УДК 004.42

А.М. Глибовець, М.А. Гонтар

РОЗРОБКА WEB-ПРОТОТИПУ ДЛЯ АНАЛІЗУ ВИСЛОВЛЮВАНЬ КОРИСТУВАЧІВ СОЦІАЛЬНОЇ МЕРЕЖІ TWITTER

Описано розроблений програмний прототип статистичного аналізу висловлювань у соціальних мережах з попереднім фільтруванням висловлювань неінформативного характеру, відбором характеристичних слів і використанням наївного байєсового класифікатору з машинним навчанням для аналізу суджень користувачів.

Вступ

Соціальні мережі (СМ) – надзвичайно цікавий об'єкт дослідження, що відкриває широкі можливості аналізу соціальної взаємодії між людьми, прогнозування їх поведінки, класифікації та типології соціальної поведінки, моделювання інформаційних потоків у мережах, розвитку бізнес проектів. СМ – це ідеальний експериментальний майданчик для вивчення соціальної поведінки людей, апробації технологій впливу та реклами, просування сучасних бізнес проектів [1, 2].

Розвиток СМ поставив питання про аналіз даних, якими вони наповнені.

Актуальність проблеми аналізу уподобань користувачів полягає у тому, що, вивчивши їх уподобання, на основі даних СМ можна здійснити моделювання соціальних, економічних та політичних процесів та створювати нові сервіси, бізнес рішення, програмні продукти.

Однак, на сьогоднішній день, не існує готових програмних рішень, що дозволять аналізувати висловлювання користувачів соціальних мереж. Немає чітких критеріїв та підходів до аналізу висловлювань у контексті соціальних мереж.

Тут ми опишемо основні характеристики програмної реалізації розробленого нами прототипу, який дозволяє автоматично збирати та аналізувати висловлювання користувачів соціальних мереж у режимі реального часу з застосуванням попереднього фільтрування висловлювань користувачів, які дозволили зменшити обсяг неінформативних висловлювань та застосування наївного баєсівського класифікатора з машинним навчанням для первинного аналізу повідомлень і використання експертного навчання для покращення ефективності аналізатора.

Практичне застосування прототипу системи аналізу висловлювань можна уявляти по різному, на нашу думку, кінцевий користувач – це хтось з області маркетингу, реклами тощо. Для зазначених працівників, важливо отримати легкий, простий та зрозумілий інтерфейс користування платформою. Для кращого зрозуміння їхніх потреб та психології мислення, було проведено багато зустрічей з представниками знаних фірм, які б хотіли користуватись такою аналітичною платформою. Звісно ж, що їх цікавила інформація про позитивні відгуки щодо своєї компанії та негативні щодо компаній-конкурентів.

Проаналізовано вимоги майбутніх користувачів та сформульовано список вимог до прототипу користувача. Останній хотів мати можливість:

- переглядати та видаляти існуючі завдання за аналізом;
- зазначити назву завдання, короткий опис, умови відбору висловлювань до завдання при створенні;
- переглядати вибірку висловлювань що потрапила до завдання, при цьому кожне висловлювання має містити інформацію про дату публікації, користувача, а також оцінку (позитивна, негативна);
- змінювати оцінку висловлювання, при цьому, оцінка повинна мати інше візуальне забарвлення для кращої ідентифікації оцінки системи та експерта;

- переглядати загальну статистичну інформацію за аналізом завдання – кількість позитивних, негативних висловлювань (з можливістю переходу до них);
- з легкістю проглядати висловлювання;
- фільтрувати висловлювання за оцінкою;
- мати захищений доступ до системи.

Проаналізувавши вимоги, можна зробити висновок, що інтерфейс користувача не має бути складним у реалізації, проте аналітична система все ж потребує складних архітектурних рішень в основі. Правильно спроектованою, вона зможе надати швидкий та легкий доступ користувачам.

1. Архітектура прототипу

Для збереження статистичних даних необхідно обрати сховище. На даному етапі, як сховище розглядалась реляційна і нереляційна бази даних. Перш за все, слід оцінити, які можливі сутності передбачатимуть статистичні дані, а потім проаналізувати їх та обрати найкращу систему керуванння даними, можливо комбінований підхід.

Проектування сутностей та вибір RDMS. Отже, перш за все, потрібен засіб для роботи користувача аналітичної платформи. Сутність «користувач» (User) буде відносно відокремленою від загальної моделі і не впливатиме на аналіз, вона потрібна лише для аутентифікації, підтримування сесії Web-додатку та позначення завдання (хто його власник). Для сутності, достатньо полів, у яких будемо запам'ятовувати унікальне ім'я, закодований пароль та інформацію про стан облікового запису.

Для збереження проміжних даних про висловлювання, потрібна сутність «висловлювання» (Tweet), з інформацією про висловлювання, дату публікації та користувача, що опублікував його. Така сутність є абсолютно незалежною і не потребує реляційної бази даних (DB).

При первинному аналізі висловлювання, будемо розкладати його на слова. Звідси, необхідна сутність «слово» (Word), що характеризуватиме унікальне слово. Варто зазначити, що «слова» не потребують залежності з сутністю «висловлювання», оскільки це не є необхідністю, «слова» використовуватимуться лише для їх пошуку серед різних оцінок.

Оскільки користувач платформи захоче створювати унікальні завдання, потрібно ввести поняття «завдання» (Job), що зможе описати суть завдання – його назву, короткий опис, а також критерій для фільтрування висловлювань. Однозначно, що в кожного завдання має бути власник – «користувач». Сутність «завдання» буде інтегровано в модель аналізу, що накладає певні обмеження – тісний реляційний зв'язок з іншими моделями.

Для оцінювання висловлювання, введемо поняття «оцінка» (Mark), що матиме опис оцінки, а також унікальний ідентифікатор.

Для об'єднання всіх понять у реляційній моделі, потрібно ввести дві додаткові реляційні сутності – «аналіз» (Analysis) та «слова в аналізі» (Words In Analysis). Кожна з сутностей має об'єднати унікальні ідентифікатор інших сутностей, щоб задовольнити головну вимогу для кожного «завдання», кожне «висловлювання» повинно мати оцінку та посилання на набір «слів», що зустрічаються у «висловлюваннях».

Перевага в нереляційному підході полягає у можливості ефективного збереження великих обсягів даних і виграшу в швидкодії при їх обробці. Проте, для нашої задачі, не всю модель потрібно зберегти в нереляційній базі, оскільки більшість сутностей потребують зв'язків між собою. Єдине, що можна відокремити від іншої моделі – це вибірка висловлювань («твітів»), все інше, тобто «оцінки», «завдання», «слова» тощо, потрібно якось реляційно поєднати (рис. 1).

Реляційний підхід має основний недолік – ефективність зчитування/запису при великих навантаженнях. Проте, його можна компенсувати заощадливими і нечастими обчисленнями. А от, збереження

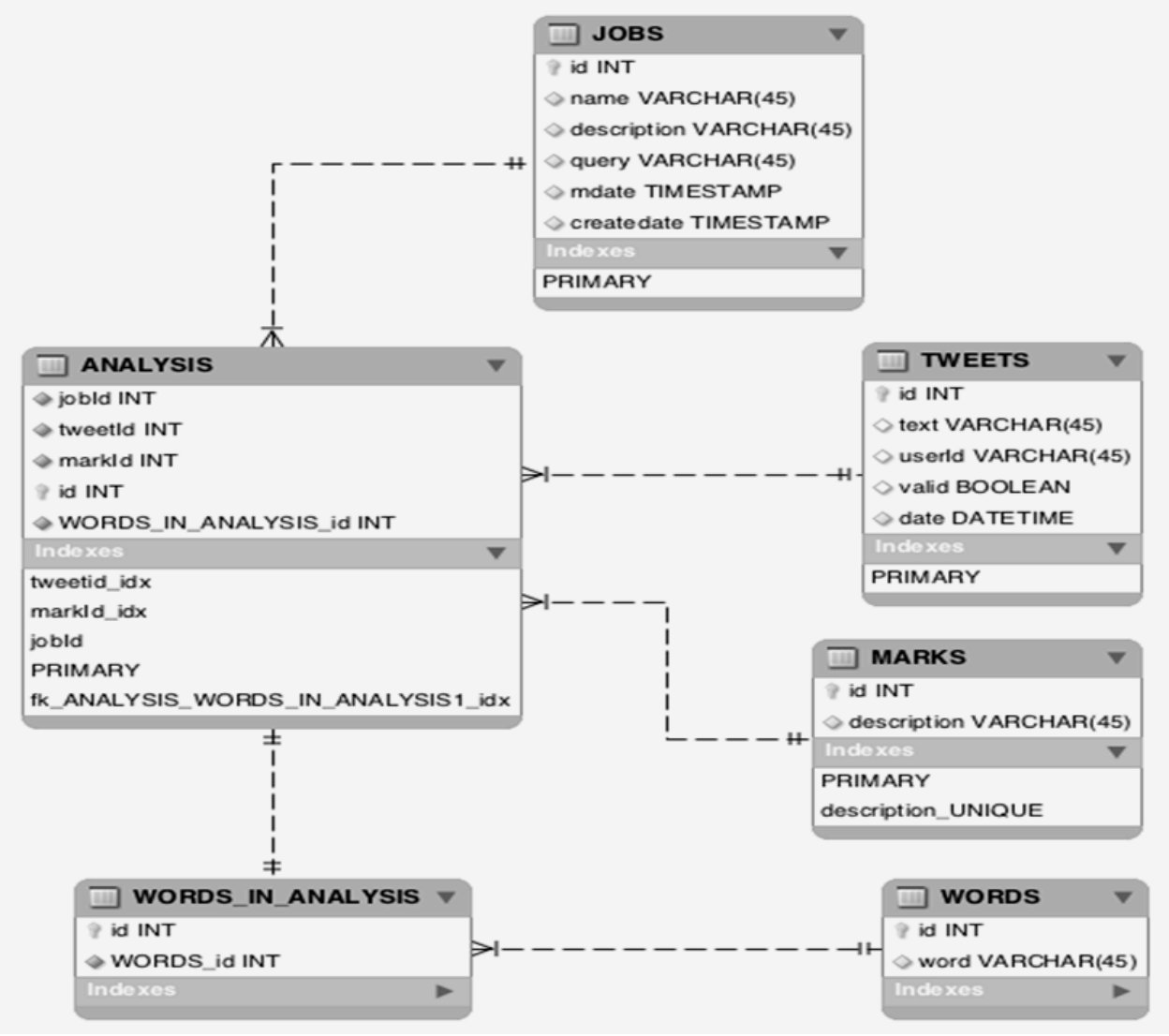

Рис. 1. Модель реляційної бази даних для збереження статистичних даних

статистичних даних до неструктурованого нереляційного сховища важко компенсувати взагалі. Тому ми обрали реляційну модель на основі MySQL RDMS платформи.

Взаємодія прототипу з BD-платформою. Оскільки висловлювання будуть попередньо опрацьовуватись іншою платформою, то потрібно налагодити зв'язок між прототипом та такою платформою.

Обрана BD-платформа Cloudera, має інтегрований сервіс швидкого пошуку Apache Solr, що має реалізований REST-ful Web-інтерфейс у форматах JSON/XML, який дозволяє з легкістю налагодити спілкування клієнт-сервер (рис. 2). На схемі можна розгледіти сервер прототипу аналітичної платформи Social Glutton (виступає як клієнт), Cloudera Server – BD платформа, яка зберігатиме всі висловлювання соціальних мереж та Twitter Server – сервер соціальної мережі.

Варто додати, що спілкування через REST інтерфейс у такій моделі є виправданим, оскільки задача не потребує спілкуватись в режимі реального часу зі сховищем всіх висловлювань. Навпаки, необхідно використати техніку синхронізації, що зменшить навантаження на центральний сервер-сховище. Синхронізацію, як таку, варто проводити у визначені інтервали часу, в залежності від пріоритетності завдання.

Схема спілкування центрального сервера та сервера соціальної мережі описана в [2].

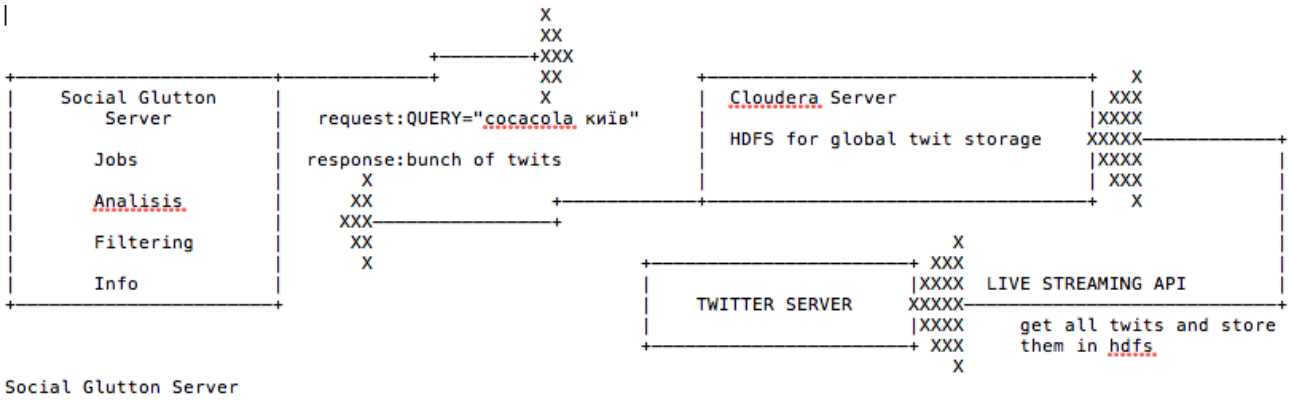

Рис. 2. Схема спілкування між прототипом аналітичної мережі, BD-платформою Cloudera та сервером соціальної мережі Twitter

Налаштування системи розподіленого опрацювання та збереження висловлювань. Для налаштування Cloudera, необхідно вирішити проблему місце розташування сервера. Ідеальний варіант – відокремити його та забезпечити безперебійну досяжність у мережі.

Cloudera передбачає можливість вирішення даної проблеми двома способами: розгорнути платформу в хмарному середовищі AWS, розгорнути платформу на Linux RedHat.

Перший варіант є ідеальним, та, на жаль, його не вдалось реалізувати із-за високої платіжної ставки компанії Amazon. Безкоштовний та пробний варіант використання хмарного середовища AWS надає занадто мало ресурсів для безперебійної роботи платформи Cloudera.

Отже, довелось розгортати платформу в середовищі Linux RedHat.

Після встановлення всіх необхідних компонентів, можна керувати системою через панель управління Cloudera Manager (рис. 3).

Варто додати, що платформа налаштована з такими системними ресурсами:

- потужність процесора – 2.2 GHz (Intel Core i7);
- обсяг оперативної пам'яті (RAM) – 4096 MB;
- обсяг пам'яті жорсткого диску (HDD) – 64 GB.

Не зважаючи на те, що Cloudera – це платформа для розподілених обчислень, вона була налаштована в «псевдорозподіленому» режимі, тобто в режимі одного кластера-точки. Додати ще декілька точок і масштабувати платформу не важко, але це потребує більше фізичних ресурсів. Робота в режимі однієї точки не відзначається високою продуктивністю і надійністю, проте в умовах даної задачі є абсолютно самодостатньою.

Встановлення додаткових компонентів. Під час запуску сервера, Cloudera автоматично запускає сервіси першої необхідності, хоча необхідні для синхронізації з соціальною мережею сервіси виявились відсутніми. Необхідно додатково встановити і запустити Apache Solr, що використовується для швидкого пошуку по базі даних, та Flume, що використовується для потокової Twitter-інтеграції.

Для того, щоб запустити процес потокового скачування «твітів», необхідно перш за все зареєструватись розробником соціальної мережі Twitter. Далі, необхідно створити свій додаток, щоб отримати унікальний API ключ, що дозволить використовувати Twitter API.

Маючи унікальний ключ розробника та API ключ, можна налаштувати сервіс потокового скачування «твітів». Для цього потрібно описати модель даних, що будуть приходити з іншого боку «труби» і надати необхідні ключі.

Після того як модель додано, запуск Flume з ключами виглядає так:

flume-ng-agent start
[CONSUMER_KEY]
[CONSUMER_SECRET]
[ACCESS_TOKEN]
[ACCESS_TOKEN_SECRET]

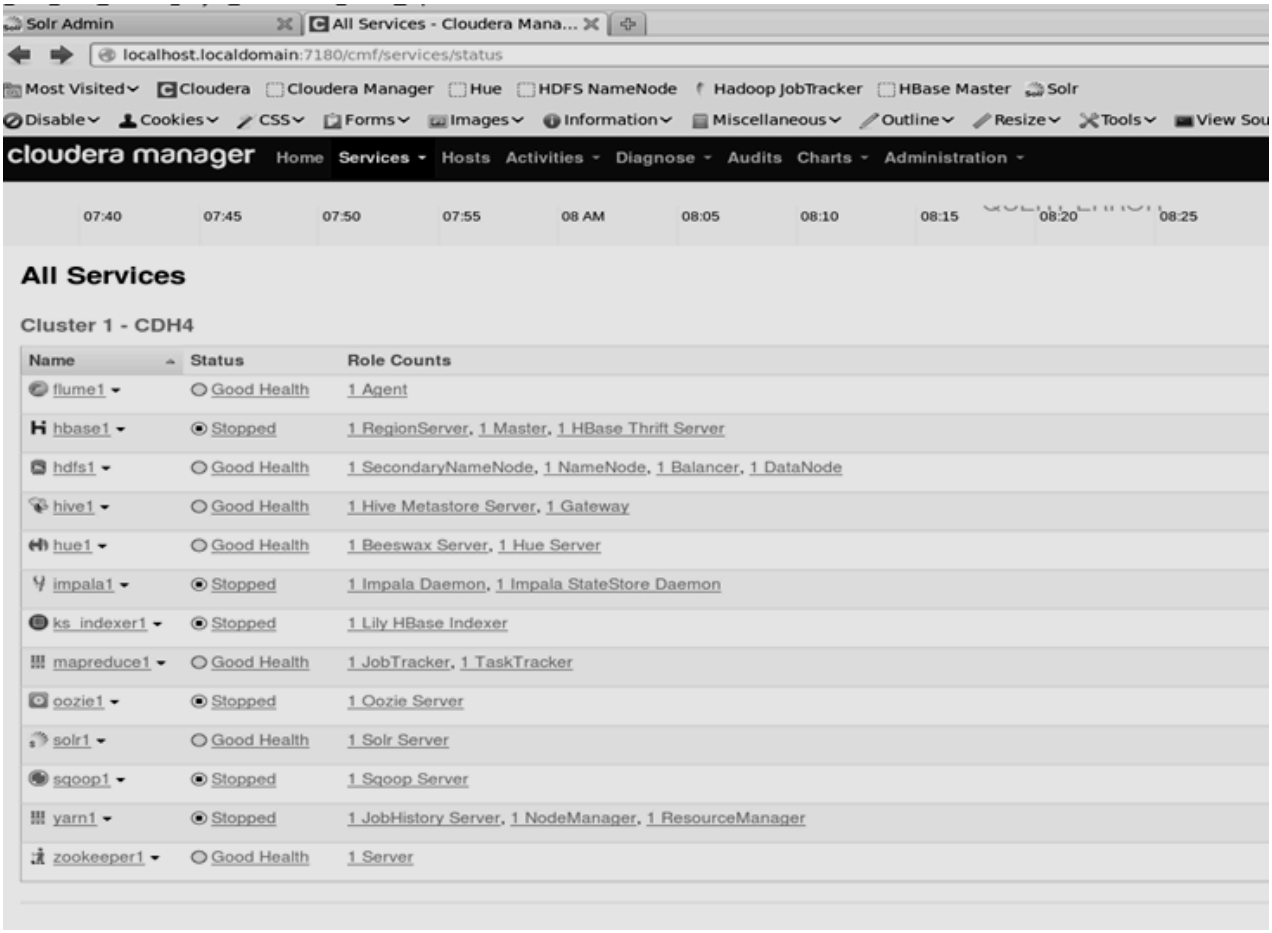

Рис. 3. Панель керування Cloudera Manager та список сервісів платформи

Зупинити сервіс можна відповідно командою – stop.

Для того, щоб перевірити чи скачування «твітів» успішне, достатньо зайти на сторінку модуля Hue і скористатись сервісом Impala Search, що надає можливість інтерактивного.

Сервіс Flume буде використано для інтеграції з Cloudera платформою і синхронізацією «твітів» для завдань. Він надає REST-інтерфейс, що задовольняє поточну архітектуру прототипу аналітичної системи.

2. Проектування та реалізація прототипу для проведення та збереження проміжних результатів статистичного аналізу

Для реалізації прототипу, окрім проблеми моделі та комунікації з іншим сервером, перш за все, необхідно було вирішити поширену проблему набору інструментів для реалізації Web-прототипу.

Базовий набір інструментів. До поля зору найпопулярніших рішень, серед швидкої розробки Web-додатків, потрапили Play!, Grails, Ruby on Rails (RoR).

Варто навести порівняльні характеристики вищезазначених інструментів (таблиця). Всі рішення є простими в налаштуванні і продуктивності, а також всі заохочують TDD-парадигму. Основна відмінність RoR від інших двох – це використання мови програмування Ruby, що є скриптовою. Всі інші характеристики є більш менш однаковими. Тобто, стек технології у кожному рішенні досить великий, і є з чого обирати. Основним критерієм при виборі готового інструментарію, стала мова програмування. Вибір впав на динамічний groovy і grails.

Groovy легко інтегрується в інший Java-код і легко налаштувати «заміну на льоту», що так необхідна при розробці Web-додатків [3]. Ще одною перевагою grails, виявилося строге структурування додатку на сервіси, контролери, модель та

Таблиця. Порівняльні характеристики інструментів

Особливість	Grails	RoR	Play!
Принципи розробки	Test Driven Development (TDD)	TDD	TDD
Шаблон розробки	Model View Controller (MVC), Dependency Injection	MVC	MVC
Інтер-націоналізація	+	+	+
Операційна система	JVM-сумісна	Linux, Windows, Mac OS X	JVM-сумісна
Мова програмування	Groovy, Scala, Java, Clojure	Ruby	Java, Scala
DB	MySQL, NoSQL, Oracle, SQLite	MSSQL, MySQL, MongoDB, PostgreSQL, Drizzle, Oracle, Redis, SQLite	MSSQL, MySQL, MongoDB, NoSQL, PostgreSQL, Hazelcast, Oracle
Мова шаблонів	GSP, HAML, Mustache, Handlebars JS	HAML, ERB, Radius	Groovy, Japid, Scala Template Engine
REST-ful	+	+	+
АОП	+	–	+
Сервер	Jetty, Jboss, Tomcat	Jboss	Jetty, Jboss, Tomcat

сторінки (притаманне MVC-парадигмі). Такий підхід полегшує тестування окремих класів і примушує розробників мислити модульно.

Користувацький інтерфейс. Для швидкої реалізації користувацького інтерфейсу, вирішено використати розробку Twitter Bottstrap [4] – безкоштовний набір інструментів для створення Web-сайтів, що містить досить великий набір HTML та CSS-базованих шаблонів для всіляких HTML елементів (форм, кнопок тощо), а також інших інтерфейсних компонентів.

Прототип користувацького інтерфейсу на основі Bootstrap, розроблено за допомогою безкоштовного Web-додатку Jetstrap. За допомогою даного сервісу, вдалось легко збудувати скелет сторінок, а також модифікувати їх стилі [5].

Після того, як скелет сторінок був збудований, для їх оптимізації, були створенні gsp-шаблони. Основними перевагами шаблонів є можливість їх функціонального тестування, а також структурування коду (рис. 4).

```
<h2>${job.name}</h2>
<p>${job.description}</p>
<g:if test="${job.dateCreated != null}">
   <p>created at: <g:formatDate date="${job.dateCreated}" type="datetime" style="SHORT" timeStyle="SHORT"/></p>
</g:if>
<g:if test="${job.dateSynced != null}">
   <p>last synced at: <g:formatDate date="${job.dateSynced}" type="datetime" style="SHORT" timeStyle="SHORT"/></p>
</g:if>
```

Рис. 4. Приклад шаблону елемента "Завдання"

Безпека. Для авторизації та аутентифікація користувачів застосовано модуль Spring Security, який надає комплексний і розширювальний підхід до проблеми безпеки.

Даний модуль допоміг розмежувати ролі користувачів та їх доступ до окремих ресурсів прототипу. На даний момент, існує дві ролі для авторизованих користувачів: «адміністратор» та «користувач» [6].

Роль «користувач» – це базовий вид авторизованих користувачів, який може тільки керувати «завданнями». «Адміністратор» розширює роль «користувача» і має доступ до керування системними сервісами.

Синхронізація висловлювань. Комунікація між Web-прототипом та платформою-сховищем висловлювань користувачів відбувається через REST-інтерфейс. Для зменшення навантаження на сховище, встановлено обмеження на частоту синхронізації – часовий інтервал в 5 хв. Для усунення ситуації, коли одночасно синхронізація відбуватиметься для великої кількості завдань, що дуже сильно завантажить сервер-сховище, було реалізовано власну чергу завдань синхронізації.

Черга представляє собою набір завдань на синхронізацію, де кожне містить інформацію про пошуковий запит та унікальний ідентифікатор завдання. Для забезпечення асинхронного лімітованої синхронізації – встановлено ліміт на проведення максимальної кількості одночасних операцій синхронізації – 3. Така модель досить гарно себе зарекомендувала при значній кількості завдань, проте при малій кількості завдань працює трішки довше ніж підхід «в лоб».

Синхронізаційний запит складається з декількох параметрів, найважливішим з яких є запитовий рядок. Очікуваний формат відповіді також зазначається як параметр. Основним форматом спілкування клієнт-сервер обрано JSON (рис. 5). Відповідь містить набір висловлювань, що представляють опис загального висловлювання соціальної мережі Twitter.


```
{
 "responseHeader":{
  "status":0,
  "QTime":7,
  "params":{
   "indent":"true",
   "q":"putin",
   "wt":"json"}},
 "response":{"numFound":8,"start":0,"docs":[
   {
    "text":"I think Putin has gone #FullGodwin\n\nhttps://t.co/iYPmxID09q",
    "user_statuses_count":167409,
    "user_screen_name":"GayPatriot",
    "user_location":"South Carolina",
    "user_followers_count":35826,
    "id":"448203325444722689",
    "user_name":"Bonehead Frank",
    "source":"<a href=\"http://twitter.com/download/iphone\" rel=\"nofollow\">Twitter for iPhone</a>",
    "in_reply_to_status_id":[-1],
    "created_at":"2014-03-24T07:02:54Z",
    "retweet_count":0,
    "in_reply_to_user_id":-1,
    "user_friends_count":22297,
    "_version_":1463492255319851008},
```


Рис. 5. Приклад відповіді сервера-сховища висловлювань

3. Аналіз основних компонентів прототипу

Розподілена BD платформа-сховище. Як BD платформа, Cloudera виправдала очікування та продемонструвала себе з кращого боку, продукт зручний у користуванні та керуванні. На день безперервного збереження «твітів» витрачається приблизно 3 ГБ місця на жорсткому диску, що звісно потребує об'ємного сховища в майбутньому. Оскільки навантаження на сервер були не сильними, то під час розробки і тестування вистачило однієї фізичної точки, проте в подальшій розробці та використані бажано розширити можливості розподіленого середовища ще декількома машинами.

Тестування синхронізації Web-клієнта з BD платформою виконано за допомогою додатку JMeter. Для локальних налаштувань, синхронізація не викликала ніяких проблем для платформи (максимальна кількість одночасних запитів – 3, синхронізація відбувалась що 5 хв).

Модуль аналізу. Алгоритм модуля аналізу написаний використовуючи підхід TDD, тому відповідає стандартам, оскільки повністю покритий функціональними тестами. Це означає, що всі математичні поняття алгоритму окремо протестовані на критичні і випадкові значення. Варто зазначити, що модуль аналізу активно співпрацює з базою даних, що негативно впливає на продуктивність. В подальшому, дану область варто покращувати оскільки саме вона найбільше завантажує Web-прототип.

Користувацький інтерфейс містить сторінку управління «завданнями» (рис. 6). Тут можна додати, видалити, переглянути інформацію про завдання, а також припинити / відновити синхронізацію.

Варто зазначити, що для даної сторінки реалізовано посторінкове прогортання (по 10 сторінок), що забезпечує швидку відповідь сервера на запит користувача.

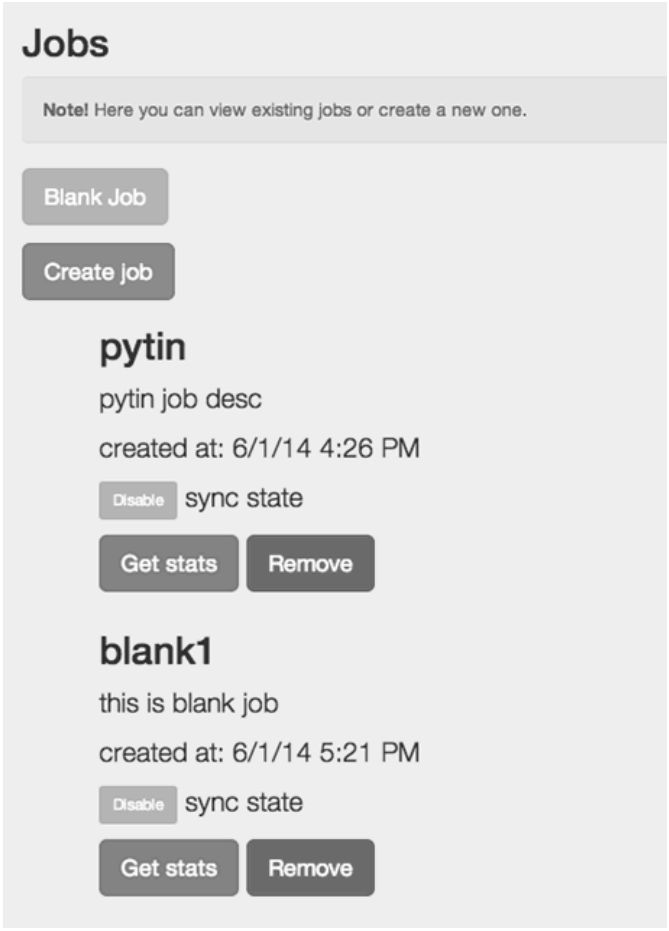

Рис. 6. Сторінка управління "Завданнями"

Сторінка статистики (рис. 7) містить статистичну інформацію про кількість позитивних і негативних висловлювань позначених як системою так і експертом. Додатково реалізовано можливість натиснути на певний сектор і перейти до сторінки з переліком висловлювань, відповідно позначених так як вказано в назві категорії.

Сторінка детальної інформації «завдання» містить посторінковий набір «висловлювань», що потрапили до даного «завдання» (рис. 8).

З короткою інформацією про автора висловлювання, текстом висловлювання, датою публікації, а також можливістю переглянути і встановити експертну оцінку для кожного висловлювання. Натиснувши на окреме висловлювання, відкривається джерело висловлювання в окремій сторінці.

Рис. 7. Сторінка статистики "Завдання"

Рис. 8. Сторінка детальної інформації «Завдання»

Висновки

Описано програмну реалізацію вирішення проблеми системного аналізу висловлювання користувачів соціальних мереж за допомогою статичного підходу у вигляді наївного баєсівського класифікатора. Ключовими поняттями підходу є взаємна інформація та правдоподібність висловлювань. Статистичний підхід, в подальшому, передбачив залучення думки експерта для встановлення об'єктивної оцінки висловлювань.

Для вирішення проблеми ефективної синхронізації та доступу до висловлювань користувачів соціальної мережі, використано розподілену платформу, що полегшила процес інтеграції потужних сервісів потокової комунікації соціальної мережі, забезпечила швидке опрацювання та надійне сховище даних, і надала тонкий інтерфейс для швидкого пошуку висловлювань за ключовими словами.

Розроблений прототип аналітичної системи надає зручний та компактний інструментарій керування завданнями пошуку та оцінки висловлювань користувачів соціальної мережі за ключовими словами. Під час планування архітектури системи, враховано потребу забезпечити швидкодію, а отже, введено певні обмеження під час синхронізаційних задач і розроблено власну чергу операцій. Первинний аналіз висловлювань показав доцільність потреби фільтрування суджень неінформативного характеру на ранній стадії, яка була вирішена комплексним підходом – мовним, сміттєвим та геолокаційним фільтрами.

Процес самонавчання прототипу організовано завдяки існуючим методам машинного навчання. Аналітична платформа навчена оцінювати нові висловлювання на основі попередніх статистичних показників. Ключовим моментом в забезпеченні об'єктивності оцінки аналітичної платформи виявилось пошук та формування набору характеристичних слів. Для кращої ефективності наївного оцінювання було розроблено критерій відбору характеристичних слів. Для покращення експертного навчання системи, було запропоновано ввести поняття фільтрування характеристичних слів.

В прототипі реалізовано можливість перегляду статистики конкретного завдання аналізу, що відображає кількісну характеристику системної та експертної оцінок висловлювань користувачів з можливістю їх сегментованого перегляду.

Варто додати, що вирішення питань про уподобання користувачів потребує подальших досліджень та вдосконалень. Для покращень прототипу необхідно реалізувати нові можливості експертного навчання і аналізу висловлювання, а саме: враховувати глобальні характеристичні слова, що дозволить зменшити вагу експерта для щойно створених завдань; надати експерту можливість впливати на характеристичні слова; реалізувати повний мовний фільтр.

Загальні вимоги до впровадження прототипу потребують інтеграцію інших популярних соціальних мереж, а також проведення додаткової дослідницької роботи і впровадження низки цікавих статистичних відображень.

1. *Глибовець М.М., Жигмановський А.А., Заболотний Р.І., Захоженко П.О.* Веб сервіси оброблення документів. – М.: – К.: НаУКМА, 2012. – 212 с.
2. *Salvatore Catanese, Pasquale De Meo, Emilio Ferrara, Giacomo Fiumara, Alessandro Provetti* Extraction and Analysis of Facebook Friendship Relations. www.**emilio**.**ferrara**.name/wp-content/uploads/2011/06/SN-76.pdf
3. *Grails.* What is Grails? [Електронний ресурс] // Grails: [сайт]. [2014]. URL: https://grails.org/learn
4. *Bootstrap:* [сайт]. [2014]. URL: http://getbootstrap.com/
5. *JetStrap.* The Bootstrap 3 builder [Електронний ресурс] // Jetstrap: [сайт]. [2014]. URL: https://jetstrap.com/
6. *Spring* Security [Електронний ресурс] // Spring IO: [сайт]. [2014]. URL: http://projects.spring.io/spring-security/

Одержано 09.04.2014

Про авторів:

Глибовець Андрій Миколайович,
кандидат фізико-математичних наук,
доцент кафедри мережних технологій,

Гонтар Максим Андрійович,
магістр,
розробник в компанії GlobalLogic.

Місце роботи авторів:

Національний університет
«Києво-Могилянська академія»,
04655, Київ,
вул. Г. Сковороди, 2.
Тел. (044) 425 0245.
E-mail: andriy@glybovets.com.ua

GlobalLogic Україна
03680, Київ,
вул. Миколи Грінченка 2/1.
Тел.: (044) 492 9695.
E-mail: maksym.hontar@gmail.com