Εθνικο Μετσοβιο Πολυτεχνειο

Επιστημη Δεδομενων και Μηχανικη Μαθηση

Διαχείριση Δεδομένων Μεγάλης Κλίμακας

Αναφορά Εργαστηριακής Άσκησης

Ατζαράκης Κωνσταντίνος	ΑΜ: 03400085	konstantinosatzarakis@mail.ntua.gr
Ορφανουδάκης Φίλιππος Σκόβελεφ	ΑΜ: 03400107	philipposorfanoudakis@mail.ntua.gr

Αθήνα, Ιούλιος 2021

Περιεχόμενα

1. Υπολογισμός Αναλυτικών Ερωτημάτων με τα APIs του Apache Spark

1.1 Μεθοδολογία και Αποτελέσματα

Το σύνολο δεδομένων του πρώτου μέρους προέρχεται από το [1] και αφορά διαδρομές ταξί της Νέας Υόρκης. Εμείς κατά την εκφώνηση χρησιμοποιούμε ένα υποσύνολο του αρχικού dataset [3] για εξοικονόμηση πόρων.

Ζητούμενο 1

Στήσαμε το HDFS σύστημα και φορτώσαμε τα αρχεία

- `yellow_tripdata_1m.csv`
- `yellow_tripvendors_1m.csv`

κατά την εκφώνηση.

Ζητούμενο 2

Μετατρέψαμε τα προηγούμενα αρχεία σε `parquet` format με τις ονομασίες

- `tripdata.parquet`
- `tripvendors.parquet`

μέσω του script `Parquet_a2.py`.

Οι χρόνος που μετρήσαμε είναι :

- Tripdata Read Time: 165.5783770084381 sec
- Tripdata Convert and Write Time: 114.34104323387146 sec
- Tripvendors Read Time: 14.937104940414429 sec
- Tripvendors Convert and Write Time: 9.221333742141724 sec

Ζητούμενο 3

Υλοποιήσαμε τα queries Q1 και Q2 με την χρήση των δύο APIs και για κάθε format εισόδου (`parquet` ή `csv`). Συγκεκριμένα τα αρχεία

- `RDD_1_a3.py`

- `RDD_2_a3.py`

υλοποιούν τα δύο queries με χρήση RDD API, ενώ τα αρχεία

- `SQL_1_a3_csv_parquet.py`
- `SQL_2_a3_csv_parquet.py`

χρησιμοποιούν το Spark SQL με είσοδο `csv` και infer schema == True, όταν δεχθούν σαν argument **csv**.Ενώ έχουν σαν είσοδο `parquet` όταν δεχθούν σαν argument **parquet**.
Στόχος του query Q1 ηταν η εύρεση του μέσου γεωγραφικού πλάτους/μήκους για κάθε ώρα της ημέρας. Η μεθοδολογία που ακολουθήσαμε είναι η εξής:

- Διαβάζουμε το αρχείο μας και μέσω ενός map κρατάμε τις πληροφορίες που μας ενδιαφέρουν, δηλαδή την ώρα που θέτουμε σαν key και το γεωγραφικό μήκος/πλάτος, που μαζί με την τιμή 1 θέτουμε ως value.
- Πραγματοποιούμε έναν καθαρισμό με βάση γεωγραφικά μήκη και πλάτη που επιλέχθηκαν και κρίθηκαν ότι ορίζουν την Νέα Υόρκη. (γεωγραφικό μήκος $\mathbf{in}[-75, -73]$, γεωγραφικό πλάτος $\mathbf{in}[40, 41]$)
- Με ένα reduce για κάθε κλειδί, δηλαδή για κάθε ώρα αθροίζουμε τα γεωγραφικά μήκη, τα γεωγραφικά πλάτη και τους άσσους μεταξύ τους για να έχουμε την συχνότητα εμφάνισης για κάθε ώρα.
- Τέλος με ένα map διαιρούμε το άθροισμα των γεωγραφικών μηκών/πλατών με την συχνότητα για να έχουμε το μέσο όρο.
- Εμφανίζουμε το ζητούμενο

Στόχος του query Q2 είναι η εύρεση της μέγιστης απόστασης και του αντίστοιχου χρόνου για κάθε vendor ταξί. Η μεθοδολογία που ακολουθήσαμε για το query Q2 είναι η εξής:

- Διαβάζουμε το αρχείο μας και μέσω ενός map κρατάμε τις πληροφορίες που μας ενδιαφέρουν, δηλαδή το id που θέτουμε σαν key, την ώρα επιβίβασης/αποβίβασης και τα αντίστοιχα γεωγραφικά μήκη/πλάτη
- Πραγματοποιούμε έναν καθαρισμό με βάση τα γεωγραφικά μήκη και πλάτη που επιλέχθηκαν και κρίθηκαν ότι ορίζουν την Νέα Υόρκη. (γεωγραφικό μήκος $\mathbf{in}[-75, -73]$, γεωγραφικό πλάτος $\mathbf{in}[40, 41]$)
- Με ένα map υπολογίζουμε και θέτουμε σαν έξοδο το id (key), την haversine απόσταση και την διάρκεια της διαδρομής.
- Πραγματοποιούμε ένα join με τον πίνακα που έχει τα id και το αντίστοιχο αναγνωριστικό, για να συνδέσουμε το αναγνωριστικό στην προηγούμενη έξοδο μας και με ένα map θέτουμε σαν key το αναγνωριστικό στη θέση του id.
- Με ενα reduce , για κάθε τιμή του κλειδιού βρίσκουμε την μέγιστη απόσταση και επιστρέφουμε και τον αντίστοιχο χρόνο της διαδρομής.
- Εμφανίζουμε το ζητούμενο

Ζητούμενο 4

Οι χρόνοι των αντίστοιχων ερωτημάτων φαίνονται στο παρακάτω διάγραμμα:

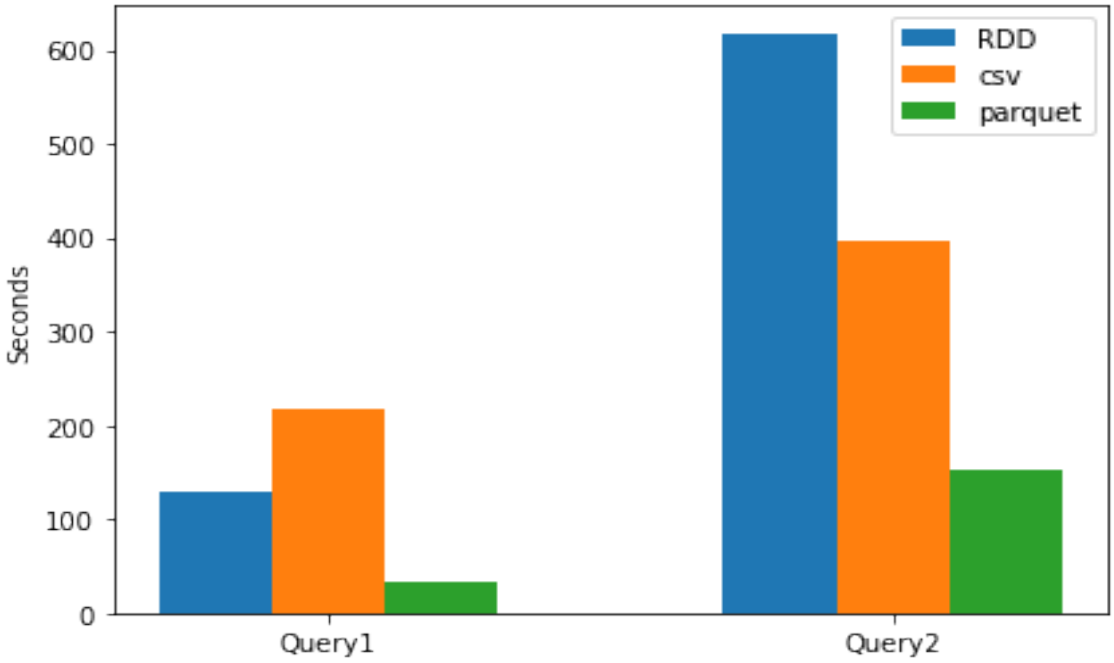

Figure 1.1: Seconds by query and implemantation

Βλέπουμε ότι όταν το πρόγραμμα μας δέχεται ως είσοδο αρχεία σε μορφή parquet τότε επιτυγχάνει πολύ καλύτερους χρόνους. Με το inferschema, γίνεται αυτόματα ο ορισμός του τύπου κάθε στήλης ανάλογα με τα δεδομένα, παρόλα αυτά όταν το αρχείο μας είναι σε parquet μορφή έχει αποθηκευτεί column oriented και όχι row oriented, επομένως δεν έχει νόημα να συμπεράνουμε τον τύπο ολόκληρης της σειράς καθώς δεν ορίζεται. Την πληροφορία αυτή την διαθέτει το parquet ήδη κατα την κατασκευή του.

Ζητούμενο 5

Στον κώδικα που μας δίνεται :

- Συμπληρώνουμε τα κατάλληλα directories
- Ορίζουμε μέσω του script `Optimizer_a5.py` την επιλογή απενεργοποίησης του broadcast join με την εντολή

```
spark.conf.set("spark.sql.autoBroadcastJoinThreshold", -1)
```

Αρχικά αν δεν θέσουμε κάποια επιλογή, θα πραγματοποιηθεί αυτομάτως Broadcast join. Αυτό συμβαίνει γιατί ο πίνακας συνένωσης είναι μικρός (100 γραμμές) και έτσι υπάρχει η δυνατότητα να γίνει broadcast ολόκληρος στη μνήμη κάθε executor. Αντιθέτως αν απενεργοποιήσουμε το broadcast, πραγματοποιείται SortMerge join, κατα τον οποίο τα δεδομένα δέχονται shuffling και διαμοιράζονται στους executors. Στην πρώτη επιλογή, κάθε executor έχει στη μνήμη του ολόκληρο τον μικρό πίνακα και ένα μέρος του μεγάλου πίνακα, ενώ στη δεύτερη επιλογή κάθε executor έχει ένα μέρος του μικρού πίνακα και ένα μέρος του μεγάλου πίνακα. Η επιλογή του διαχωρισμού στη δεύτερη περίπτωση βασίζεται στο κλειδί κατά τα οποίο θα γίνει το join. Η δεύτερη διαδικασία είναι αρκετά πιο χρονοβόρα καθώς απαιτεί ένα αρχικό sorting με βάση την τιμή του κλειδιού. Οι αντίστοιχοι χρόνοι που παρατηρήθηκαν είναι:

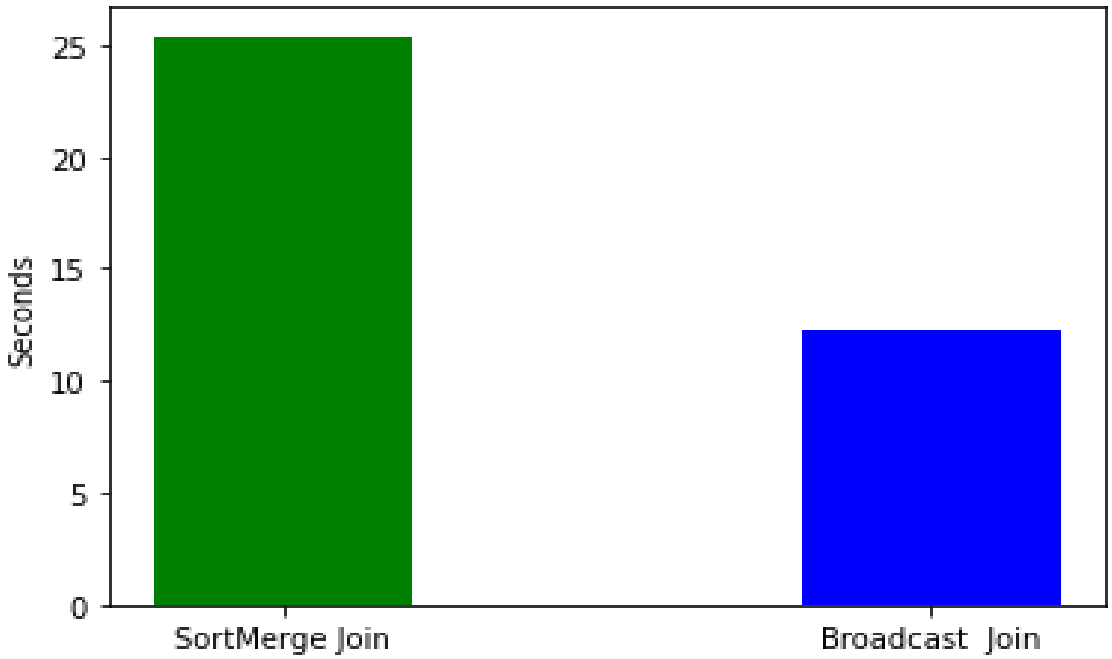

Figure 1.2: Seconds by Join type

1.2 Ψευδοκώδικας

Παραθέτουμε με την μορφή ψευδοκώδικα τις διεργασίες τύπου Map-Reduce που χρησιμοποιήσαμε για τα δύο ερωτήματα. Στον αλγόριθμο 1 φαίνονται τα βήματα που ακολουθήθηκαν για την υλοποίηση του Q1 και στον αλγόριθμο 2 τα βήματα της Q2 αντίστοιχα.

Algorithm 1: Ερώτημα Q1 με χρηση RDD

Input:
trips:= taxi-trip dataset,
format: ("idTrip, time1, time2, lon1, lat1, lon2, lat2")
Output:
mean latitude, longitude of start by hour of the day,
format: (hour, lonMean, latMean)

```
1
2  # apply on trips
3  Map m1(line):
4  |   values = line.split(',')
5  |   hour= values[1].asDateTime().getHour()
6  |   longitude = values[3].asFloat()
7  |   latitude = values[4].asFloat()
8  |   Emit (hour, (longitude, latitude, 1))
9
10 # assert coordinates are in NY
11 Filter f1(−75 ≤ longtitude ≤ −73 & 40 ≤ latitude ≤ 41)
12
13 Reduce  r1(hour, list[(longitude, latitude, 1)]):
14 |   sumlong,sumlat,frequency = (0, 0, 0)
15 |   foreach value in list do
16 |   |   sumlong + = value[0]
17 |   |   sumlat + = value[1] frequency++
18 |   end
19 |   Emit (hour, (sumlong,sumlat,frequency))
20
21 Map m2(hour,(sumlong,sumlat,frequency) ):
22 |   n = frequency
23 |   lonMean = sumlong/n
24 |   latMean = sumlat/n
25 |   Emit (hour, lonMean, latMean)
26
27 Return trips.m1().f1().r1().m2()
```

Algorithm 2: Ερώτημα Q2 με χρηση RDD

Input:
trips:= taxi-trip dataset, format: ("idTrip, time1, time2, lon1, lat1, lon2, lat2")
vendors:= vendors dataset, format: ("idTrip, idVendor")
Output:
max distance & relevant time by vendor, format: (idVendor, distanceMax, time)

```
1
2  Function HaversineDistance(lon1, lat1, lon2, lat2):
3      # convert decimal degrees to radians
4      lon1, lat1, lon2, lat2 = degrees2radians([lon1, lat1, lon2, lat2])
5      # haversine formula
6      dlon = lon2 - lon1
7      dlat = lat2 - lat1
8      a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
9      c = 2 * asin(sqrt(a))
10     r = 6371 # Radius of earth in kilometers
11     Return c * r
12
13 Function maximum(x, y):
14     if x[0]>y[0] then
15         Return (x[0], x[1])
16     else
17         Return (y[0], y[1])
18
19 # applied on trips
20 Map m1(line):
21     values = line.split(',')
22     tripID = values[0].asInt()
23     t1= values[1].asDateTime()
24     t2= values[2].asDateTime()
25     lat1 = values[3].asFloat()
26     lon1 = values[4].asFloat()
27     lat2 = values[5].asFloat()
28     lon2 = values[6].asFloat()
29     Emit (tripID, (t1, t2, lat1, lon1, lat2, lon2))
30
31 # assert coordinates are in NY
32 Filter f1(−75 ≤ both(lat1,lat2) ≤ −73 & 40 ≤ both(lon1, lon2) ≤ 41)
33
34 Map m2(tripID, (t1, t2, lat1, lon1, lat2, lon2)):
35     dist = HaversineDistance(lat1, lon1, lat2, lon2)
36     dt = abs(t1-t2).toSeconds()
37     Emit (tripID, (dist, dt))
38
39 # applied on data
40 Map m3(line):
41     terms = line.split(,)
42     Emit (tripID.asInt(), vendor.asInt())
43
44 Map m4(tripID, ((distance,time),idVendor)):
45     Emit (idVendor, (distance, time))
46
47 Reduce r1(idVendor, values=list[distance,time]):
48     (newmax_distance,new_time) = (distance[0],time[0])
49     foreach value in values do
50         (newmax_distance,new_time) = maximum((newmax_distance,new_time), values)
51     end
52     Emit (key, valueMax)
53
54 trips = trips.m1().f1().m2()
55 data = data.m3().joinByKey(trips, vendors)
56 Return data.m4().r1().sortByKey()
```

2. Machine Learning - Κατηγοριοποίηση κειμένων

2.1 Μεθοδολογία και Αποτελέσματα

Το σύνολο δεδομένων του δεύτερου μέρους προέρχεται από το [2] και αφορά παράπονα πελατών σε σχέση με οικονομικά προϊόντα και υπηρεσίες. Εμείς κατά την εκφώνηση χρησιμοποιούμε ένα υποσύνολο του αρχικού dataset [4] για εξοικονόμηση πόρων.

Ζητούμενο 1

Ανεβάζουμε το αρχείο στο hdfs, σε csv μορφή.

Ζητούμενο 2

Επόμενη κίνηση είναι ο καθαρισμός των δεδομένων μας και τα βήματα που ακολουθήσαμε είναι τα εξής :

- Διαβάζουμε τα δεδομένα μας και κρατάμε μόνο όσες γραμμές ξεκινάνε από '201' και επίσης μπορούν να χωριστούν σε 3 μέρη.
- Στη συνέχεια με ένα map θέτουμε σαν κλειδί το είδος του παραπόνου και εφαρμόζουμε σε κάθε κείμενο της γραμμής μια συνάρτηση καθαρισμού. Η συνάρτηση καθαρισμού περιλαμβάνει την διατήρηση μόνο λέξεων χωρίς σύμβολα, σε πεζά γράμματα, και αφαίρεση ορισμένων stopwords, όπως άρθρα , σύνδεσμοι κτλπ.

Ζητούμενο 3

Στη συνέχεια πρέπει να ορίσουμε ένα λεξικό που θα χρησιμοποιήσουμε για την κωδικοποίηση των κειμένων. Τα βήματα που ακολουθήσαμε για την κατασκευή του λεξικού είναι τα εξής :

- Από κάθε σειρά με την χρήση flatmap δημιουργούμε πολλές σειρές κάθε μια έχει σαν κλειδί μια λέξη του κειμένου και το 1 σαν συχνότητα
- Αθροίζουμε με ένα reduce τις συχνότητες κάθε λέξεις και στη συνέχεια ταξινομούμε με βάση τη συχνότητα. Το λεξικό μας θα αποτελείται από τις συχνότερες K λέξεις. Στη περίπτωση μας δοκιμάσαμε 100, 200 και δυστυχώς αντιμετωπίσαμε θέματα μνήμης για K=700.
- Τέλος το λεξικό αυτό το κάνουμε broadcast σε όλους τους executors, έτσι ώστε να το διαθέτουν στη μνήμη τους.

Ζητούμενο 4

Αφού έχουμε το λεξικό θα προχωρήσουμε στην κωδικοποίηση των κειμένων μας στην εξής μορφή :

$$(K, (ind1, ind2, \ldots, indM), (tfidf1, tfidf2, tfidfM))$$

Τα βήματα που ακολουθήσαμε είναι :

- Αρχικά έχουμε να διαχειριστούμε την μορφή (είδος παραπόνου,κείμενο), όποτε το κείμενο το "σπάμε" σε όλες τις λέξεις του.
- Στη συνέχεια κρατάμε από κάθε σειρά μόνο τις λέξεις που βρίσκονται μέσα στο λεξικό που έχουμε ορίσει και αφαιρούμε τις σειρές που καταλήγουν να είναι κενές.
- Έπειτα προσθέτουμε σε κάθε γραμμή την πληροφορία του πλήθους των λέξεων που έχει, όπως επίσης αναθέτουμε και ένα index σε κάθε γραμμή, έτσι ώστε να γνωρίζουμε το id κάθε κειμένου.
- Έπειτα με ένα flatmap για κάθε λέξη κάθε γραμμής έχω την πληροφορία του σε ποιο κείμενο αντιστοιχεί, ποιο είναι το παράπονο του κειμένου και πόσες λέξεις έχει το κείμενο και προσθέτω και την συχνότητα εμφάνισης του συγκεκριμένου κλειδιού, δηλαδή 1.
- Με έναν reduce για το άθροισμα των συχνοτήτων και με ένα map για την διαίρεση της συχνότητας με το πλήθος, καταλήγουμε η έξοδος μας να είναι η λέξη σαν κλειδί , το index του κειμένου που έχει εντοπιστεί, το παράπονο του κειμένου , η τιμή του tf, και η συχνότητα 1 , δηλαδή το πλήθος των κειμένων που έχει εντοπιστεί αυτή η λέξη.
- Στη συνέχεια με έναν reduce αθροίζουμε τις συχνότητες και με ένα map εφαρμόζουμε τον τύπο του idf για κάθε λέξη.
- Πλέον έχουμε και το tf και το idf για κάθε λέξη συνεπώς με ένα join και με ένα map έχουμε για κάθε λέξη το index του κειμένου που αντιστοιχεί, το παράπονο του κειμένου και το tf-idf της.
- Με ένα map προσθέτουμε στη σειρά την πληροφορία του index στο λεξικό της λέξης και στη συνέχεια με ακόμα ένα map θέτουμε σαν key το index και το παράπονο του κειμένου και σαν value το index της λέξης στο λεξικό και το tf-idf της.
- Συνεπώς με ένα reduce κατασκευάζουμε τη λίστα απο λέξεις και τα αντίστοιχα tf-idf, για κάθε κείμενο. Αφού ταξινομήσουμε τη λίστα μας με βάση το index έχουμε το ζητούμενο.

Παραθέτουμε την έξοδο 5 κειμένων μετά από την υλοποίηση των παραπάνω βημάτων :

('Mortgage', (200, [0, 6, 8, 22, 25, 42, 53, 109], [0.029521113079721474, 0.03850942111413195, 0.1657368283669483, 0.042136223547121175, 0.26594667205047146, 0.04372585227762923, 0.16854983494455614, 0.0613257711943639 1]))

('Debt collection', (200, [0, 1, 2, 14, 17, 18, 32, 33, 49, 50, 54, 71, 86, 88, 128, 130, 142, 162, 186], [0.010218846835288203, 0.04341664291011247, 0.05501341748439607, 0.02424121822047273, 0.026027776254940585, 0.02922563714076144, 0.03429274296923661, 0.06347854229091275, 0.03300924946733428, 0.035643934725845296, 0.03645220198256607, 0.03855307631088083, 0.1307832813541705, 0.03837077063957647, 0.04416947991172635,

0.04515062990399045, 0.04557712883343707, 0.04815339668949177, 0.04848476909549594]))

('Credit card or prepaid card', (200, [0, 1, 4, 5, 10, 11, 12, 13, 14, 15, 30, 31, 35, 39, 43, 44, 47, 63, 64, 72, 89, 97, 102, 111, 118, 127, 135, 138, 147, 161, 163, 166, 178], [0.020437693670576407, 0.07236107151685411, 0.012658643727622789, 0.02136884198383248, 0.023813734179296938, 0.022525779392954262, 0.015485879710263491, 0.012486987676201591, 0.012120609110236364, 0.012246567441357705, 0.04504895097872682, 0.01520776149655636, 0.015811507309651186, 0.018228216178570438, 0.015424626453942013, 0.01634476073745456, 0.016990791668470395, 0.01917065592795509, 0.019399585819093717, 0.018477441927148244, 0.02101112582613904, 0.04286729880862068, 0.02011430152986064, 0.020377292331531183, 0.020896576450026634, 0.04510035671912697, 0.022677911815598584, 0.021454746872450515, 0.022317668439204217, 0.022400927117098037, 0.022395450189123266, 0.024293144269402205, 0.022738109541188628]))

('Checking or savings account', (200, [1, 5, 8, 9, 13, 14, 23, 28, 38, 42, 48, 67, 83, 156, 175, 196], [0.01710352599489279, 0.07576225794267878, 0.03390071489323943, 0.05896658537435559, 0.029514698143749216, 0.028648712442376857, 0.06570876056961197, 0.03745814423404723, 0.08106498827712279, 0.03577569731806028, 0.03945848305073618, 0.04352511900861283, 0.04553925515022631, 0.05527651339981835, 0.05300246391820777, 0.1132386544305202]))

('Credit card', (200, [0, 1, 4, 5, 12, 13, 16, 17, 26, 31, 35, 36, 57, 67, 68, 77, 128, 130, 131], [0.009840371026573825, 0.04180861909862682, 0.02437961014208833, 0.02057740339183868, 0.02982465721976672, 0.024049013302314175, 0.058564263699293635, 0.025063784541794636, 0.029790952970322918, 0.1171560885660638, 0.030451791855624504, 0.030252567683254857, 0.06823061380265079, 0.03546491178479563, 0.042893621382774735, 0.07407728992500229, 0.04253357324832907, 0.04347838435199079, 0.039978307957874]))

Τέλος εφαρμόζουμε SparseVector στη λίστα μας, το μετατρέπουμε σε DataFrame και το αποθηκεύουμε σε μορφή parquet. Το Ζητούμενο 2,3 και 4 υλοποιούνται στο ML_clean_tfidf.py, ενώ ο ψευδοκώδικας παρατίθεται στον αλγόριθμο 3 στο κεφάλαιο 2.2.

Ζητούμενο 5

Τελευταίες τροποποιήσεις στο dataset μας πριν να προχωρήσουμε στην εκπαίδευση του μοντέλου μας είναι οι εξής :

- Μετατρέπουμε το παράπονο του κειμένου δηλαδή το label, σε target index με τη χρήση StringIndexer
- Χωρίζουμε τα δεδομένα μας σε train, test με την χρήση stratified split 70-30, έτσι ώστε να είμαστε σίγουροι ότι έχουμε σε κάθε σετ από όλες τις κλάσεις.

Το train set μας αποτελείται από **343214** γραμμές και το test set από **121998** γραμμές. Ο χωρισμός σε κλάσεις φαίνεται παρακάτω.

label	count
0.0	100447
1.0	74900
2.0	43075
3.0	22543
4.0	22118
5.0	17475
6.0	13390
7.0	12982
8.0	10429
9.0	6541
10.0	5748
11.0	5545
12.0	4500
13.0	1251
14.0	1025
15.0	1023
16.0	210
17.0	12

Figure 2.1: Κατανομή Train Set

label	count
0.0	24828
1.0	27597
2.0	18218
3.0	9159
4.0	7938
5.0	7576
6.0	5603
7.0	5703
8.0	4367
9.0	2875
10.0	2422
11.0	2332
12.0	1922
13.0	486
14.0	469
15.0	421
16.0	79
17.0	3

Figure 2.2: Κατανομή Test Set

Ζητούμενο 6

Τέλος είμαστε έτοιμο να δημιουργήσουμε και να εκπαιδεύσουμε ένα μοντέλο ταξινόμησης. Στα πλαίσια της άσκησης χρησιμοποιούμε το MultilayerPerceptronClassifier της βιβλιοθήκης pyspark και η αρχιτεκτονική στρωμάτων που τελικά μας έφερε το καλύτερο test accuracy αποτελείται από $[200, 100, 50, 18]$ νευρώνες σε κάθε στρώμα και μεγέθους λεξικού $= 200$. Εκπαιδεύουμε για 100 εποχές το μοντέλο μας και το τελικό test accuracy που επιτυγχάνουμε είναι 0.5550500827882424.

Η παραπάνω διαδικασία πραγματοποιείται στο ML_Model.py, στο οποίο δίνεται η δυνατότητα να γίνει cache το train set στους executors, αν δώσουμε σαν είσοδο το "Υ", διαφορετικά με την είσοδο "N" δεν γίνονται cache τα δεδομένα. Οι χρόνοι που μετρήθηκαν για την εκπαίδευση του μοντέλου όπως επίσης και το inference είναι οι εξής :

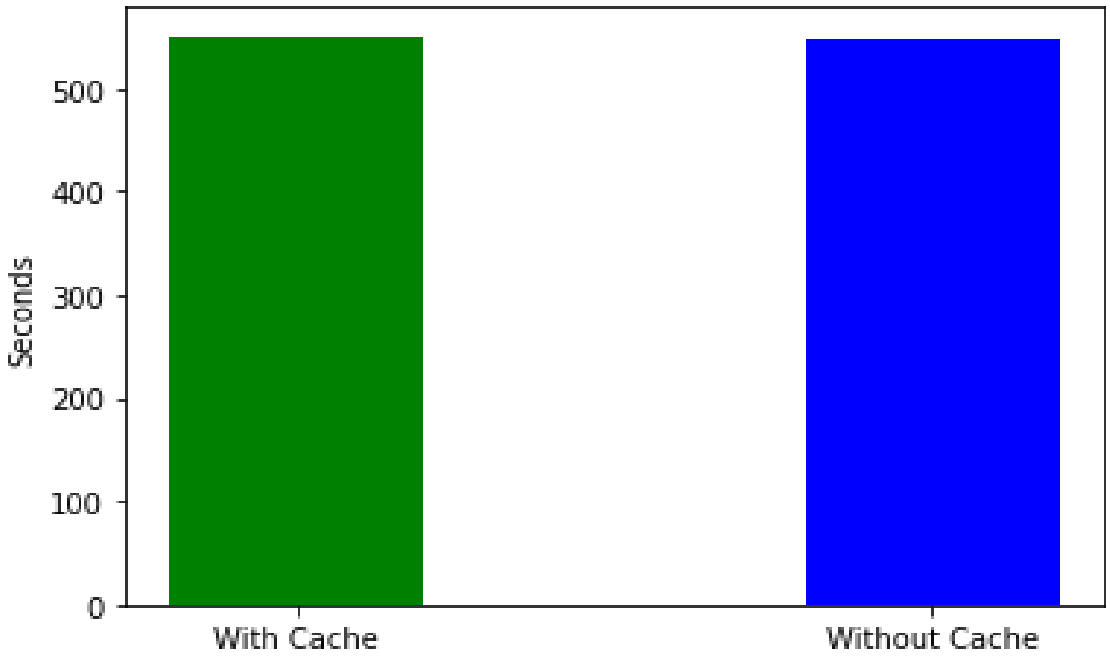

Figure 2.3: Seconds with and without Caching

Όπως βλέπουμε, δεν παρατηρούμε κάποιο πλεονέκτημα με την χρήση του caching. Σε μια προσπάθεια μας να κατανοήσουμε γιατί συμβαίνει αυτό, εξετάσαμε τον optimizer που χρησιμοποιεί ο MultilayerPerceptronClassifier ο οποίος είναι ο L-BFGS. Ο L-BFGS είναι ένας full batch optimizer συνεπώς θα χρησιμοποιήσει όλα τα δεδομένα σε κάθε εποχή. Αυτό θα έχει αποτέλεσμα να τοποθετηθούν στη cache των executors όλα τα δεδομένα από την πρώτη εποχή , επομένως ακόμα και αν δεν ορίσουμε την εντολή cache, εμμέσως πραγματοποιείται.

2.2 Ψευδοκώδικας

Στον αλγόριθμο 3 παρατίθενται οι προκαταρκτικοί καθαρισμοί που έγιναν στα δεδομένα του αρχείου `customer_complaints.csv`. Στον αλγόριθμο 4 περιγράφεται η δημιουργία του λεξικού με τις συχνότερες λέξεις, ενώ τέλος στον αλγόριθμο 5 οι τελικές διεργασίες MapReduce για να παραχθούν τα χαρακτηριστικά TFIDF. Οι παραπάνω αλγόριθμοι υλοποιούνται από το αρχείο `ML_clean_tfidf.py`.

Algorithm 3: Καθαρισμός dataset

```
Input: complaints:= dataset with customer complaints, format: ("date, type, complaint"),
stopWords:= list with stopwords from 'nltk'
Output: cleaned complaints, format: (string_label, complaint)

Function cleanText(text):
    # replace each non-letter character with space
    text.replace(r'[^A-Za-z]+', ' ')
    text.lower()
    cleaned = ' '
    foreach word in text.split() do
        if word not in stopWords then
            cleaned.join(word, delimiter=' ')
        end
    end
    Return (cleaned)

Filter f1( line.startswith('201') & length(line.split(,))=3)

Filter f2( line.split(,)[2] not Empty)

Map m1(line):
    values = line.split(,)
    # ignore date
    Emit (string_label, cleanText(text))

complaints = complaints.f1().f2().m1()
Return complaints
```

Algorithm 4: Υπολογισμός λεξικού k συχνότερων λέξεων

```
Input: complaints:= cleaned dataset with customer complaints, format: (string_label, complaint),
k:= size of dictionary,

FlatMap fm1(string_label, complaint):
    foreach word in complaint.split(' ') do
        Emit (string_label, word)
    end
Map m2(string_label, word):
    Emit (word, 1)

Reduce reduceSum(key, values):
    Emit (key, values.sum()))

Map m3(word, times_in_text):
    Emit word

most_common_words = complaints.fm1().m2().reduceSum().sortBy('value', ascending=False)
broad_com_words = most_common_words..m3().take(k).broadcast()
```

Algorithm 5: Κωδικοποιήση tf-idf

Input:
complaints:= cleaned dataset with customer complaints, format: (string_label, complaint),
k:= size of dictionary
Output: tf-idf SparseVector, format: (K, (ind1, ..., indM), (tfidf1, ..., tfidfM))

```
1
2  Map m4(string_label, complaint):
3  |   words = value.split(' ')
4  |   words_in_dict = [word for word in words if word in broad_com_words]
5  |   Emit (string_label, words_in_dict)
6
7  complaints = complaints.m4().filter(value not Empty)
8  n_texts = complaints.count()
9
10 # compute tf
11 Map m5(string_label, words_in_dict):
12 |   Emit (string_label, words_in_dict, length(words_in_dict))
13
14 FlatMap fm2((string_label, words_in_dict, total_sentence_words), sentence_index):
15 |   foreach word in words_in_dict do
16 |   |   Emit ((word, string_label, sentence_index, total_sentence_words), 1)
17 |   end
18
19 Map m6((word, string_label, sentence_index, total_sentence_words), n_in_sentence):
20 |   tf = n_in_sentence/total_sentence_words
21 |   Emit (word, (string_label, sentence_index, tf, 1))
22
23 tf = complaints.m5().zipWithIndex().fm2().reduceSum().m6()
24
25 # compute idf
26 Map m7(word, (string_label, sentence_index, tf, 1)):
27 |   Emit (word, 1)
28
29 Map idf(word, n_texts_with_word):
30 |   idf = log10(n_texts/n_texts_with_word)
31 |   Emit (word, idf)
32
33 idf = tf.m7().reduceSum().idf()
34 tfidf = join(tf, idf)
35
36 # last pipeline for SparseVector format
37 Map m8(word, ((string_label, sentence_index, tf, 1), idf)):
38 |   word_index_dict = broad_com_words.index(word)
39 |   Emit (( sentence_index, string_label), [(word_index_dict, tf*idf)])
40
41 Map m9(( sentence_index, string_label), index_tfidf_list:
42 |   # sort list of (dictionary_index, tfidf) pairs
43 |   sorted_index_tfidf_list = index_tfidf_list.sortBy(dictionary_index)
44 |   # unzip pairs to 2 different tuples
45 |   index_tuple, tfidf_tuple = unzip(sorted_index_tfidf_list)
46 |   Emit (string_label, (k, index_tuple, tfidf_tuple))
47
48 sparseVector = tfidf.m8().reduceSum().m9()
49 Return sparseVector
```

Βιβλιογραφία

[1] *2015 Yellow Taxi Trip Data*. URL: https://data.cityofnewyork.us/Transportation/2015-Yellow-Taxi-Trip-Data/ba8s-jw6u.

[2] *Consumer Complaint Database*. URL: https://catalog.data.gov/dataset/consumer-complaint-database.

[3] *cslab 2015 Yellow Taxi Trip Data*. URL: http://www.cslab.ntua.gr/courses/atds/yellow_trip_data.zip.

[4] *cslab Consumer Complaint Database*. URL: http://www.cslab.ntua.gr/courses/atds/customer_complains.tar.gz.