Identifying Regions for Business Growth: A K-Means Analysis of Business Density and Income

Team GPT

Alexander Nunn, Nicholas Hotelling, Ethan Trantalis, Justin Kopcinski

What is our project?

- Analyze economic activity in the United States by zip code
- Use this to identify areas with high potential for new business development
- Can do this by examining economic density + purchasing power per zip code
- We utilize k-means clusters to analyze the specific data used in order to see where there would be high potential for businesses due to high median incomes+population ratio

What data are we drawing from?

- Dataset from CBP:
  - Lists numbers of business establishments within each zip code in the US
  - Also categorized by a NAICS (North American Industry Classification System) code. Describes specific industries.
- Dataset from ICPSR (Inter-university Consortium for Political and Social Research)
  - Demographic information by zip code
  - Contains total population + median income within each zip code in the US

Preprocessing

- Feature Selection: est_per_capita, MEDFAMINC
- Conversion to .csv files compatible with Spark
- Convert ZCTA region codes to postal codes
- Include state, city & zip Metadata

```
val businessDF = spark.read
    .option("header", "true")
    .csv("../Data/zipcodeBusinessPatterns.txt")
    .withColumn("zip", lpad(col("zip"), 5, "0"))
    .withColumn("est", col("est").cast(DoubleType))
    .filter(col("naics") === targetNaics)

val popIncDF = spark.read
    .option("header", value = true)
    .csv("../Data/zip_population_income_2022.csv")
    .withColumnRenamed("ZCTA20", "zip")
    .withColumn("TOTPOP", col("TOTPOP").cast(DoubleType))
    .withColumn("MEDFAMINC", col("MEDFAMINC").cast(DoubleType))

println("Business rows: " + businessDF.count())
println("Census rows: " + popIncDF.count())

val joinedDF = businessDF
    .join(popIncDF, Seq("zip"))
    .filter(col("TOTPOP") > 0)
```

```
val joinedDF = businessDF
    .join(popIncDF, Seq("zip"))
    .filter(col("TOTPOP") > 0)

val featuresDF = joinedDF
  .withColumn("est_per_capita", col("est") / col("TOTPOP"))
  .select("zip", "stabbr", "cty_name", "est_per_capita", "MEDFAMINC")
    val cleanDF = featuresDF.na.drop(Array("MEDFAMINC", "est_per_capita"))

    println("Number of rows in DataFrame: " + cleanDF.count())
    println("Schema of DataFrame:")
    cleanDF.printSchema()
```

Preprocessing

Join business establishment data with population/median income data base

- Joined on zipcode as key
- Selection ok “------” NAICS code is all industries

EXAMPLE NAICS CODES

---445 - Food and Beverage Retailers

311811 - Retail Bakeries

517122 - Wireless Telecommunications Providers

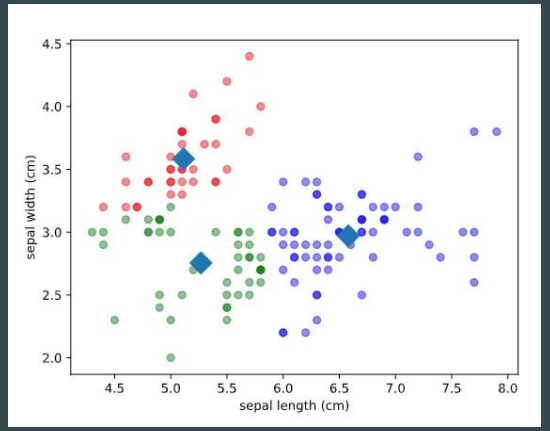
```
val targetNaics = "------" //change this value if specific naics requested
```

```
zip,stabbr,cty_name,est_per_capita,MEDFAMINC
01001,MA,HAMPDEN,0.02923028980990963,101722.0
01002,MA,HAMPSHIRE,0.021877174669450244,138520.0
01005,MA,WORCESTER,0.02212114947281373,111000.0
01007,MA,HAMPSHIRE,0.015291514836022905,123932.0
01008,MA,HAMPDEN,0.017059301380991064,112083.0
01009,MA,HAMPDEN,0.01859799713876967,69544.0
01010,MA,HAMPDEN,0.024330900243309004,129375.0
01011,MA,HAMPDEN,0.006828528072837633,81960.0
01012,MA,HAMPSHIRE,0.031818181818181815,76667.0
01013,MA,HAMPDEN,0.01332168595763267,77471.0
01020,MA,HAMPDEN,0.017555971989348062,80098.0
```

K Means In Python

Kmeans Algorithm

- Select K for initial centroids
- Compute cartesian product for each ingest datum with each centroid
- Find nearest centroid for each datum
- Add datum to centroid group
- Recompute centroid by averaging datums in centroid group
- Repeat until convergence



Kmeans Python

- Selected three k values randomly 3, 5, 8
- Created k centroids derived from dataset
- Computed results

```
totalConverged = 0
maxIterations = 100
totalIterations = 0
while (totalConverged < centroids.shape[0]) and (totalIterations < maxIterations):

    # Reset the cluster arrays and reset the totalConverged for each iteration
    totalConverged = 0
    clusters = [[] for _ in range(centroids.shape[0])]

    for dp in data:
        # Calculate the norm for each row
        distances = np.linalg.norm(dp - centroids, axis=1)

        # Return one answer per row
        closestCentroid = np.argmin(distances)

        # Add to the cluster array
        clusters[closestCentroid].append(dp)


    for i in range(centroids.shape[0]):

        if len(clusters[i]) > 0:

            centroidOld = centroids[i].copy()
            centroidNew = np.mean(clusters[i], axis=0)

            converged = np.allclose(centroidOld, centroidNew, atol=1e-4)

            if converged:
                totalConverged += 1
            else:
                centroids[i] = centroidNew
        else:      You, 4 days ago • added kmeans implementation in python

            # Counting a cluster with no points as converged
            totalConverged += 1

    if totalConverged == len(clusters):
        break
    else:
        totalIterations += 1

for i, c in enumerate(clusters):

    with open(os.path.join(outputDir, f"cluster{i}.csv"), 'w', newline='') as f:

        csvFile = csv.writer(f)
        csvFile.writerows(c)

for i, cent in enumerate(centroids):

    print(f"Cluster {i}: {cent}")
```

Kmeans Control

- Selected three k values randomly 3, 5, 8
- Created k centroids derived from dataset
- Computed results
- Note on feature normalization

```
3 Centroids ------------------------------------------------------------

Initial Centroids:
7435,6,65,0.0415817366490012,141585
66429,33,116,0.0162037037037037,81875
25537,13,322,0.0127260549229738,58385

Final Centroids:
39857,20,542,0.038460,160125
72466,35,1048,0.024348,81454
28356,14,322,0.021105,77077

5 Centroids ------------------------------------------------------------

Initial Centroids:
54123,26,946,0.0384615384615384,47404
85736,43,1715,0.0059949273691491,69332
60410,31,1170,0.0194702934860415,126221
55723,27,1003,0.0529870129870129,81513
46601,23,821,0.100398406374502,55278

Final Centroids:
74367,36,1078,0.022881,68632
69299,34,1000,0.027907,109995
40341,20,570,0.045111,197316
16070,8,149,0.028913,116409
31698,15,369,0.020070,69451

8 Centroids ------------------------------------------------------------

Initial Centroids:
14812,7,131,0.0088253195374315,84940
49921,24,901,0.0165745856353591,85114
61047,31,1169,0.0141451414514145,91406
53527,26,953,0.0230727955642997,130171
95524,46,919,0.0170588235294117,105930
3307,2,20,0.0164952110677545,85893
60549,31,1171,0.0159189580318379,84271
44657,22,800,0.0209667424085932,82753

Final Centroids:
17289,8,171,0.023213,88860
54233,26,692,0.023984,91394
86811,42,1346,0.025988,94148
34288,17,466,0.050454,218528
78095,38,1192,0.034807,144516
16377,8,151,0.033916,137470
75269,37,1096,0.021662,61020
33560,16,401,0.019151,59658
```

K Means In Spark

K Means In Spark

```
val init_pts = sc.textFile(f"../Data/features_std_sample_${sample_size}.csv")
  .map(line => {
    line.split(",")
  })
  .filter(
    {tokens => tokens.head != ""}
  )
  .map(tokens => {
    (
      tokens.head,
      Point(List(tokens(4).toDouble, tokens(5).toDouble))
    )
  })
  .persist() //persisting because points never move
```

K Means In Spark

```
var cent_positions = sc.textFile(s"../Data/features_std_k${k}.csv")
  .map(line => {
    line.split(",")
  })
  .filter(
    {tokens => tokens.head != ""}
  )
  .map(tokens => {
    (
      tokens.head,
      Point(List(tokens(4).toDouble, tokens(5).toDouble))
    )
  })
```

K Means In Spark

```
var cnt_pnts = init_pts.cartesian(cent_positions)
  .map(
    {case ((pid, pnt), (cid, cnt)) => (pid, (cid, DistanceSqr(pnt, cnt)))}
  )
  .reduceByKey({
    case ((cnt1, d1), (cnt2, d2)) => (if (d1 < d2) (cnt1, d1) else (cnt2, d2))
  })
  .map({
    case (pnt, (cnt, _)) => (cnt, pnt)//f"${pnt}%s, ${cnt}%s"
  })
```

K Means In Spark

```
def DistanceSqr(p1: Point, p2: Point): Double = {
  //calculate the squared distance between the two points
  return p1.pos.zip(p2.pos).map({
    case (x1, x2) => math.pow(x1 - x2, 2)
  }).sum
}
```

K Means In Spark

```
cent_positions = cnt_pnts.map({case (c, p) => (p, c)})
  .join(init_pts)
  .map({case (pid, (c, pos)) => (c, (1, pos))}) // 1 is for counting in next step:
  .reduceByKey(
    {case ((count1, p1), (count2, p2)) =>
      (
        count1+count2,
        Point(p1.pos.zip(p2.pos).map(
          {case (a, b) => a+b}
        ))
      )}
  )
  .mapValues(
    {case (count, summed) => Point(summed.pos.map(_/count))}
  )
```

K Means Results - Scala

Initial Centroids:
-0.53953357004927 73,-0.45150387493487515
-0.42495715376313603,-0.6679050048584859
0.09292606366962891,-1.112765050093671

Resultant Centroids (n=50):

```
(0,Point(List(0.232520861857930 42, 1.283896110421008)))
(1,Point(List(-0.1265967593533746, -0.562475190383625)))
(2,Point(List(8.370745743794 58, -0.1391961202203 6634)))
```

K Means Results - Python

Original Centroids

Initial Centroids:
-0.5395335700492773,-0.45150387493487515
-0.4249571537631360 3,-0.6679050048584859
0.09292606366962891,-1.112765050093671

Resultant Centroids (n=50):
0.232521,1.283896
-0.126597,-0.562475
8.370746,-0.139196

Issues

- Data took a while to get (couldn't test algorithms until data was in)
- Scala wasn't efficient enough to run the full dataset (only ran for n=50 to check)