ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Δ.Π.Μ.Σ ΕΠΙΣΤΗΜΗ ΔΕΔΟΜΕΝΩΝ & ΜΗΧΑΝΙΚΗ ΜΑΘΗΣΗ

Εξαμηνιαία Εργασία

Διαχείριση Δεδομένων Μεγάλης Κλίμακας

Νικήτας Νικόλαος (03400043)
Ζωγραφάκης Δημήτριος (03400050)

Αθήνα

30/07/2020

Περιεχόμενα

Δομή Εργασίας

Για την εξαμηνιαία εργασία χρησιμοποιήθηκε το Apache Spark και αφορά την εξαγωγή πληροφορίας αλλά και τη χρήση αλγορίθμων μηχανικής μάθησης. Η γλώσσα προγραμματισμού που επιλέχθηκε για την εκπόνηση της εργασίας αυτής είναι η Python.

Το σύστημα αποτελείται από έναν master node και έναν slave node. Και στα 2 nodes έχει εγκατασταθεί το hadoop και το spark, και υπάρχουν συνολικά 2 spark executors που θα εκτελούν τα spark jobs, ένας στο server και ένας στο slave node.

Η ip του master node είναι η 83.212.77.161. Το **hdfs site** όπου έχουν τοποθετηθεί τα datasets της εργασίας είναι στο link http://83.212.77.161:50070/explorer.html#/project.

Οι ολοκληρωμένες εργασίες μπορούν να παρατηρηθούν από την ιστοσελίδα του Spark (http://83.212.77.161:8080/).

Η δομή του παραδεοτέου είναι η ακόλουθη:

```
bigdata_03400043_03400050.zip
|
|_ code
|     |_ part1a
|     |    |_ query1_rdd.py
|     |    |_ query1_sql.py
|     |    |_ query1_sql_parquet.py
|     |    |_ query2_rdd.py
|     |    |_ query2_sql.py
|     |    |_ query2_sql_parquet.py
|     |    |_ convert_to_parquet.py
|     |
|     |_ part1b
|     |    |_ 1b_question1.py
|     |    |_ 1b_question2.py
|     |
|     |_ part2
|          |_ ml.py
|
|_ logs
|     |_ part1a
|     |    |_ *.log.txt
|     |
|     |_ part1b
|     |    |_ *.log.txt
|     |
|     |_ part2
|          |_ *.log.txt
|
|_ results
|     |_ *.png
|
|_ report.pdf
```

Εξαγωγή Πληροφορίας - SQL

Σημειώσεις

Το σύνολο δεδομένων αφορά διαδρομές taxi που πραγματοποιήθηκαν στην Νέα Υόρκη από τον Ιανουάριο του 2015 μέχρι τον Ιούνιο του 2015, και είναι διαθέσιμο εδώ.

Το συγκεκριμένο σύνολο δεδομένων αποτελείται από 2 αρχεία μορφής csv(yellow_tripdata_1m.csv, yellow_tripvendor_1m.csv) τα οποία έχουν συνολικό μέγεθος 2GB.

Τα αναφερόμενα αρχεία εμπεριέχουν τα χαρακτηριστικά όπως ακολουθούν:

Πίνακας 1.1: Αρχείο yellow_tripdata_1m.csv.

Παράμετρος	Τύπος Μεταβλητής	Επεξήγηση
trip_id	Integer	Unique Identifier
pickup_datetime	Timestamp	Starting Time*
dropoff_datetime	Timestamp	Ending Time*
pickup_longitude	Float	Starting Point Longitude
pickup_latitude	Float	Starting Point Latitude
dropoff_longitude	Float	Ending Point Longitude
dropoff_latitude	Float	Ending Point Latitude
fare_amount	Float	Ride's Total Cost

* Υπό της μορφής: (YYYY-MM-DD HH:MM:SS).

Πίνακας 1.2: Αρχείο yellow_tripvendors_1m.csv.

Παράμετρος	Τύπος Μεταβλητής	Επεξήγηση
trip_id	Integer	Unique Identifier
vendor_id	Integer	Unique Identifier

Για όλα τα spark jobs θα εκτελεστεί η εντολή:

```
spark-submit file.py 2>\&1 | tee logs/file.log.txt
```

ώστε να εμφανίζονται τα logs και στη console αλλά και να σώζονται σε ξεχωριστό αρχείο το οποίο θα συμπεριληφθεί στο τελικό παραδοτέο με το κατάλληλο όνομα.

1.1 Εξαγωγή πληροφορίας με διαφορετικούς τρόπους

1.1.1 Φόρτωση Δεδομένων

Αρχικά έγιναν download τα csv αρχεία και φορτώθηκαν στο HDFS μέσω του ακόλουθου κώδικα:

```
wget http://www.cslab.ntua.gr/courses/atds/yellow_trip_data.zip
sudo apt install unzip
unzip yellow_trip_data.zip

hadoop fs -mkdir hdfs://master:9000/project
hadoop fs -put yellow_tripvendors_1m.csv hdfs://master:9000/project
hadoop fs -put yellow_tripdata_1m.csv hdfs://master:9000/project
hadoop fs -ls hdfs://master:9000/project
```

1.1.2 Εκτέλεση ερωτημάτων του Πίνακα 1

a) Ζητήθηκε να εκτελεστούν τα δύο ερωτήματα με MapReduce. Εφόσον ο κώδικας θα τρέξει απευθείας πάνω στα αρχεία κειμένου, αξιοποιήθηκαν το RDDs.

HourOfDay	Longitude	Latitude
00	-73.9754605861983	40.74351721688552
01	-73.97832921516131	40.7415581500586
02	-73.9810569934179	40.741124783390234
03	-73.98198090976972	40.741579197411895
04	-73.9777659488774	40.74491740093727
05	-73.96814540680097	40.74816838848859
06	-73.9698490289864	40.751141454263525
07	-73.97122582122232	40.754124031475975
08	-73.97345441771405	40.754161052832586
09	-73.97476127814892	40.754094686341055
10	-73.97364449974846	40.754515110422794
11	-73.97422556703316	40.75451866881081
12	-73.97460424129109	40.754278165710524
13	-73.97427133563463	40.753677219825704
14	-73.97310950132282	40.75340238755971
15	-73.97134303094197	40.75330651802606
16	-73.96981434209148	40.75291196392322
17	-73.9714843112065	40.75327102002783
18	-73.9738627452453	40.75274772921755
19	-73.97521068092398	40.75137123599664
20	-73.97529798864198	40.74968344805434
21	-73.9748781544584	40.74879079383981
22	-73.97499296443972	40.74770730912061
23	-73.97403341100642	40.74577470148902

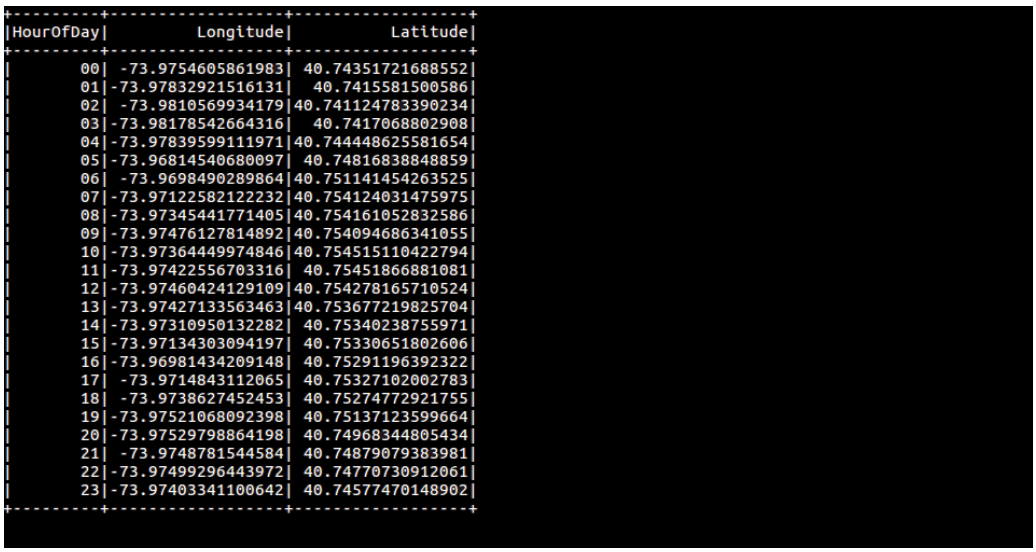
Σχήμα 1.1: Αποτελέσματα του Q1 με RDD.

Vendor	Duration	Distance	TripID
1.0	12.95	1268.09	352187962737
2.0	13.3	922.84	249108442247

Σχήμα 1.2: Αποτελέσματα του Q2 με RDD.

b) Ζητήθηκε εδώ να αξιοποιηθεί η SparkSQL και τα Dataframes. Φορτώθηκαν λοιπόν τα αρχεία σε Dataframes και εκτελέστηκαν SQL queries.

Τα αποτελέσματα της εκτέλεσης του Q1 είναι τα ακόλουθα:



HourOfDay	Longitude	Latitude
00	-73.9754605861983	40.74351721688552
01	-73.97832921516131	40.7415581500586
02	-73.9810569934179	40.741124783390234
03	-73.98178542664316	40.7417068802908
04	-73.97839599111971	40.744448625581654
05	-73.96814540680097	40.74816838848859
06	-73.9698490289864	40.751141454263525
07	-73.97122582122232	40.754124031475975
08	-73.97345441771405	40.754161052832586
09	-73.97476127814892	40.754094686341055
10	-73.97364449974846	40.754515110422794
11	-73.97422556703316	40.75451866881081
12	-73.97460424129109	40.754278165710524
13	-73.97427133563463	40.753677219825704
14	-73.97310950132282	40.75340238755971
15	-73.97134303094197	40.75330651802606
16	-73.96981434209148	40.75291196392322
17	-73.9714843112065	40.75327102002783
18	-73.9738627452453	40.75274772921755
19	-73.97521068092398	40.75137123599664
20	-73.97529798864198	40.74968344805434
21	-73.9748781544584	40.74879079383981
22	-73.97499296443972	40.74770730912061
23	-73.97403341100642	40.74577470148902

Σχήμα 1.3: Αποτελέσματα του Q1 με SQL.

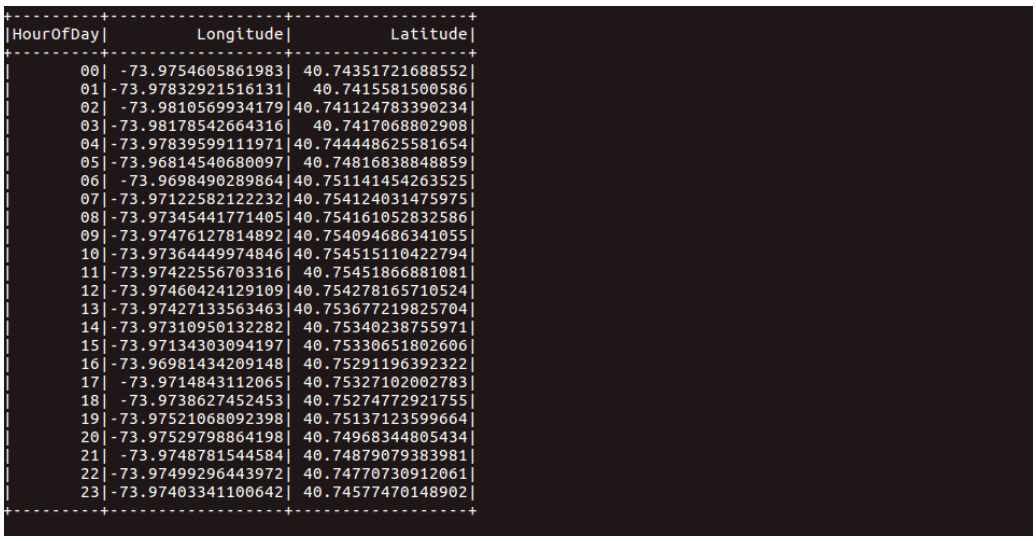
Τα αποτελέσματα της εκτέλεσης του Q2 είναι τα ακόλουθα:

Vendor	Duration	Distance	TripID
1	12.95	1268.09	352187962737
2	13.3	922.84	249108442247

Σχήμα 1.4: Αποτελέσματα του Q2 με SQL.

1.1.3 Μετατροπή αρχείων σε αρχεία Parquet

Σε αυτό το σημείο αξιοποιήθηκε το Apache Parquet, ένα format που αποθηκεύει δυαδικά δεδομένα, το οποίο σε συνδυασμό με το Spark αυξάνει την αποδοτικότητα και βελτιστοποιεί το I/O και χρήση της κοινής μνήμης. Ο χρόνος που χρειάστηκε για την μετατροπή και για τα 2 αρχεία σε Parquet files είναι **4.2 λεπτά** (τρέχοντας το αρχείο convert_to_parquet.py).



HourOfDay	Longitude	Latitude
00	-73.9754605861983	40.74351721688552
01	-73.97832921516131	40.7415581500586
02	-73.9810569934179	40.741124783390234
03	-73.98178542664316	40.7417068802908
04	-73.97839599111971	40.744448625581654
05	-73.96814540680097	40.74816838848859
06	-73.9698490289864	40.751141454263525
07	-73.97122582122232	40.754124031475975
08	-73.97345441771405	40.754161052832586
09	-73.97476127814892	40.754094686341055
10	-73.97364449974846	40.754515110422794
11	-73.97422556703316	40.75451866881081
12	-73.97460424129109	40.754278165710524
13	-73.97427133563463	40.753677219825704
14	-73.97310950132282	40.75340238755971
15	-73.97134303094197	40.75330651802606
16	-73.96981434209148	40.75291196392322
17	-73.9714843112065	40.75327102002783
18	-73.9738627452453	40.75274772921755
19	-73.97521068092398	40.75137123599664
20	-73.97529798864198	40.74968344805434
21	-73.9748781544584	40.74879079383981
22	-73.97499296443972	40.74770730912061
23	-73.97403341100642	40.74577470148902

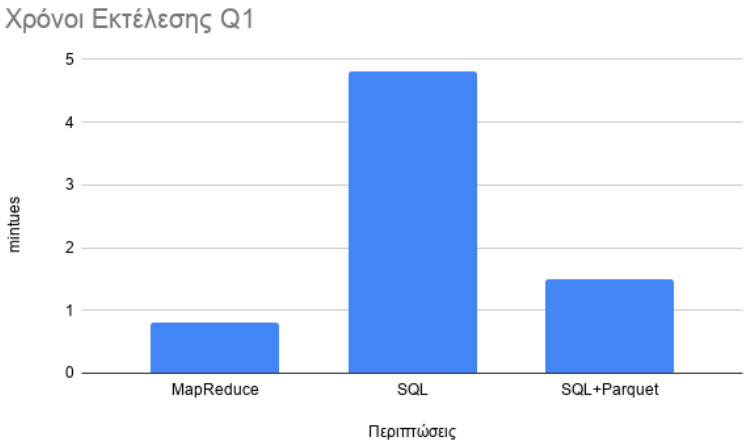
Σχήμα 1.5: Αποτελέσματα του Q1 με SQL σε Parquet.

```
+------+--------+--------+------------+
|Vendor|Duration|Distance|      TripID|
+------+--------+--------+------------+
|     1|   12.95| 1268.09|352187962737|
|     2|    13.3|  922.84|249108442247|
+------+--------+--------+------------+
```

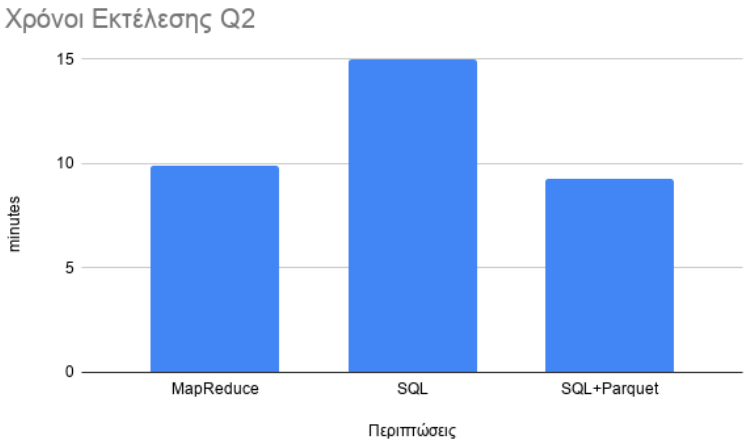
Σχήμα 1.6: Αποτελέσματα του Q2 με SQL σε Parquet.

1.1.4 Σύγκριση Αποτελεσμάτων

Για όλες τις προηγούμενες υλοποιήσεις μετρήθηκαν οι χρόνοι εκτέλεσης τους μέσω του site http://83.212.77.161:8080/, και κατασκευάστηκε τελικά το παρακάτω διάγραμμα:



Σχήμα 1.7: Διάγραμμα χρόνων εκτέλεσης για το Q1.



Σχήμα 1.8: Διάγραμμα χρόνων εκτέλεσης για το Q2.

Παρατηρείται ότι η εκτέλεση με RDDs φαίνεται να είναι πιο γρήγορη και απ'τις 2 περιπτώσεις για το Q1, και παρόμοια σε χρόνο με χρήση Parquet και SQL για το Q2. Η περίπτωση με το SQL πιθανώς αργεί διότι δεν έχει γίνει καλή ρύθμιση του Spark optimizer εξαρχής, και σίγουρα πρέπει να γίνει ούτως ή άλλως μετάφραση σε RDDs μετά την επιλογή του τελικού πλάνου εκτέλεσης. Τέλος, παρατηρείται λοιπόν ότι η χρήση του Apache Parquet αυξάνει την αποδοτικότητα του κώδικα και βελτιστοποιεί την διαδικασία εκτέλεσης του, μειώνοντας έτσι το συνολικό χρόνο εκτέλεσης σε σχέση με την περίπτωση με SQL, λόγω του ειδικού columnar format αποθήκευσης δεδομένων και γρήγορης ανάκτηση πληροφοριών από τα metadata που έχουν σωθεί.

1.2 Μελέτη του βελτιστοποιητή για την συνένωση δεδομένων

1.2.1 Εκτέλεση του υποερωτήματος χωρίς ρύθμιση

Σε αυτό το ερώτημα ζητείται να γίνει JOIN μεταξύ 2 δύο parquet αρχείων με SparkSQL, αφού πρώτα έχουν επιλεχθεί μόνο οι 50 πρώτες εγγραφές από το αρχείο με τις εταιρίες ταξί.

Μέσω της εντολής explain της SQL φαίνεται το παρακάτω φυσικό πλάνο εκτέλεσης:

```
== Physical Plan ==
*(9) SortMergeJoin [_c0#0L], [_c0#16L], Inner
:- *(4) Sort [_c0#0L ASC NULLS FIRST], false, 0
:  +- Exchange hashpartitioning(_c0#0L, 200)
:     +- *(3) BroadcastHashJoin [_c0#0L], [_c0#16L#31L], LeftSemi, BuildRight
:        :- *(3) Project [_c0#0L, _c1#1, _c2#2, _c3#3, _c4#4, _c5#5, _c6#6, _c7#7]
:        :  +- *(3) Filter isnotnull(_c0#0L)
:        :     +- *(3) FileScan parquet [_c0#0L,_c1#1,_c2#2,_c3#3,_c4#4,_c5#5,_c6#6,_c7#7] Batched: true,
 Format: Parquet, Location: InMemoryFileIndex[hdfs://master:9000/user/user/trip_data.parquet], PartitionF
ilters: [], PushedFilters: [IsNotNull(_c0)], ReadSchema: struct<_c0:bigint,_c1:timestamp,_c2:timestamp,_c
3:double,_c4:double,_c5:double,_c6:double,_c7:dou...
:        +- BroadcastExchange HashedRelationBroadcastMode(List(input[0, bigint, true]))
:           +- *(2) Project [_c0#16L AS _c0#16L#31L]
:              +- *(2) GlobalLimit 50
:                 +- Exchange SinglePartition
:                    +- *(1) LocalLimit 50
:                       +- *(1) Project [_c0#16L]
:                          +- *(1) FileScan parquet [_c0#16L] Batched: true, Format: Parquet, Location: I
nMemoryFileIndex[hdfs://master:9000/user/user/trip_vendors.parquet], PartitionFilters: [], PushedFilters:
 [], ReadSchema: struct<_c0:bigint>
+- *(8) Sort [_c0#16L ASC NULLS FIRST], false, 0
   +- Exchange hashpartitioning(_c0#16L, 200)
      +- *(7) BroadcastHashJoin [_c0#16L], [_c0#16L#31L], LeftSemi, BuildRight
         :- *(7) Project [_c0#16L, _c1#17]
         :  +- *(7) Filter isnotnull(_c0#16L)
         :     +- *(7) FileScan parquet [_c0#16L,_c1#17] Batched: true, Format: Parquet, Location: InMemo
ryFileIndex[hdfs://master:9000/user/user/trip_vendors.parquet], PartitionFilters: [], PushedFilters: [IsN
otNull(_c0)], ReadSchema: struct<_c0:bigint,_c1:int>
         +- ReusedExchange [_c0#16L#31L], BroadcastExchange HashedRelationBroadcastMode(List(input[0, big
int, true]))
```

Σχήμα 1.9: Explain του μη optimized SQL query.

Παρατηρείται λοιπόν ότι για την συνένωση των 2 αρχείων, το Spark χρησιμοποίησε το Sort Merge Join (βήμα 9).

Ουσιαστικά, με το Sort Merge Join ακολουθείται η all-to-all communication στρατηγική μεταξύ των κόμβων, μία υλοποίηση που είναι γενικά ακριβή από θέμα χρόνου διότι οι κόμβοι ξοδεύουν χρόνο στο δίκτυο για να μοιράζονται τα δεδομένα.

Η ρύθμιση για την προτίμηση αυτή στο Spark φαίνεται και από την ακόλουθη εντολή μέσα από το διαδραστικό terminal του Spark, η οποία επιστρέφει την τιμή “true” :

```
>>> spark.conf.get("spark.sql.join.preferSortMergeJoin")
'true'
```

1.2.2 Εκτέλεση του υποερωτήματος με ρύθμιση

Για την ρύθμιση του Spark αξιοποιώντας τις ρυθμίσεις του βελτιστοποιητή έγινε η κατάλληλη τροποποίηση αρχικά μέσα στον κώδικα:

```
spark.conf.set("spark.sql.join.preferSortMergeJoin", "false")
```

αλλά το query εκτελέστηκε με Shuffle Join. Τελικά έγινε αλλαγή [1] στο query όπως φαίνεται παρακάτω:

```
"""SELECT /*+ BROADCASTJOIN(trip_vendors) */ trip_data.*, trip_vendors.*
    FROM ...
"""
```

Μέσω της εντολής explain της SQL φαίνεται το παρακάτω φυσικό πλάνο εκτέλεσης:

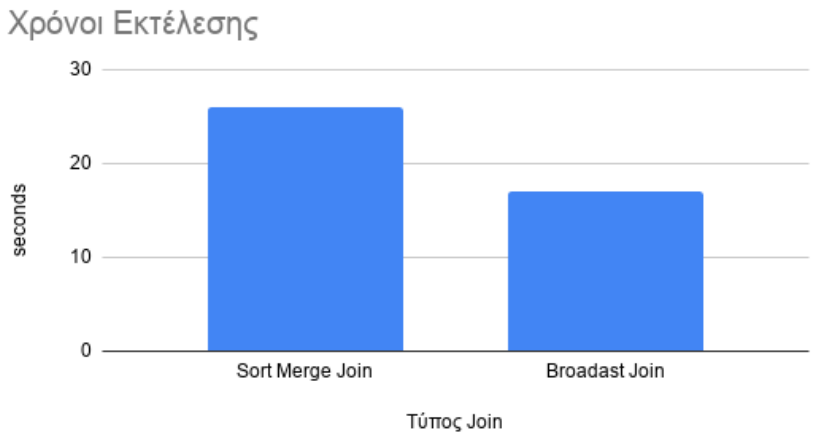
```
== Physical Plan ==
*(6) BroadcastHashJoin [_c0#0L], [_c0#16L], Inner, BuildRight
:- *(6) BroadcastHashJoin [_c0#0L], [_c0#16L#31L], LeftSemi, BuildRight
:  :- *(6) Project [_c0#0L, _c1#1, _c2#2, _c3#3, _c4#4, _c5#5, _c6#6, _c7#7]
:  :  +- *(6) Filter isnotnull(_c0#0L)
:  :     +- *(6) FileScan parquet [_c0#0L,_c1#1,_c2#2,_c3#3,_c4#4,_c5#5,_c6#6,_c7#7] Batched: true, Forma
t: Parquet, Location: InMemoryFileIndex[hdfs://master:9000/user/user/trip_data.parquet], PartitionFilters
: [], PushedFilters: [IsNotNull(_c0)], ReadSchema: struct<_c0:bigint,_c1:timestamp,_c2:timestamp,_c3:doub
le,_c4:double,_c5:double,_c6:double,_c7:dou...
:  +- BroadcastExchange HashedRelationBroadcastMode(List(input[0, bigint, true]))
:     +- *(2) Project [_c0#16L AS _c0#16L#31L]
:        +- *(2) GlobalLimit 50
:           +- Exchange SinglePartition
:              +- *(1) LocalLimit 50
:                 +- *(1) Project [_c0#16L]
:                    +- *(1) FileScan parquet [_c0#16L] Batched: true, Format: Parquet, Location: InMemor
yFileIndex[hdfs://master:9000/user/user/trip_vendors.parquet], PartitionFilters: [], PushedFilters: [], R
eadSchema: struct<_c0:bigint>
+- BroadcastExchange HashedRelationBroadcastMode(List(input[0, bigint, true]))
   +- *(5) BroadcastHashJoin [_c0#16L], [_c0#16L#31L], LeftSemi, BuildRight
      :- *(5) Project [_c0#16L, _c1#17]
      :  +- *(5) Filter isnotnull(_c0#16L)
      :     +- *(5) FileScan parquet [_c0#16L,_c1#17] Batched: true, Format: Parquet, Location: InMemoryF
ileIndex[hdfs://master:9000/user/user/trip_vendors.parquet], PartitionFilters: [], PushedFilters: [IsNotN
ull(_c0)], ReadSchema: struct<_c0:bigint,_c1:int>
      +- ReusedExchange [_c0#16L#31L], BroadcastExchange HashedRelationBroadcastMode(List(input[0, bigint
, true]))
```

Σχήμα 1.10: Explain του optimized SQL query.

Παρατηρείται λοιπόν ότι για την συνένωση των 2 αρχείων το Spark τώρα χρησιμοποίησε το Broadcast Join (βήμα 6).

Ορίστηκε λοιπόν, η συνένωση να πραγματοποιηθεί με brodcast Join. Το Spark θα εκμεταλλευτεί το γεγονός ότι το 2ο αρχείο είναι πολύ μικρό, και θα στείλει ο driver μία αντιγραφή του αρχείου σε κάθε executor για να εργαστεί ξεχωριστά. Έτσι, μειώνεται σημαντικά το overhead της επικοινωνίας μεταξύ του driver και των executors διότι θα υπάρξει επικοινωνία μόνο στην αρχή που θα σταλθεί το αρχείο και στο τέλος αφού γίνουν οι απαραίτητες επεξεργασίες που θα σταλθούν όλα στον driver, αυξάνοντας την απόδοση της εκτέλεσης.

Ο χρόνος εκτέλεσης για το ερώτημα με χρηση Sort Merge Join ήταν 26 seconds, ενώ για το ερώτημα με χρήση Broadcast Join ήταν 17 seconds. Πραγματοποιήθηκε λοιπόν μείωση του χρόνου εκτέλεσης κατά 34.6 %. Πράγματι λοιπόν, για το συγκεκριμένο query ήταν πιο αποδοτική η χρήση του broadcast join από το Spark. Παρακάτω φαίνονται οι ζητούμενοι χρόνοι:



Σχήμα 1.11: Χρόνοι εκτέλεσης με χρήση του βελτιστοποιητή.

Machine Learning - Κατηγοριοποίηση κειμένων

Σημειώσεις

Το σύνολο δεδομένων που θα χρησιμοποιηθεί για αυτό το κομμάτι της εργασίας, είναι το customer_complaints.csv και εμπεριέχει 1.739.625 εγγραφές της μορφής, όπως περιγράφονται παρακάτω (Πίνακας 2.1). Βέβαια, αρκετές από τις εγγραφές δεν είναι καθαρές και θα αφαιρεθούν στα επόμενα στάδια της εργασίας.

Πίνακας 2.1: Αρχείο customer_complaints.csv.

Παράμετρος	Τύπος Μεταβλητής	Επεξήγηση
date	Timestamp	Date of Submission*
category	String	Category of service/product
comment	String	Context

* Υπό της μορφής: (YYYY-MM-DD).

2.1 Φόρτωση Δεδομένων

Αρχικά, γίνεται λήψη του Συνόλου Δεδομένων. Στη συνέχεια λόγω έλλειψης υπολογιστικών πόρων είναι αναγκαίο να ελαττωθούν οι υπολογισμοί. Συνεπώς, ως νέο σύνολο δεδομένων θα θεωρηθούν οι πρώτες 250 χιλιάδες γραμμές. Η ομάδα έχει δοκιμάσει και με το αρχικό σύνολο δεδομένων (λεξιλόγιο 500 και ακρίβεια 74.8%), ωστόσο η περάτωση αργεί αρκετά (πάνω από 90 λεπτά) και δεν είναι δόκιμο για περαιτέρω δοκιμές (δοκιμές για διάφορορετικό μήκος λεξιλογίου, άλλες αρχιτεκτονικές μοντέλου).

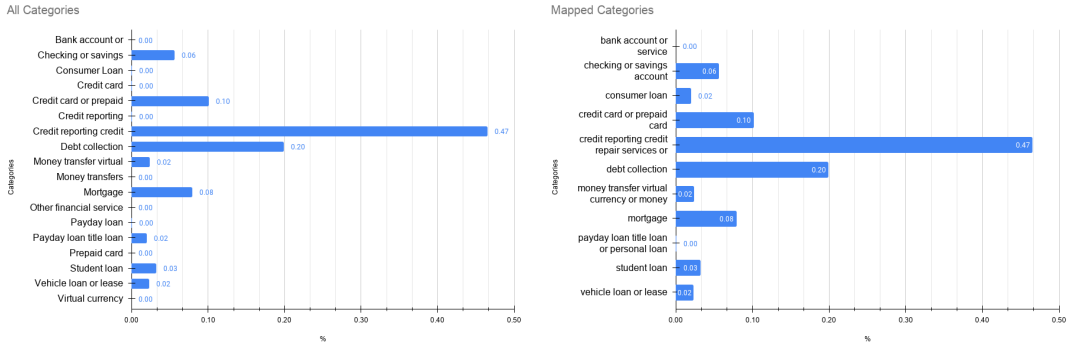
```
wget http://www.cslab.ntua.gr/courses/atds/customer_complains.tar.gz
tar xvzf customer_complains.tar.gz

# Keep 250k lines
head -250000 customer_complaints.csv >> customer_complaints_250k.csv
hadoop fs -put customer_complaints_250k.csv
    hdfs://master:9000/project/customer_complaints_250k.csv
hadoop fs -ls hdfs://master:9000/project
```

2.2 Καθαρισμός Δεδομένων

Σε αυτό το στάδιο θα γίνει προσπάθεια καθαρισμού του συνόλου δεδομένων. Αρχικά, προτού ξεκινήσει οποιαδήποτε μορφή καθαρισμού, είναι δόκιμο να αναγνωριστούν οι κατηγορίες που εμπεριέ-

χονται στο σύνολο δεδομένων καθώς και το πλήθος δειγμάτων για κάθε μία από αυτές. Ένα βασικό φιλτράρισμα, πριν μετρηθούν οι εγγραφές είναι η αφαίρεση κενών εγγραφών, των εγγραφών που δεν αποτελούνται από 3 στήλες και τέλος αφαίρεση εγγραφών όπου η ημερομηνία δεν ξεκινάει από "201". Δημιουργώντας ένα python script (counts_per_category.py), προκειμένου να υλοποιηθεί ένα απλό count και groupby ανά κατηγορία προκύπτει το Σχήμα 2.1.

Πλήθος Καθαρών δειγμάτων: 63.528.



Σχήμα 2.1: Δείγματα ανά κατηγορία (Χωρίς - Με χρήση Mapping) για 250 χιλ. εγγραφές.

Το αριστερό σχήμα παρουσιάζει τα δείγματα ανά κατηγορία στο σύνολο δεδομένων με τις 250 χιλ. εγγραφές. Ως συμπέρασμα προκύπτει ότι αρκετές κατηγορίες δεν έχουν ισάριθμη εκπροσώπηση δειγμάτων μέσα στο σύνολο δεδομένων. Αυτό μπορεί να διορθωθεί μέ δύο τρόπους, είτε αφαιρώντας κατηγορίες, είτε ενώνοντας υπο-κατηγορίες σε μεγαλύτερες. Το δεξιά σχήμα, αποτυπώνει το ίδιο σύνολο δεδομένων όταν έχουν συνενωθεί οι κατηγορίες. **Οι συνενώσεις αυτές προέκυψαν μελετώντας το αρχικό σύνολο δεδομένων (2 εκατ. εγγραφές)** και πραγματοποιήθηκαν όπως φαίνεται στη συνέχεια:

```
mappings = {
    'Bank account or service': 'bank account or service',
    'Checking or savings account': 'checking or savings account',
    'Consumer Loan': 'consumer loan',
    'Credit card': 'credit card or prepaid card',
    'Credit card or prepaid card': 'credit card or prepaid card',
    'Credit reporting': 'credit reporting credit repair services or other personal
        consumer reports',
    'Credit reporting credit repair services or other personal consumer reports': 'credit
        reporting credit repair services or other personal consumer reports',
    'Debt collection': 'debt collection',
    'Money transfers': 'money transfer virtual currency or money service',
    'Mortgage': 'mortgage',
    'Payday loan': 'payday loan title loan or personal loan',
    'Payday loan title loan or personal loan': 'consumer loan',
    'Prepaid card': 'credit card or prepaid card',
    'Student loan': 'student loan',
    'Vehicle loan or lease': 'vehicle loan or lease',
    'Virtual currency': 'money transfer virtual currency or money service',
    'Money transfer virtual currency or money service': 'money transfer virtual currency or
        money service'
}
```

Οι κατηγορίες που θα αφαιρεθούν (αφού δεν έχουν αρκετή εκπροσώπηση στο Σύνολο Δεδομένων με τις 250 χιλ. παρατηρήσεις) είναι: "Other financial service" και "bank account or service". **Σημείωση**: είναι καλό να μελετώνται οι κατηγορίες προς αποχώρηση εφόσον αλλάξει το μέγεθος του συνόλου δεδομένων, αν π.χ αντί για 250χιλ. εγγραφές χρησιμοποιηθούν 500χιλ. . Σχετικά με την κατηγορία "Other financial service", προτείνεται να αφαιρείται πάντα, διότι ακόμα και στο αρχικό σύνολο δεδομένων έχει εκπροσώπηση μόλις 221 δειγμάτων.

Μετά το πέρας των παραπάνω, το σύνολο δεδομένων με 250χιλ. εγγραφές αποτελείται από **11 διαφορετικές κατηγορίες**. Όπως αναφέρθηκε, ένα αρχικό στάδιο καθαρισμού των δεδομένων είναι η αφαίρεση εγγραφών που δεν ανταποκρίνονται σε 3 στήλες (κάνοντας split το κόμμα ","), αφαίρεση εγγραφών με κενές εγγραφές, αφαίρεση εγγραφών που η ημερομηνία δεν ξεκινάει από "201".

Επιπλέον, μετατρέπονται τα κεφαλαία γράμματα κάθε λέξης σε πεζά. Στη συνέχεια, χρησιμοποιείται RegEx ([∧a-z]+|[xx]+), προκειμένου να γίνουν δεκτοί μόνο οι χαρακτήρες του λατινικού αλφαβήτου, εκτός συνεχόμενων x (π.χ XX, XXX κ.α.), καθώς βρέθηκαν κάποιες λέξεις να είναι της μορφής αυτής.

Επιπρόσθετα, γίνεται χρήση της βιβλιοθήκης nltk [2], προκειμένου να εξαχθούν tokens λέξεων από τις προτάσεις [3], να λημματοποιηθούν [4] λέξεις, και τελικά να αφαιρεθούν κοινές αγγλικές λέξεις [5] που δεν προσφέρουν ιδιαίτερο νόημα στα συμφραζόμενα. Τέλος, έγιναν δεκτές μόνο λέξεις που έχουν τουλάχιστον μήκος 2 χαρακτήρων για να έχουν κάποιο νόημα (Ψευδοκώδικας 2.4).

2.3 Υλοποίηση TF-IDF

Η διαδικασία υπολογισμού του TF-IDF, ξεκινάει υπολογίζοντας τις k πιο συχνές λέξεις (vocabulary) μέσα στο σύνολο δεδομένων. Αυτό υπολογίζεται εύκολα, με την χρήση ενός word count (flatMap + reduceByKey), ταξινόμιση κατά φθίνουσα σειρά και τέλος επιλογή των k πρώτων εγγραφών του RDD (take(k)). Η πληθώρα των πειραμάτων τις ομάδας, πραγματοποιήθηκε για μέγεθος λεξικού (k) στο εύρος 500-800 (Ψευδοκώδικας 2.5).

Η διαδικασία συνεχίζει, κάνοντας broadcast το λεξιλόγιο. Έχοντας πρόσβαση στο λεξιλόγιο, κάθε mapper μπορεί να φιλτράρει και να κρατήσει τις λέξεις που ανήκουν μόνο στο λεξιλόγιο. Για κάθε κείμενο, χρησιμοποιείται η συνάρτηση zipWithIndex(), προκειμένου να δοθεί ένας μοναδικός ακέραιος σε κάθε κείμενο. Σε αυτό το στάδιο, έχει δημιουργηθεί ένα RDD που εμπεριέχει όλα τα κείμενα, με μοναδικό κλειδί για το καθένα, και οι λέξεις από κάθε κείμενο έχουν φιλτραριστεί με τέτοιο τρόπο ώστε να ανταποκρίνονται στο λεξικό (Ψευδοκώδικας 2.6).

Κάνοντας count στο παραπάνω RDD, βρίσκεται ο συνολικός αριθμός κειμένων που θα χρειαστεί κατά τον υπολογισμό του IDF. Ο υπολογισμός του IDF, είναι αρκετά απλός. Ουσιαστικά πραγματοποιείται ένα word count με (flatMap + reduceByKey), όπως προηγουμένως με την μόνη διαφορά ότι σε κάθε κείμενο θα γίνουν map οι μοναδικές (unique) λέξεις. Ένα τελικό map, υπολογίζει για κάθε λέξη την ζητούμενη μετρική $(word, log\frac{N}{f_{word}})$. Το αποτέλεσμα του IDF είναι ένας πίνακας δύο διαστάσεων που περιέχει την λέξη και τον συντελεστή IDF. Ο πίνακας αυτός γίνεται broadcoast προκειμένου να είναι ορατός κατά τον υπολογισμό του TF (Ψευδοκώδικας 2.7).

Κατά τον υπολογισμό του TF, γίνεται ένα flatMap για κάθε λέξη σε κάθε κείμενο, με κλειδί (λέξη, κατηγορία, id πρότασης, μήκος κειμένου) και στη συνέχεια reduceByKey για να αθροιστούν. Μετά το πέρας αυτού, έχει δημιουργηθεί ένα RDD το οποίο εμπεριέχει το πλήθος των εμφανίσεων κάθε λέξης μέσα σε κάθε κείμενο. Μετά, πραγματοποιείται ένα map προκειμένου να διαιρεθεί το πλήθος αυτό με το συνολικό μήκος της αντίστοιχης πρότασης και εν τέλει πολλαπλασιάζεται με την μετρική IDF της συγκεκριμένης λέξης, η οποία βρίσκεται εύκολα προσπελαύνοντας την λίστα tf που έγινε broadcasted πριν (Ψευδοκώδικας 2.8).

Στη συνέχεια γίνεται ένα map με σκοπό να βρεθεί το index της λέξης μέσα στο λεξικό. Ένα ακόμα map αναδιοργανώνει την έξοδο κρατώντας μόνο το index του κειμένου μέσα στο Σύνολο Δεδομένων, την κατηγορία, σαν value μία λίστα με το index της λέξης μέσα στο λεξικό, και τη μετρική tf-idf. Τελικά, αθροίζονται σε μία ενιαία λίστα με reduceByKey και με ένα τελευταίο map ταξινομούνται σε αύξουσα σειρά (με βάση το index μέσα στο λεξικό) (Ψευδοκώδικας 2.9).

Σαν τελική έξοδος αυτού του σταδίου είναι η παρακάτω

$$(category, [(id_1 : tfidf_1), \ldots, (id_5 : tfidf_5), \ldots, (id_l : tfidf_l)]) \text{ με } 5 < l \leq k$$

2.4 Ζητούμενη Μετρική Map-Reduce

Τέλος, πραγματοποιείται ένα τελικό map με σκοπό το τελικό RDD να μετασχηματίστει σε Sparse Vector (Ψευδοκώδικας 2.10). Το SparseVector αποτελεί μία "αραιή" έκδοση της προηγούμενης εξόδου,

ιδιαίτερα χρήσιμη αφού εξοικονομεί χώρο στην μνήμη και αμελεί μηδενικές τιμές. Ως αποτελέσμα, το “ταξίδι” των δεδομένων κατά την εκπαίδευση είναι πιο ομαλό και άρα πιο αποδοτικό.

Έτσι, κατά την εκτέλεση του κώδικα λαμβάνονται οι 5 παρακάτω έξοδοι (Σχήμα 2.2)

```
('credit reporting credit repair services or other personal consumer reports', SparseVector(800, {0: 0.06
16, 2: 0.0698, 7: 0.0679, 8: 0.0608, 29: 0.1578, 35: 0.0732, 46: 0.0759, 52: 0.0753, 57: 0.157, 64: 0.080
2, 73: 0.0914, 74: 0.0934, 191: 0.112, 287: 0.123, 312: 0.1297, 314: 0.1379, 490: 0.1462, 625: 0.1598, 65
1: 0.3349}))
('credit reporting credit repair services or other personal consumer reports', SparseVector(800, {0: 0.08
56, 1: 0.1048, 2: 0.1454, 8: 0.2533, 195: 0.4625, 226: 0.4833}))
('credit reporting credit repair services or other personal consumer reports', SparseVector(800, {1: 0.04
19, 2: 0.0581, 4: 0.0829, 26: 0.1214, 29: 0.1315, 68: 0.1496, 83: 0.5047, 97: 0.3127, 126: 0.1667, 131: 0
.1734, 195: 0.185, 414: 0.2502}))
('debt collection', SparseVector(800, {1: 0.0277, 3: 0.1158, 7: 0.1865, 14: 0.0633, 15: 0.0533, 16: 0.018
5, 17: 0.0327, 23: 0.0195, 29: 0.0217, 34: 0.1028, 37: 0.0223, 38: 0.0216, 42: 0.0454, 49: 0.0435, 54: 0.
045, 57: 0.0216, 61: 0.0218, 67: 0.1187, 97: 0.0258, 101: 0.0251, 103: 0.0252, 113: 0.0274, 116: 0.0272,
117: 0.0298, 119: 0.0276, 124: 0.0275, 130: 0.029, 151: 0.0292, 155: 0.0611, 156: 0.0294, 160: 0.0314, 18
2: 0.0343, 191: 0.0308, 203: 0.0305, 210: 0.0314, 232: 0.0652, 239: 0.0722, 247: 0.0365, 254: 0.0363, 259
: 0.07, 333: 0.0362, 367: 0.0363, 370: 0.0382, 415: 0.0379, 522: 0.043, 564: 0.0423, 601: 0.0442, 631: 0.
0436, 679: 0.0461, 718: 0.0459, 743: 0.0501}))
('mortgage', SparseVector(800, {3: 0.0113, 5: 0.0291, 6: 0.0673, 9: 0.0105, 16: 0.0144, 17: 0.0127, 22: 0
.0148, 24: 0.0152, 27: 0.0151, 28: 0.0156, 33: 0.0163, 34: 0.032, 48: 0.0855, 57: 0.0168, 58: 0.0167, 62:
 0.0175, 68: 0.0192, 74: 0.0399, 76: 0.0211, 79: 0.0394, 82: 0.0219, 87: 0.0437, 90: 0.0429, 96: 0.0241,
102: 0.0203, 112: 0.0204, 115: 0.0203, 116: 0.0212, 122: 0.0223, 135: 0.0466, 179: 0.0467, 182: 0.0267, 1
90: 0.0232, 207: 0.0484, 219: 0.0249, 229: 0.026, 232: 0.0507, 234: 0.0495, 236: 0.0248, 238: 0.053, 253:
 0.0604, 260: 0.027, 265: 0.0259, 270: 0.0267, 291: 0.0269, 294: 0.0269, 308: 0.0272, 314: 0.0295, 316: 0
.0281, 360: 0.0295, 370: 0.0297, 379: 0.0577, 382: 0.0575, 384: 0.0328, 386: 0.1149, 390: 0.0574, 431: 0.
03, 442: 0.0307, 461: 0.0311, 496: 0.0327, 523: 0.032, 530: 0.0336, 580: 0.0333, 605: 0.0335, 614: 0.0338
, 675: 0.0742, 679: 0.1076, 726: 0.5258, 734: 0.1944, 736: 0.0358, 760: 0.037, 781: 0.0367}))
```

Σχήμα 2.2: Ζητούμενες Έξοδοι (250 χιλ.).

2.5 Μετατροπή RDD σε Spark Dataframe

Τώρα, χρησιμοποιείται η συνάρτηση toDF() με ορίσματα “category”, “features” προκειμένου να ληφθεί το τελικό Dataframe.

2.6 Προετοιμασία Εκπαίδευσης

Σε αυτό το κομμάτι της εργασίας θα χρησιμοποιηθεί η κλάσση StringIndexer() προκειμένου να κωδικοποιηθεί κάθε κατηγορία σε έναν αύξοντα ακέραιο αριθμό. Έτσι, οι 11 διαφορετικές κατηγορίες (“credit card or prepaid card”, “student loan”, κ.λ.π.) θα αναφέρονται πλέον ως αριθμοί από το 0 εώς το 10.

Οι κατηγορίες είναι 11 λόγω της προηγούμενης ανάλυσης (Ενότητα 2.2). Αρχικά, πραγματοποιείται ένα query πάνω στο Dataframe προκειμένου να βρεθούν και να μετρηθούν (count) όλες οι μοναδικές (distinct) κατηγορίες.

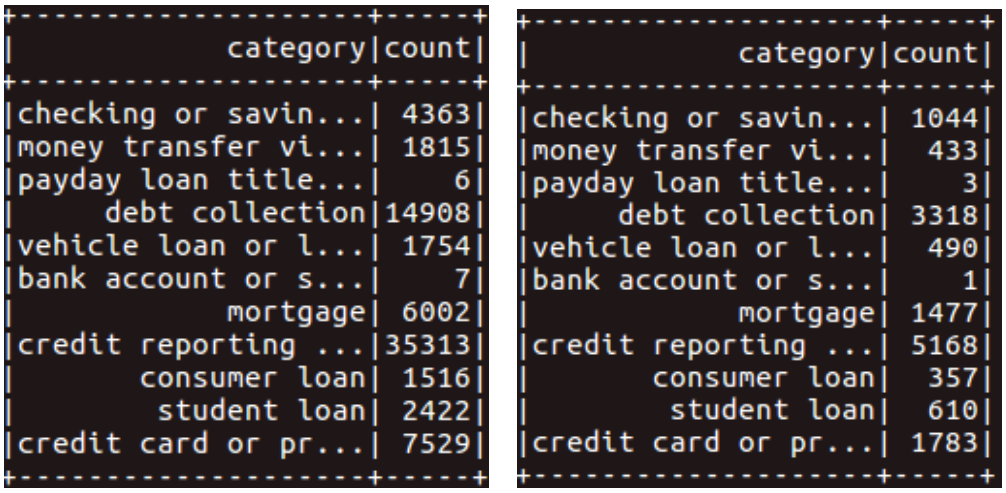
Με βάση το πλήθος των μοναδικών κατηγοριών, δημιουργείται ένα λεξικό που εμπεριέχει τον αριθμό της κατηγορίας και το ποσοστό (%) των δειγμάτων που θα χρησιμοποιηθούν για την εκπαίδευση, εδώ θα γίνει χρήση 80-20, δηλαδή 80% των δειγμάτων για εκπαίδευση και 20% για έλεγχο επίδοσης του αλγορίθμου Μηχανικής Μάθησης.

Η κατασκευή του συνόλου εκπαίδευσης (train set) γίνεται με την συνάρτηση sampleBy() πάνω στο αρχικό Dataframe με την στήλη, την οποία επιθυμείται να χωριστεί (εδώ “label”), το λεξικό που περιγράφτηκε παραπάνω και έναν ακέραιο seed ώστε να δημιουργείται το ίδιο “τυχαίο” μοίρασμα κάθε φορά (σαν seed χρησιμοποιούνται οι 2 τελευταίοι αριθμοί από τα 2 ΑΜ των φοιτητών: 03400**43**, 03400**50**, δηλ. seed=4350). Το σύνολο ελέγχου (test set) δημιουργείται με την συνάρτηση subtract(), δηλαδή ότι απομένει από το αρχικό Dataframe μείον την τομή του με το σύνολο εκπαίδευσης. Η τεχνική που χρησιμοποιείται είναι το stratified split ώστε κάθε set να έχει στοιχεία από κάθε κατηγορία. Παρακάτω παρουσιάζεται το μέγεθος συνόλων εκπαίδευσης και ελέγχου, καθώς και το πλήθος των γραμμών της κάθε κατηγορίας για το κάθε τμήμα.

250 χιλιάδες εγγραφές

Πίνακας 2.2: Μεγέθη των Sets για 250 χιλιάδες εγγραφές

Μέγεθος Train Set	Μέγεθος Test Set
75.635	14.684



category	count
checking or savin...	4363
money transfer vi...	1815
payday loan title...	6
debt collection	14908
vehicle loan or l...	1754
bank account or s...	7
mortgage	6002
credit reporting ...	35313
consumer loan	1516
student loan	2422
credit card or pr...	7529

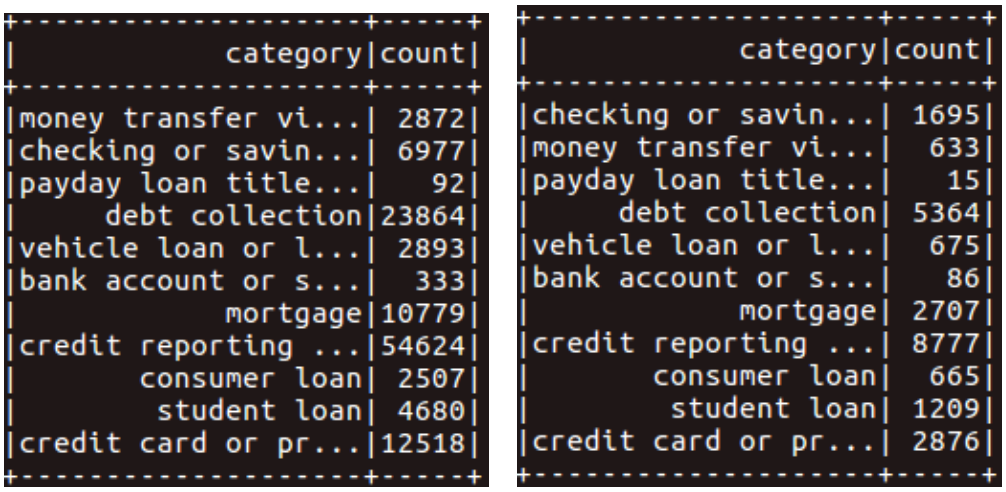
category	count
checking or savin...	1044
money transfer vi...	433
payday loan title...	3
debt collection	3318
vehicle loan or l...	490
bank account or s...	1
mortgage	1477
credit reporting ...	5168
consumer loan	357
student loan	610
credit card or pr...	1783

Σχήμα 2.3: (i) Δείγματα ανά Κατηγορία (Train - Test).

500 χιλιάδες εγγραφές

Πίνακας 2.3: Μεγέθη των Sets για 500 χιλιάδες εγγραφές

Μέγεθος Train Set	Μέγεθος Test Set
122.139	24.802



category	count
money transfer vi...	2872
checking or savin...	6977
payday loan title...	92
debt collection	23864
vehicle loan or l...	2893
bank account or s...	333
mortgage	10779
credit reporting ...	54624
consumer loan	2507
student loan	4680
credit card or pr...	12518

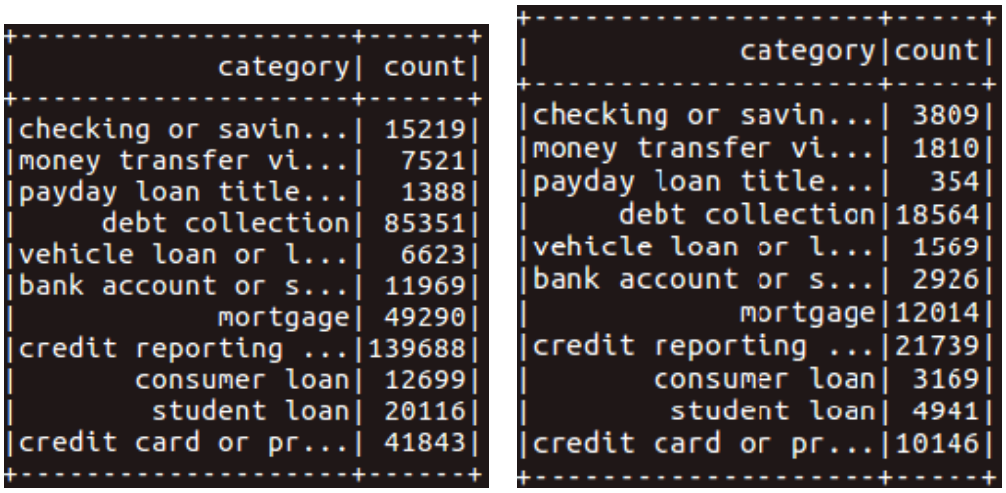
category	count
checking or savin...	1695
money transfer vi...	633
payday loan title...	15
debt collection	5364
vehicle loan or l...	675
bank account or s...	86
mortgage	2707
credit reporting ...	8777
consumer loan	665
student loan	1209
credit card or pr...	2876

Σχήμα 2.4: (ii) Δείγματα ανά Κατηγορία (Train - Test).

1.739.625 εγγραφές

Πίνακας 2.4: Μεγέθη των Sets για όλες τις εγγραφές

Μέγεθος Train Set	Μέγεθος Test Set
391.707	81.146



category	count
checking or savin...	15219
money transfer vi...	7521
payday loan title...	1388
debt collection	85351
vehicle loan or l...	6623
bank account or s...	11969
mortgage	49290
credit reporting ...	139688
consumer loan	12699
student loan	20116
credit card or pr...	41843

category	count
checking or savin...	3809
money transfer vi...	1810
payday loan title...	354
debt collection	18564
vehicle loan or l...	1569
bank account or s...	2926
mortgage	12014
credit reporting ...	21739
consumer loan	3169
student loan	4941
credit card or pr...	10146

Σχήμα 2.5: (iii) Δείγματα ανά Κατηγορία (Train - Test).

2.7 Εκπαίδευση MLP

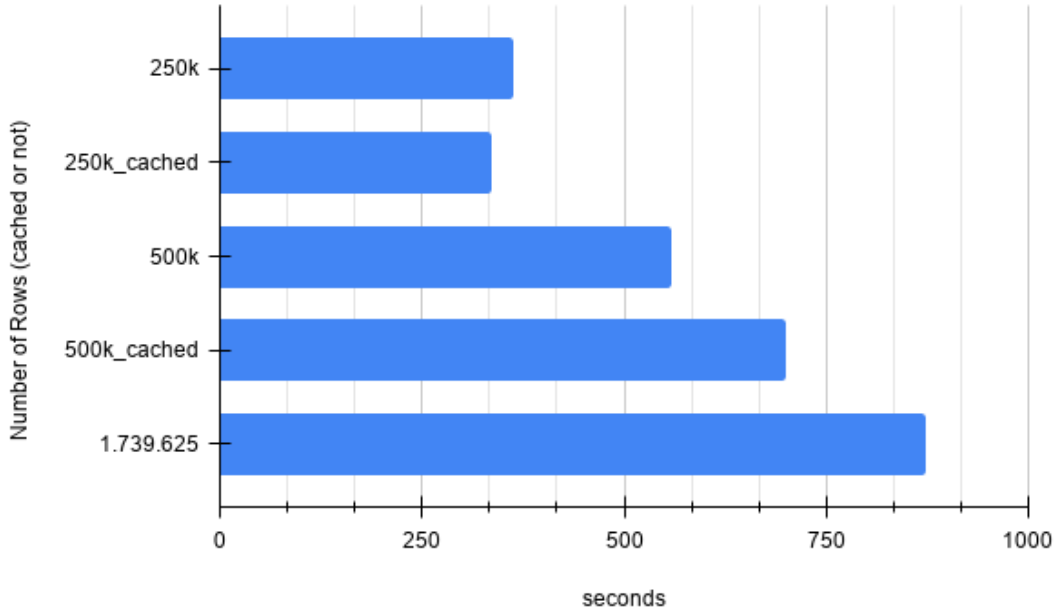
Τα δεδομένα τώρα είναι έτοιμα να εισαχθούν σε έναν ταξινομητή. Για την παρούσα εργασία, θα χρησιμοποιηθεί ένας Multilayer Perceptron Classifer. Δοκιμάστηκαν αρκετές αρχιτεκτονικές, με 2 κρυφά ή και 3 κρυφά επίπεδα, όμως η αρχιτεκτονική που έδινε τα καλύτερα αποτελέσματα για τους υπολογισμούς είναι αρκετά απλή. Ένα επίπεδο μεγέθους όσο το λεξικό (k), ένα "κρυφό" επίπεδο με 200 νευρώνες και ένα τελικό επίπεδο εξόδου ίσο με το πλήθος των διαφορετικών κατηγοριών. Σαν block size θα χρησιμοποιηθεί 64, ενώ σαν seed ο αριθμός που περιγράφηκε παραπάνω.

Για την εξαγωγή του ποσοστού ακρίβειας του μοντέλου πάνω στο Σύνολο Ελέγχου, γίνεται χρήση της κλάσης MulticlassClassificationEvaluator και μετρική την ακρίβεια (accuracy).

Η ακρίβεια του μοντέλου, μαζί με τον χρόνο καθώς και διαφοροποιήσεις όπως διαφορετικά μεγέθη λεξικού, cached (on/off) του συνόλου εκπαίδευσης αποτυπώνονται στον παρακάτων πίνακα (οι χρόνοι έχουν παρθεί από την σελίδα του Spark και αφορούν ολόκληρη την εκτέλεση του προγράμματος).

Πίνακας 2.5: Επιδόσεις Τελικού Μοντέλου.

Ακρίβεια (%)	Λεξικό (k)	Dataset	Cached	Συν. Χρόνος (m)	Χρόνος Εκπ. (s)
76.9	650	250.000	Ναι	16	364
77.2	650	250.000	Όχι	16	337
76.5	500	500.000	Ναι	18	559
77.4	500	500.000	Όχι	21	699
69.9	200	1.739.625	Όχι	41	873



Σχήμα 2.6: Συνολικοί Χρόνοι Περάτωσης Εργασιών.

2.7.1 Συμπεράσματα

Όπως παρατηρείται από τους παραπάνω χρόνους (Σχήμα 2.6) η χρήση του cache στο training set βελτιώνει τους χρόνους εκπαίδευσης. Σε μερικές περιπτώσεις (250 χιλ. εγγραφές), το cached ήταν οριακά πιο αργό από το no cached αλλά αυτό κατα πάσα πιθανότητα οφείλεται στο γεγονός ότι μερικοί workers μπορεί να γεμίσουν από μνήμη και να επανεκκινήσουν σε κάποιο από τα στάδια της εκπαίδεσης και να χρειαστεί η επαναλήψη των σταδίων μέχρι εκείνο το σημείο. Σε γενικότερα πλαίσια (με βάση τις συνολικές δοκιμές της ομάδας), το cached ελάττωνε σημαντικά τον χρόνο εκπαίδευσης του μοντέλου, όπου το επέτρεπε η μνήμη. Εν κατακλείδι, η ομάδα δεν είχε στην διάθεση της αρκετούς υπολογιστικούς πόρους προκειμένου να εκμεταλλευτεί στο έπακρον την δυνατότητα του caching, και για μεγαλύτερους υπολογιστικούς πόρους αναμένεται να βελτιωθεί σημαντικά ο χρόνος της εκπαίδευσης.

Επιπλέον, έγινε χρήση cache και unpersist στα ενδιάμεσα κομμάτια (κατά τον καθαρισμό των δεδομένων και τον υπολογισμό του tf-idf) και παρατηρήθηκε σημαντική βελτίωση στους χρόνους της τάξεως του 50% για το σύνολο δεδομένων με τις 250 χιλιάδες εγγραφές.

Λόγω όμως της φύσης του caching (διατήρηση αντιγράφων στην μνήμη τυχαίας προσπέλασης) πρέπει να λαμβάνεται πάντα υπόψιν τα μεγέθη των αρχείων που παραμένουν φορτωμένα, διότι αν γεμίσει η μνήμη τότε το υλικό και άρα το λογισμικό θα αρχίσει να συμπεριφέρεται απρόσμενα καθιστώντας το έτσι μη αξιόπιστο.

Παράρτημα - Ψευδοκώδικας MapReduce

Ερώτημα 1.Α: Εξαγωγή πληροφορίας με διαφορετικούς τρόπους

Ψευδοκώδικας 2.1: Query 1 (1A).

```
input: tuple (String tripdata)
output: tuple (String hour, (Float avg(lon), Float avg(lat)))

def keepHourFromDate(String date):
    return(String hour)

def map(key, value):
# key: null, value: String tripdata
    hour = keepHourFromDate(date)
    emit(String hour, (Float longitude, Float latitude))
# Filter
def map(key, value):
# key: hour, value: (longitude, latitude)
    if (-80<=longitude<=-60) & (30<=latitude<=50):
        emit(String hour, (Float longitude, Float latitude))

def map(key, value):
# key: hour, value: (longitude, latitude)
    emit(hour, ((longitude, 1), (latitude, 1)))

def reduce(key, values):
# key: hour, values: List((longitude, 1), (latitude, 1))
    emit(hour, ((sum(longitude), count(longitude), (sum(latitude), count(latitude)))))

def map(key, value):
# key: hour, value: ((sum(longitude), count(longitude), (sum(latitude), count(latitude))))
    emit(hour, (avg(lon), avg(lat)))

sortByKey()
```

Ψευδοκώδικας 2.2: Query 2 (1A).

```
input: tuple (String tripdata)
output: tuple (Int vendor_id, (Int trip_id, Float duration, Float haversine)))

def duration(String start_date, String end_date):
    return(duration)

def haversine(Float start_lon, Float start_lat, Float end_lon, Float end_lat):
    return(distance)

def map(key, value):
# key: null, value: String tripdata
```

```
    emit(Int trip_id, (rest_of_tuple))
# Filter
def map(key, value):
# key: trip_id, value: rest_of_tuple
    if (-80<=start_longitude<=-60) & (30<=start_latitude<=50) &
       (-80<=end_longitude<=-60) & (30<=end_latitude<=50):
        emit(trip_id, (rest_of_tuple))

def map(key, value):
# key: trip_id, value: rest_of_tuple
    emit(trip_id, (duration, haversine))

def map(key, value):
# key: null, value: String vendordata
    emit(Int trip_id, (Int vendor_id))

# Join
def map(key, value):
# key: null, value: tuple(join_key trip_id, value v1, value v2, ...)
    emit(join_key trip_id, tagged_tuple(set_name tag, values [v1, v2 ,...]) )
def reduce(key, values):
# key: join_key trip_id, values: tagged_tuples[t1, t2 ,...]
    H = new Array(set_name -> values)
    for all tagged_tuple t in [t1, t2 ,...]:
        H{t.tag}.add(values)
    for all values td in H{trip_data}:
        for all values vd in H{vendor_data}:
            emit(null, (k, td, tv))

def map(key, value):
# key: null, value: (trip_id, td, tv)
    emit(vendor_id, (trip_id, duration, haversine))

def reduce(key, values):
# key: vendor_id, values: List (trip_id, duration, haversine)
    emit(vendor_id, (trip_id, duration, max(haversine)))
```

Ερώτημα 1.Β: Μελέτη του βελτιστοποιητή για την συνένωση δεδομένων

Ψευδοκώδικας 2.3: Query 1 (1B).

```
# Broadcast Join
def init():
    if vendor_data not in local storage:
        remotely retrive vendor_data
        partition vendor_data into p chunks R1, ..., Rp
        save chunks in local storage

    if vendor_data < a split of trip_data:
        # H_R <- H_vendor_data
        H_R <- hash table from R1,...Rp
    else:
        # H_L_i <- H_trip_data_i
        H_L_1, ..., H_L_p <- init p hash tables for trip_data

def map(key, value):
# key: null, value: record from trip_data split
    if H_R exists:
        probe H_R with the join column extracted from value
```

```
            for each match r from H_R:
                emit(null, new_record(r,value))
        else:
            add value to an H_L_i hashing its join column

def close():
    if H_R not exists:
        for each non_empty H_L_i:
            load R_i in memory
            for each record r in R_i:
                probe H_L_i with r join column
                for each match l from H_l_i:
                    emit(null, new_record(r,l))
```

Ερώτημα 2.2: Καθαρισμός Δεδομένων

Ψευδοκώδικας 2.4: Καθαρισμός Δεδομένων.

```
# Data Filtering
input: tuple (date, category, comment)
output: tuple (category, comment)
Define list bad_categories; # Categories to ignore altogether
Define map mappings;

# Filter
def map(tuple):
    if tuple.category not in bad_categories:
        if numberOfColumns(tuple) == 3:
            if startsWith(tuple.date, '201'):
                emit(tuple)

# More cleaning
def clearComment(comment):
    removeNonAlphaChars(comment)
    lemmatize(tokenize(comment)) # comment is now a list
    removeSingleChars(comment)
    listToString(comment)

    return comment

# Map subcategories to wider ones
def map(tuple):
    emit(mappings(tuple.category), clearComment(tuple.comment))

def map(String category, String comment):
    if comment != '':
        emit(category, comment)

cache()
```

Ερώτημα 2.3: Υλοποίηση TF-IDF

Ψευδοκώδικας 2.5: TF-IDF (i).

```
# Most common Words
input: tuple (String words)
output: List<String> of k most frequent words
```

```

def map(String words):
    for each word in split(words, ' '):
        emit(word)

def map(String word):
    emit(word, 1)

def reduce(String word, List<Int> counter):
    emit(word, sum(counter))

def map(String word, Int counter):
    emit(word)

sort()
take(k)
broadcast()
```

Ψευδοκώδικας 2.6: TF-IDF (ii).

```
# Purest Data of them All
input: tuple (String category, String comment)
output: tuple ((String category, List<String> words), Int doc_id)

def map(String category, String comment):
    emit(category, split(comment, ' '))

def map(String category, List<String> words):
    emit(category, isIn(word, vocabulary) for each word in words)

# Filter
def map(String category, List<String> words):
    emit(category, nonEmpty(words))

zipWithIndex()
```

Ψευδοκώδικας 2.7: TF-IDF (iii).

```
# IDF
input: tuple ((String category, List<String> words), Int doc_id)
output: tuple (String word, Float idf)
define N no_of_docs

def map((String category, List<String> words), Int doc_id):
    for each word in unique(words):
        emit(word, 1)

def reduce(String word, List<Int> counter):
    emit(word, sum(counter))

def map(String word, int counter):
    emit(word, log(N/counter))

broadcast()
```

Ψευδοκώδικας 2.8: TF-IDF (iv).

```
# TF-IDF 1
input: tuple ((String category, List<String> words), Int doc_id)
output: tuple ((String word, String category, Int doc_id), Float tfidf)
```

```
4
5  def map((String category, List<String> words), Int doc_id):
6      for each word in words:
7          emit((word, category, doc_id, length(words)), 1)
8
9  def reduce((String word, String category, Int doc_id, Int length), List<Int> counter):
10     emit((word, category, doc_id, length, sum(counter))
11
12 def map((String word, String category, Int doc_id, Int length), Int counter):
13     emit((word, category, doc_id), (counter/length) * idf(word))
```

Ψευδοκώδικας 2.9: TF-IDF (v).

```
1  # TF-IDF 2
2  input: tuple ((String word, String category, Int doc_id), Float tfidf)
3  output: tuple (String category, List<Tuple> (Int vocab_index, Float tfidf))
4
5  def map((String word, String category, Int doc_id), Float tfidf):
6      emit((word, category, doc_id), (index(word, vocabulary), tfidf))
7
8  def map((String word, String category, Int doc_id), (Int vocab_index, Float tfidf)):
9      emit((doc_id, category), List(vocab_index, tfidf))
10
11 def reduce((Int doc_id, String category), List<Tuple> value):
12     emit((doc_id, category), mergeAsList(value))
13
14 sort(key=vocab_index)
```

Ερώτημα 2.4: Ζητούμενη Μετρική Map-Reduce

Ψευδοκώδικας 2.10: TF-IDF (vi).

```
1  input: tuple (category, sorted List<Tuple> (Int vocab_index, Float tfidf))
2  output: tuple (category, sorted SparseVector)
3
4  def map(String category, List<Tuple> words):
5      emit(category, SparseVector(vocab_size, words))
```

Βιβλιογραφία

[1] Spark SQL Guide - Hints
https://spark.apache.org/docs/3.0.0/sql-ref-syntax-qry-select-hints.html

[2] Natural Language Toolkit 3.5.
https://www.nltk.org/data.html

[3] Tokenize Words and Sentences with NLTK.
https://www.guru99.com/tokenize-words-sentences-nltk.html

[4] Python | Lemmatization with NLTK.
https://www.geeksforgeeks.org/python-lemmatization-with-nltk/

[5] NLTK's list of english stopwords.
https://gist.github.com/sebleier/554280