



ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2017-18

M112: Διαχείριση Μεγάλων Δεδομένων

Analyze customer transactions: build a system with kafka and spark to analyze customer transactions. [available @ [github](#)]

Κουκούλη Μυρτώ
Γαλούνη Κωνσταντίνα
Κανακάκης Παναγιώτης

M1503 - mkoukouli@di.uoa.gr
M1524 - kgalouni@di.uoa.gr
M1516 - pkanakakis@di.uoa.gr

ΑΘΗΝΑ

Ιούνιος 2018

Σκοπός εργασίας	3
Αρχιτεκτονική εφαρμογής	4
Kafka Web Console	5
Kafka Connect UI	6
Περιγραφή δεδομένων και Στατιστικά	9
Περιγραφή συστήματος συστάσεων	13
Προεπεξεργασία δεδομένων	13
Alternating Least Squares (ALS) Αλγόριθμος	14
Cross Validation	14
Μετρική percentile ranking	15
Χρήσιμη βιβλιογραφία και ιστοσελίδες	16
Για collaborative filtering μέσω spark	16
Για sparkR	16
Για Kafka	16

1. Σκοπός εργασίας

Η εργασία εκπονείται στο πλαίσιο του μαθήματος “Διαχείριση Μεγάλων Δεδομένων”. Τα δεδομένα μας οριακά (δεδομένου ότι υπάρχει περιορισμός στους διαθέσιμους πόρους) μπορούν να θεωρηθούν αρκετά μεγάλα, ώστε να εκμεταλλευτούμε τα εργαλεία και τις τεχνικές που είναι διαθέσιμες σχετικά με εκμετάλλευση πολλαπλών πυρήνων και μηχανημάτων ως clusters, για ταχύτερη διαχείρισή τους.

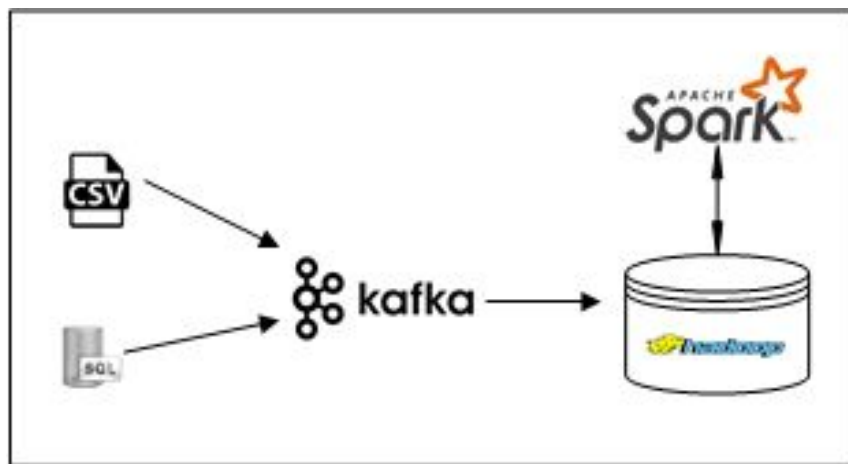
Για την επεξεργασία των δεδομένων θα ακολουθήσουμε την εξής σειρά:

- Ανάλυση δεδομένων: Εκτέλεση επερωτημάτων, εξαγωγή στατιστικών αποτελεσμάτων και οπτικοποίηση μέρους αυτών χρησιμοποιώντας το Spark (SparkSQL, SparkR).
- Χρήση της MLlib βιβλιοθήκης του Spark για κατηγοριοποίηση των δεδομένων. Βασικός στόχος είναι η εύρεση μιας μεθόδου πρόβλεψης των μελλοντικών ενεργειών των χρήστη μέσω ανάλυσης των χαρακτηριστικών των προϊόντων για τα οποία έδειξε ενδιαφέρον.

Για την εξοικείωσή μας με περισσότερα από τα διαθέσιμα εργαλεία διαχείρισης Μεγάλων Δεδομένων, παρόλο που τα δεδομένα δεν δίνονται σε συνεχή ροή (streaming), αλλά είναι στατικά αποθηκευμένα σε αρχεία, στοχεύουμε στη χρήση εργαλείων όπως το Apache Kafka, Apache Hadoop και Apache Spark.

2. Αρχιτεκτονική εφαρμογής






Η εφαρμογή δέχεται δεδομένα από μία βάση δεδομένων (sqlite) καθώς και από αρχεία csv. Προκειμένου το σύστημα να δρα ανεξάρτητα από την πηγή δεδομένων αλλά και να είναι εύκολη η προσθήκη νέων πηγών χρησιμοποιήθηκε το Apache Kafka. Η χρήση του επιτρέπει την ομογενοποίηση των δεδομένων αλλά και τον καλύτερο διαχωρισμό του συστήματος σε τμήματα. Η επεξεργασία των δεδομένων έγινε μέσω της πλατφόρμας Apache Spark, το οποίο λάμβανε τα δεδομένα από hdfs σύστημα. Συνοπτικά, η αρχιτεκτονική του συστήματος παρουσιάζεται στην παρακάτω εικόνα.















Για την ευκολότερη διαχείριση και χρήση του συστήματος σε πολλαπλά μηχανήματα επιλέχθηκε η ανάπτυξη του κύριου μέρους του σε περιβάλλον Docker. Έτσι, η βάση δεδομένων, το Kafka αλλά και το Hadoop βρίσκονται σε ένα κοινό docker compose αρχείο. Ακόμη, κρίθηκε απαραίτητη η χρήση των εργαλείων της πλατφόρμας Confluent για την οπτικοποίηση της διαδικασίας αλλά και για να καταστεί δυνατή η επικοινωνία των επιμέρους τμημάτων.

2.1 Kafka Web Console

Για την καλύτερη οπτικοποίηση και διαχείριση των περιεχομένων του Kafka χρησιμοποιήθηκε μία web εφαρμογή, η οποία παρέχει πληροφορίες σχετικά με τα topics, τους producers/consumers αλλά και τους κόμβους zookeeper. Η χρήση της είναι ιδιαίτερα απλή μιας και είναι αρκετή η παραμετροποίηση ορισμένων μεταβλητών όπως είναι το url που τρέχει ο broker του Kafka. Ακόμη, για την ορθή απεικόνιση των πληροφοριών απαιτείται η δημιουργία ενός zookeeper node όπως φαίνεται στην παρακάτω εικόνα.

Kafka Web Console						
 Zookeepers	All Development Production Staging Test Register Zookeeper					
	Name	Host	Port	Status	Group	Chroot
	zk	astero.dl.uoa.gr	2181	CONNECTED	ALL	
 Brokers						
 Topics						
 Settings						

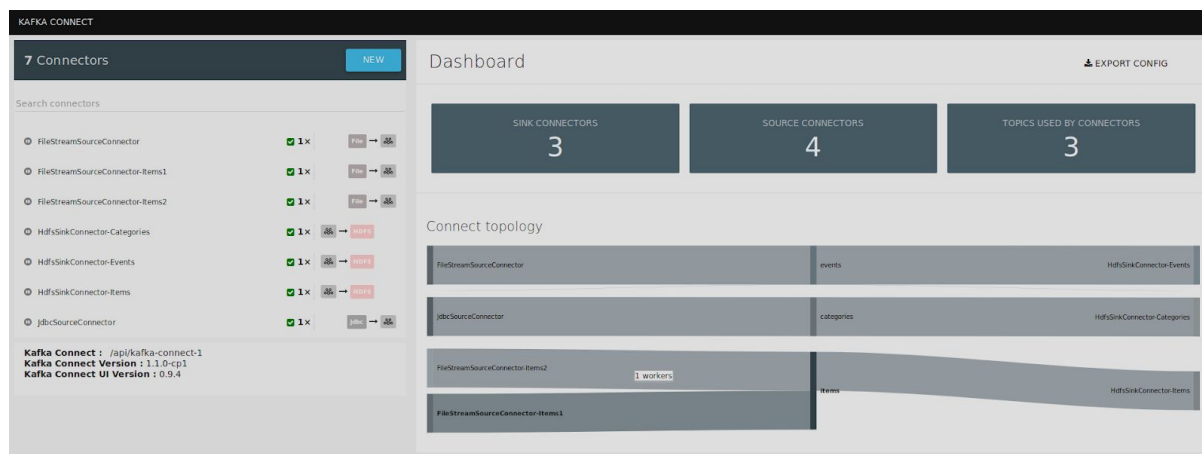
Τελικά, η συνολική εικόνα του Kafka για την εξεταζόμενη εφαρμογή μετά την εισαγωγή δεδομένων είναι η ακόλουθη.

Kafka Web Console					
 Zookeepers					
		Zookeeper	Topic	Partition	Log Size Leader
 Brokers		zk	docker-connect-status	-	940 -
		zk	_schemas	-	2 -
		zk	Items	-	158000 -
		zk	__consumer__offsets	-	333 -
 Topics		zk	docker-connect-offsets	-	87 -
		zk	docker-connect-configs	-	128 -
		zk	categories	-	0 -
 Settings		zk	events	-	150000 -

2.2 Kafka Connect UI

Για την ευκολότερη διαχείριση των δεδομένων εισόδου / εξόδου έγινε χρήση της πλατφόρμας Confluent. Η πλατφόρμα περιέχει ένα σύνολο υπηρεσιών και εργαλείων προκειμένου να είναι ευκολότερο το data integration μεταξύ των διάφορων συστημάτων της εφαρμογής. Ειδικότερα, χρησιμοποιήθηκε το Kafka Connect, το οποίο επιτρέπει την δημιουργία Source Connectors και Sink Connectors. Οι πρώτοι χρησιμοποιούνται για την εισαγωγή δεδομένων στο Kafka ενώ οι δεύτεροι για την εξαγωγή σε οποιοδήποτε άλλο σύστημα. Τέλος, για την οπτικοποίηση της εν λόγω διαδικασίας χρησιμοποιήθηκε το Kafka Connect UI μέσω του οποίου απεικονίζονται σχηματικά οι συνδέσεις μεταξύ των Connectors και του Kafka.

Στην εφαρμογή μας χρησιμοποιήσαμε συνολικά 4 SourceConnectors και 3 SinkConnectors. Οι SourceConnectors έπαιρναν δεδομένα από μία βάση δεδομένων και 3 αρχεία csv ενώ οι SinkConnectors μετέφεραν τα δεδομένα από τα topics στο hdfs. Στη συνέχεια φαίνεται η συνολική εικόνα όλων.



Ακόμη, παρουσιάζεται η παραμετροποίηση ενός SourceConnector ο οποίος παίρνει δεδομένα από μια sqlite βάση και τα αποθηκεύει στο αντίστοιχο kafka topic.

JdbcSourceConnector

RESUMEDELETE

Jdbc

Kafka is PAUSED

TASKS

0 [connect:8083]

TOPICS

categories

CONFIGURATION

EDIT

```
1 {
2   "connector.class": "io.confluent.connect.jdbc.JdbcSourceConnector",
3   "mode": "bulk",
4   "topic.prefix": "categories",
5   "tasks.max": "1",
6   "query": "select * from cat",
7   "name": "JdbcSourceConnector",
8   "connection.url": "jdbc:sqlite:opt/db/categories.db",
9   "value.converter": "org.apache.kafka.connect.storage.StringConverter",
10  "key.converter": "org.apache.kafka.connect.storage.StringConverter"
11 }
```

Στη συνέχεια, βλέπουμε έναν ακόμη SourceConnector ο οποίος διαβάζει δεδομένα από αρχείο csv και τα τοποθετεί σε ένα topic.

FileStreamSourceConnector

RESUMEDELETE

File

Kafka is PAUSED

TASKS

0 [connect:8083]

TOPICS

events

CONFIGURATION

EDIT

```
1 {
2   "connector.class": "org.apache.kafka.connect.file.FileStreamSourceConnector",
3   "file": "/opt/dataset/events.csv",
4   "tasks.max": "1",
5   "name": "FileStreamSourceConnector",
6   "topic": "events",
7   "value.converter": "org.apache.kafka.connect.storage.StringConverter",
8   "key.converter": "org.apache.kafka.connect.storage.StringConverter"
9 }
```

Τέλος, παρουσιάζουμε ένα από τους SinkConnectors, ο οποίος διαβάζει από ένα συγκεκριμένο topic και μετακινεί τα δεδομένα στο hdfs με κατάλληλη μορφοποίηση.

HdfsSinkConnector-Categories

▶ RESUME DELETE

Kafka → **HDFS** is PAUSED

TASKS 0 [connect:8083]

TOPICS categories

CONFIGURATION EDIT

```
1 {  
2   "connector.class": "io.confluent.connect.hdfs.HdfsSinkConnector",  
3   "flush.size": "3",  
4   "topics": "categories",  
5   "tasks.max": "1",  
6   "hdfs.url": "hdfs://namenode:8020",  
7   "name": "HdfsSinkConnector-Categories",  
8   "value.converter": "org.apache.kafka.connect.storage.StringConverter",  
9   "key.converter": "org.apache.kafka.connect.storage.StringConverter"  
10 }
```


3. Περιγραφή δεδομένων και Στατιστικά

Το dataset που χρησιμοποιήσαμε ονομάζεται [Retailrocket recommender system dataset](https://www.kaggle.com/retailrocket/recommender-system-dataset) και διατίθεται ελεύθερα μέσω της ιστοσελίδας www.kaggle.com, κατόπιν εγγραφής. Πρόκειται για καταγραφή της δραστηριότητας σε ένα online κατάστημα πώλησης προϊόντων σε ένα διάστημα 4,5 μηνών.

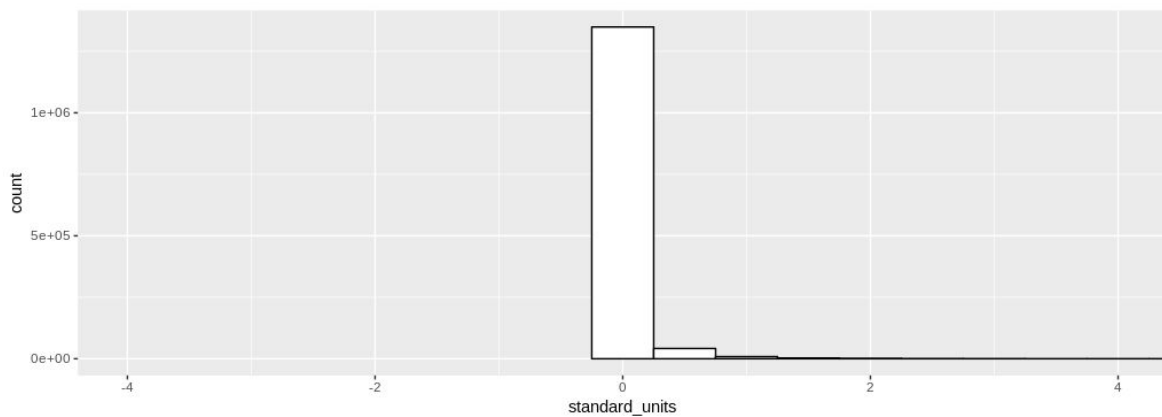
Τα δεδομένα είναι οργανωμένα σε 3 αρχεία .csv και έχουν το ακόλουθο σχήμα: (όλοι οι χρόνοι είναι εκφρασμένοι σε UNIX timestamp με ακρίβεια ms)

➤ events.csv (sample)

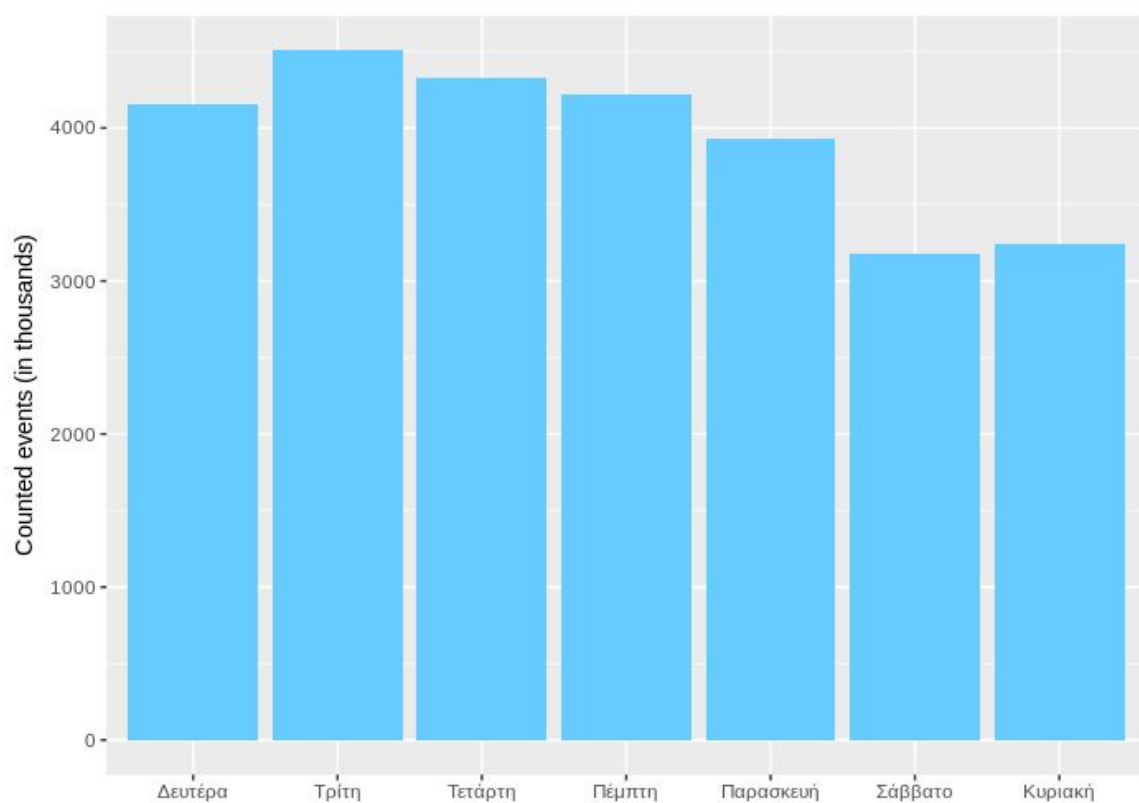
timestamp	visitorid	event	itemid	transactionid
1433221332117	257597	view	355908	
...
1433223236124	287857	addtocart	5206	
...
1433224070841	1398644	view	135256	
...
1433222276276	599528	transaction	356475	4000
...

Περιέχει πληροφορίες για τα events που καταγράφηκαν την περίοδο που καλύπτουν τα δεδομένα. Συγκεντρωτικά στοιχεία:

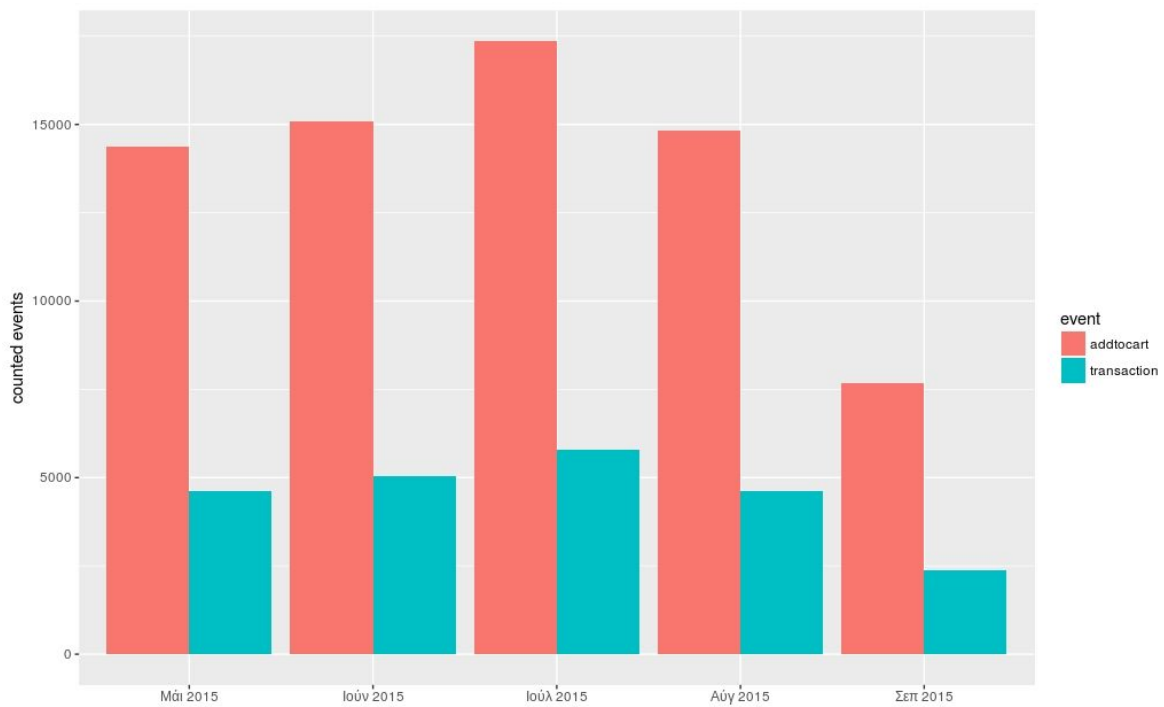
Συνολικός αριθμός γραμμών	2756101
Εγγραφές ανά είδος event	view -> 2664312 addtocart -> 69332 transaction -> 22457
Συνολικός αριθμός χρηστών	1407580
Συνολικός αριθμός προϊόντων	235061
Κατανομή αριθμού κινήσεων ανά χρήστη	max -> 7757 min -> 1 mean -> 1.96 median -> 1 sd -> 12.58
Αριθμός χρηστών με περισσότερες από 1 κινήσεις	406020
Αριθμός χρηστών με περισσότερες από 10 κινήσεις	19582



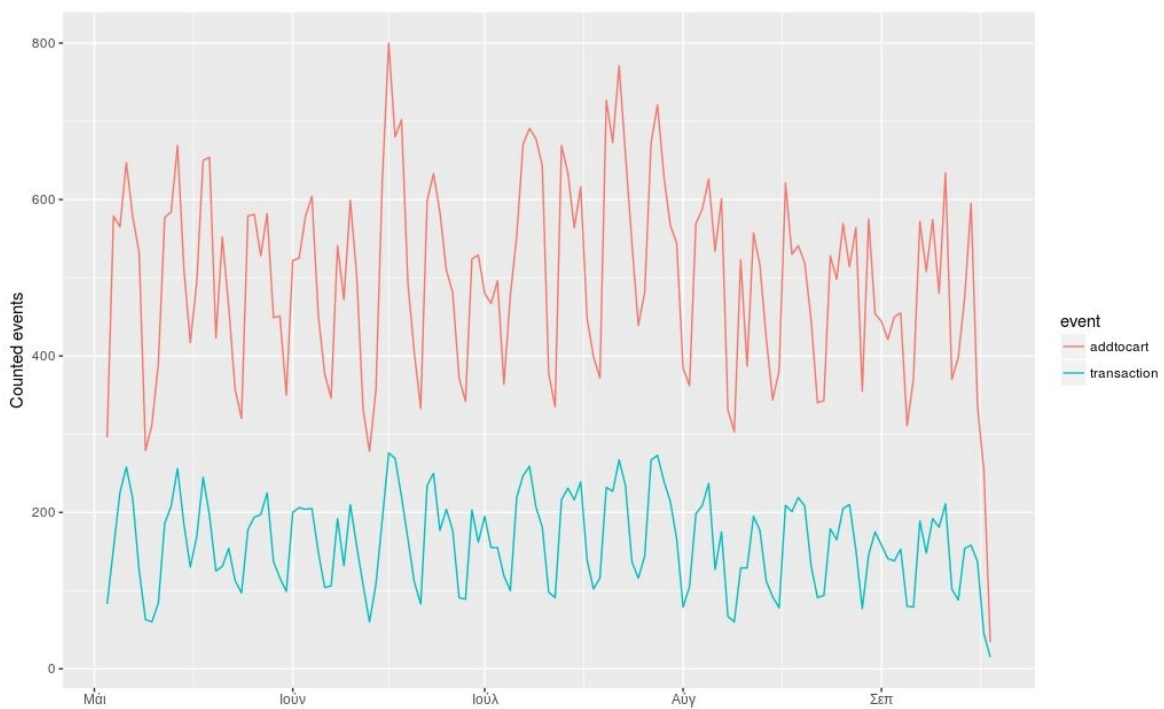
Απόσπασμα της κατανομής του αριθμού κινήσεων σε standard units (μόνο για τιμές μεταξύ -4 και 4)



Σύνολο καταγεγραμμένων κινήσεων ανά ημέρα της εβδομάδας



Αριθμός addtocart και transaction κινήσεων ανά μήνα



Αριθμός addtocart και transaction κινήσεων ανά ημέρα

➤ item_properties.csv (sample)

timestamp	itemid	property	value
1435460400000	460429	categoryid	1338
1441508400000	206783	888	1116713 960601 n277.200
1439089200000	395014	400	n552.000 639502 n720.000 424566
1431226800000	59481	790	n15360.000
1431831600000	156781	917	828513
1436065200000	285026	available	0
...

Περιέχει τις τιμές που παίρνουν οι ιδιότητες των αντικειμένων στην πάροδο του χρόνου. Το csv είχε χωριστεί σε 2 μέρη λόγω μεγέθους. Για να χρησιμοποιηθούν οι τιμές πρώτα χρειάστηκε να αφαιρεθούν τα διπλότυπα.

Συνολικός αριθμός γραμμών	20275902
Συνολικός αριθμός γραμμών μετά τον καθαρισμό των διπλοτύπων	12778737
Συνολικός αριθμός προϊόντων	417053
Συνολικός αριθμός ιδιοτήτων	1104

➤ category_tree.csv (sample)

categoryid	parentid
570	9
1691	885
536	1691
231	
542	378
...	...

Δέντρο κατηγοριών στις οποίες ανήκουν τα προϊόντα¹.

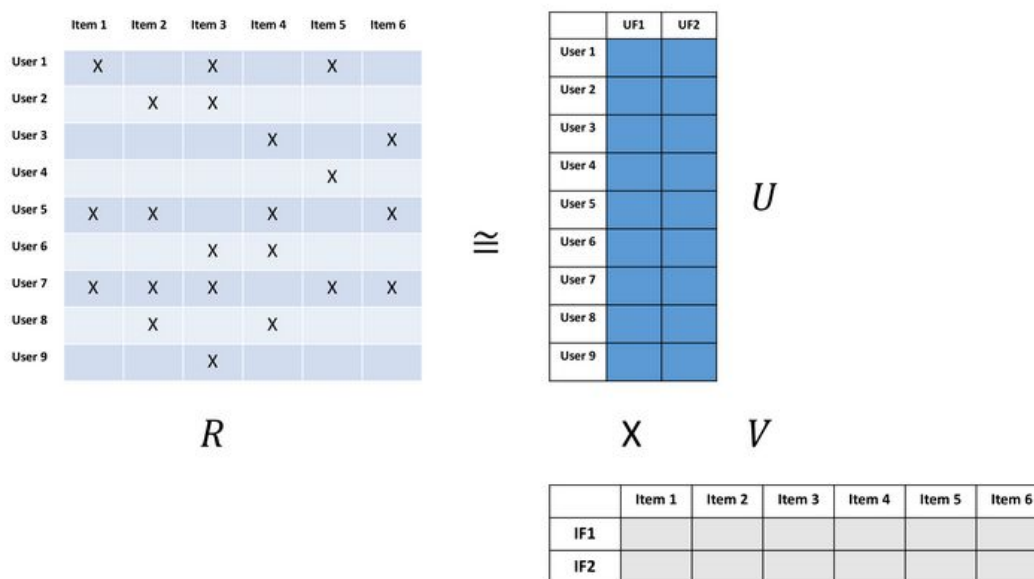
Συγκεντρωτικά στοιχεία:

Συνολικός αριθμός γραμμών	1669
Αριθμός βασικών κατηγοριών	25

¹ Επιπλέον, υλοποιήθηκε συνάρτηση σε java για την εύρεση της βασικής (γενικότερης) κατηγορίας στην οποία ανήκει κάθε categoryid.

4. Περιγραφή συστήματος συστάσεων

Τελικός στόχος της εργασίας αποτελεί η υλοποίηση μέσω Spark ενός συστήματος συστάσεων προϊόντων στους επισκέπτες του καταστήματος. Οι τεχνικές του collaborative filtering στοχεύουν να γεμίσουν έναν πίνακα συσχετίσεων μεταξύ χρηστών-αντικειμένων, ξεκινώντας από δεδομένες τιμές του πίνακα. Η συμπλήρωση των τιμών που λείπουν βασίζεται στην υπόθεση ότι οι χρήστες μεταξύ τους, όπως και τα αντικείμενα συνδέονται με ένα μικρό σύνολο από latent factors (κρυμμένα χαρακτηριστικά), τα οποία είναι εξ' αρχής άγνωστα και ο αλγόριθμος προσπαθεί να ανακαλύψει. Η παραπάνω μέθοδος αποτελεί ένα μαθηματικό μοντέλο που ονομάζεται matrix factorization (ή matrix decomposition) και περιγράφεται σχηματικά από την παρακάτω εικόνα.



4.1 Προεπεξεργασία δεδομένων

Όπως αναφέρθηκε, ο αλγόριθμος απαιτεί κάποιες γνωστές τιμές στον πίνακα συσχετίσεων. Στην παραπάνω εικόνα, οι τιμές είναι τα X του αριστερού πίνακα.

Συνήθως, οι τιμές αυτές προέρχονται από τη βαθμολογία που δίνει κάποιος χρήστης σε κάποιο προϊόν και πρόκειται για σαφείς προτιμήσεις (explicit preferences), ωστόσο στην περίπτωσή μας, η μόνη συσχέτιση που υπάρχει μεταξύ χρηστών και προϊόντων προκύπτει από τα events -view, addtocart και transaction- και είναι υπονοούμενη ανάδραση (implicit feedback).

Για να εκφράσουμε ποσοτικά την ανάδραση αυτή, δώσαμε ένα βάρος ανάλογα τον τύπο του event και στη συνέχεια υλοποιήσαμε ένα MapReduce υπολογισμό. Συγκεκριμένα, ένα view event έχει βάρος 0.3, ένα addtocart event έχει βάρος 0.7, ενώ ένα transaction έχει βάρος 1.0. Στη φάση του map, θέτουμε ως κλειδί ένα user-item συνδυασμό με τιμή κάθε φορά το βάρος του event που έχει καταγραφεί, ενώ στη φάση του reduce, προσθέτουμε τα βάρη ανά

κλειδί (user-item), σχηματίζοντας τελικά μία συνολική προτίμηση του χρήστη προς το συγκεκριμένο προϊόν με βάση τη συμπεριφορά του. Παραδείγματος χάρη, εάν υπήρχε ένας χρήστης με $id=1$ και είχε δει το προϊόν με $id=10$, 2 φορές, ενώ το είχε αγοράσει 1 φορά, στην αντίστοιχη θέση του πίνακα θα υπήρχε το βάρος 1.6.

Αξίζει να σημειωθεί, πως δεδομένου ότι συνήθως ένας χρήστης δεν αλληλεπιδρά με πολλά προϊόντα, ο πίνακας που σχηματίζεται είναι αρκετά αραιός. Παρόλα αυτά, οι κενές θέσεις του πίνακα είναι κρίσιμες για την εξαγωγή αποτελεσμάτων και συστάσεων. Σε αντίθεση με το explicit feedback, σε αυτήν την περίπτωση, απουσία βάρους στον πίνακα σημαίνει έλλειψη πληροφορίας και όχι δυσαρέσκεια για το προϊόν. Επιπλέον, η μελέτη συμπεριφοράς των καταναλωτών συνήθως περιέχει θόρυβο, καθώς η αγορά ενός προϊόντος μπορεί να έγινε σαν δώρο προς τρίτο πρόσωπο, είτε να μην άφησε το χρήστη ευχαριστημένο. Στο τελευταίο πρόβλημα που μόλις τέθηκε, μέρος της λύσης είναι ο συνυπολογισμός της συχνότητας ενός συμβάντος στο βάρος, καθώς εάν ένα προϊόν αγοράστηκε περισσότερες από μία φορές, είναι πιο πιθανό να άρεσε στο χρήστη.

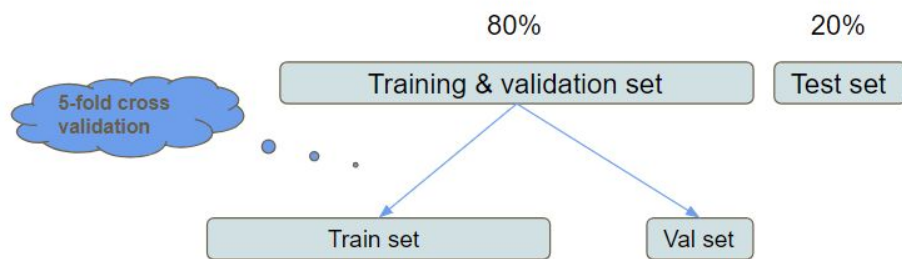
4.2 Alternating Least Squares (ALS) Αλγόριθμος

Για την πρόβλεψη των κενών τιμών του πίνακα συσχετίσεων μέσω των latent factors, η βιβλιοθήκη mllib του Spark χρησιμοποιεί τον αλγόριθμο Alternating Least Squares (ALS). Ο ALS είναι ένας επαναληπτικός αλγόριθμος, ο οποίος εναλλάξ σε κάθε επανάληψη φτιάχνει τον έναν πίνακα με factors και επιλύει ως προς τον δεύτερο, έως ότου συγκλίνει. Η εναλλαγή στην επιλογή του πίνακα που κάθε φορά βελτιστοποιείται, καθόρισε το όνομα του αλγορίθμου (alternating).

4.2.1 Cross Validation

Για την εξαγωγή και αξιολόγηση του καλύτερου μοντέλου, τα δεδομένα μας διαχωρίστηκαν αρχικά με ποσοστό 80%-20% σε train_and_validation set και test set. Στη συνέχεια, στο train_and_validation set εφαρμόστηκε 5-fold validation, δηλαδή διασπάστηκε σε πέντε ίσα και μη επικαλυπτόμενα μέρη, και κάθε ένα από τα 5 μέρη (validation set) επαληθεύει το αποτέλεσμα του τρέχοντος μοντέλου, το οποίου έχει εκπαιδευτεί με τα υπόλοιπα %, δηλαδή το train set. Στα βήματα του cross validation εντάχθηκε επιπλέον ο πειραματισμός με τις παραμέτρους του αλγορίθμου, ώστε να ελαχιστοποιηθεί το σφάλμα. Οι βέλτιστες τιμές των παραμέτρων που προέκυψαν είναι οι εξής:

- rank - αριθμός latent factors: 20
- iterations - μέγιστος αριθμός επαναλήψεων (εάν δε συγκλίνει): 20
- lambda - regularization παράμετρος του ALS: 0.1
- alpha - εμπιστοσύνη στις υπονοούμενες προτιμήσεις των χρηστών: 0.5



Το μοντέλο με το καλύτερο μέσο σφάλμα μετά το 5-fold validation, επιλέγεται ως το καλύτερο και το τελικό του σφάλμα αξιολογείται στο test set, το οποίο είναι ανεξάρτητο από τη διαδικασία της εκπαίδευσης, προκειμένου να αποφευχθεί το overfitting.

4.2.2 Μετρική percentile ranking

Η κλασική μετρική λάθους του αλγορίθμου ALS για explicit feedback είναι το Root Mean Squared Error (RMSE) και αυτή είναι και η μετρική που έχει υλοποιημένη το Spark. Ωστόσο, η μετρική αυτή απαιτεί γνώση σχετικά με το ποια προϊόντα δεν αρέσουν στο χρήστη, προκειμένου να έχει νόημα, γνώση ανύπαρκτη σε δεδομένα που προκύπτουν από τη συμπεριφορά των καταναλωτών.

Όπως αναφέρεται στο paper το οποίο εισήγαγε την έννοια του implicit feedback στον ALS αλγόριθμο, [Collaborative Filtering for Implicit Feedback Datasets](#), μπορεί να χρησιμοποιηθεί το percentile ranking ως μετρική σφάλματος.

$$\overline{rank} = \frac{\sum_{u,i} r_{ui}^t rank_{ui}}{\sum_{u,i} r_{ui}^t}$$

Ως r_{ui}^t ορίζεται το βάρος - προτίμηση του χρήστη u προς το αντικείμενο i όπως ορίζεται στο test set, και ως $rank_{ui}$ ορίζεται η θέση του προϊόντος i στο σύνολο συστάσεων για το χρήστη u , κανονικοποιημένη ως προς το 100.

Διαισθητικά, το percentile ranking εκφράζει το πόσο ψηλά βρέθηκε στη λίστα προτιμήσεων που προέκυψε από τον αλγόριθμο ένα προϊόν για ένα χρήστη, σε σχέση με την πραγματική προτίμηση. Όπως αναφέρεται χαρακτηριστικά στη δημοσίευση, η τυχαία επιλογή προτεινόμενων προϊόντων οδηγεί σε περίπου 50% ποσοστό λάθους, συνεπώς στόχος είναι μικρότερο μέσο rank.

Για να μπορέσουμε να χρησιμοποιήσουμε τη μετρική αυτή, χρειάστηκε να υλοποιηθεί ταξινόμηση των προτεινόμενων προϊόντων και κανονικοποίηση της θέσης καθενός ανά χρήστη.

Το τελικό ποσοστό λάθους που επιτεύχθηκε με την παραπάνω παραμετροποίηση στο test set των δεδομένων μας ήταν περίπου **18%**.

5. Χρήσιμη βιβλιογραφία και ιστοσελίδες

5.1 Για collaborative filtering μέσω spark

- <https://spark.apache.org/docs/2.3.0/mllib-collaborative-filtering.html>
- Hu Y., Koren Y., Volinsky C. "Collaborative Filtering for Implicit Feedback Datasets" Data Mining, 2008. ICDM '08 (2008). - [\[url\]](#)
- <https://mapr.com/ebooks/spark/08-recommendation-engine-spark.html>
- <https://dzone.com/articles/building-sales-recommendation-engine-with-apache-s>

5.2 Για sparkR

- <https://spark.apache.org/docs/latest/sparkr.html#starting-up-from-rstudio>

5.3 Για Kafka

- <https://docs.confluent.io/1.0/platform.html>