
```
def fitPipeline4(data, k):
    (protoTypeEncoder, protoTypeVecCol) = oneHotPipeline("protocol_type"
)
    (serviceEncoder, serviceVecCol) = oneHotPipeline("service")
    (flagEncoder, flagVecCol) = oneHotPipeline("flag")

    inputCols = data.columns.copy()
    for c in ["protocol_type", "service", "flag", "label"]:
        inputCols.remove(c)
    inputCols.extend(["protocol_type_vec", "service_vec", "flag_vec"])

    assembler = VectorAssembler() \
        .setInputCols(inputCols) \
        .setOutputCol("featureVector")

    scaler = StandardScaler() \
        .setInputCol("featureVector") \
        .setOutputCol("scaledFeatureVector") \
        .setWithStd(True) \
        .setWithMean(False)

    kmeans = KMeans() \
        .setSeed(42) \
        .setK(k) \
        .setMaxIter(40) \
        .setTol(1.0e-5) \
        .setPredictionCol("cluster") \
        .setFeaturesCol("scaledFeatureVector")

    pipeline = Pipeline().setStages([protoTypeEncoder, serviceEncoder, f
lagEncoder, assembler, scaler, kmeans])
    return pipeline.fit(data)
```

In [46]:
```
from collections import Counter

def clusteringScore4(data, k): # (data: DataFrame, k: Int): Double
    pipelineModel = fitPipeline4(data, k)

    clusterLabel = pipelineModel.transform(data).select(["cluster", "lab
el"])
    labels_grouped = clusterLabel.rdd.groupByKey()
    labels_counted = labels_grouped.map(lambda x: (x[0], len(x[1]), list
(Counter(x[1]).values())))
    weightedClusterEntropy = labels_counted.map(lambda x: x[1]*entropy((
x[2])))

    # Average entropy weighted by cluster size
    return sum(weightedClusterEntropy.collect())/data.count()
```

In [47]:
```
scores4 = map(lambda x: (x, clusteringScore4(data, x)), range(60, 271, 30))
list(scores4)
```

Out[47]:
```
[(60, 0.242510954391682),
 (90, 0.1511254512726149),
 (120, 0.11902392662741602),
 (150, 0.11116496274315389),
 (180, 0.09690936337625397),
 (210, 0.09986496755828855),
 (240, 0.09100749226507922),
 (270, 0.08613404524328735)]
```

Clustering in Action

In [49]:
```
# 결과가 모두 다를거고 여기선 k=180이 최적이라는 가정하에 계산
pipelineModel = fitPipeline4(data, 180)
```

In [50]:
```
countByClusterLabel = pipelineModel.transform(data) \
    .select("cluster", "label") \
    .groupBy("cluster", "label").count() \
    .orderBy(["cluster", "label"])
```

In [66]:
```
countByClusterLabel.show(20)
```

```
+-------+----------+------+
|cluster|     label| count|
+-------+----------+------+
|      0|  ipsweep.|     3|
|      0|    smurf.|131314|
|      1|portsweep.|     4|
|      2|     nmap.|    80|
|      3|  neptune.|    58|
|      3|portsweep.|     1|
|      4|  neptune.|    57|
|      4|portsweep.|     1|
|      5|  neptune.|    40|
|      5|   normal.|     1|
|      6|  neptune.|    55|
|      7|  neptune.|    17|
|      7|    saint.|     1|
|      7|    satan.|     1|
|      8|  neptune.|    56|
|      9|  neptune.|    52|
|      9|    satan.|     1|
|     10|  neptune.|    50|
|     11|   normal.|     1|
|     11|      pod.|     6|
+-------+----------+------+
only showing top 20 rows
```

In [52]:
```
kMeansModel = pipelineModel.stages[-1]
centroids = kMeansModel.clusterCenters()
clustered = pipelineModel.transform(data)
```

In [53]:
```
import numpy as np
def sqdist(a,b):
    return float(np.sqrt(np.sum((a-b)**2, axis=0)))
```

In [60]:
```
#thresholds = clustered.select("cluster", "scaledFeatureVector") \
#    .rdd \
#    .map(lambda x: sqdist(centroids[x[0]], np.array(x[1]))) \
#    .collect()

thresholds = clustered.select("cluster", "scaledFeatureVector").rdd \
    .map(lambda x: sqdist(centroids[x[0]], np.array(x[1]))).take(100)
```

In [61]:
```
threshold = sorted(thresholds)[99]
```

In [62]:
```
threshold
```

Out[62]:
```
10.652130075693822
```

In [63]:
```
samples = clustered.sample(0.01) # 너무 오래걸려서  1%만 샘플링
samples.count()
```

Out[63]:
```
3144
```

In [64]:
```
anomalies = samples.select("cluster", "scaledFeatureVector", "label") \
    .rdd \
    .filter(lambda x: sqdist(centroids[x[0]], np.array(x[1])) >= thresho
ld) \
    .toDF()
```

In [65]:
```
anomalies.select("cluster", "label").groupBy('label') \
    .count().orderBy("count", ascending=False).show()
```

```
+----------------+-----+
|           label|count|
+----------------+-----+
|      portsweep.|    2|
|         normal.|    2|
|   guess_passwd.|    1|
|buffer_overflow.|    1|
+----------------+-----+
```