**פרוייקט גמר**

StarTrace

**מחברים**: נטע לי להט, גל בן יאיר, יונתן אברהמי ועמית וייל

**מוגש לבית הספר למדעי המחשב**

**המסלול האדמאי, המכללה למנהל**

**כחלק מהדרישות האקדמאיות**

**לתואר B.Sc. במדעי המחשב**

אב התשע"ה ראשל"צ יולי 2015

**פרוייקט גמר**

StarTrace

**מחברים**: נטע לי להט, גל בן יאיר, יונתן אברהמי ועמית וייל

**המנחה המאשר**: ד"ר משה בוטמן

**בית הספר למדעי המחשב**

**המסלול האדמאי, המכללה למנהל**

אב התשע"ה ראשל"צ יולי 2015

**תוכן העניינים**

[תקציר - 4 -](#_Toc425803290)

[הקדמה - 5 -](#_Toc425803291)

[הגדרת עולם הבעיה - 5 -](#_Toc425803292)

[הגדרת הבעיה ומה נעשה בתחום - 6 -](#_Toc425803303)

[פתרון - הסבר על הפתרון המוצע וכיצד שונה מאחרים - 7 -](#_Toc425803304)

[הצדקה אקדמאית - 7 -](#_Toc425803305)

[מטרת הפרויקט - 8 -](#_Toc425803306)

[תרשים בלוקים\מודולים כללי של המערכת - 8 -](#_Toc425803308)

[DR - 8 -](#_Toc425803309)

[SRS - 9 -](#_Toc425803315)

[מתודולוגיות וכלי פיתוח, חומרה ותוכנה - 11 -](#_Toc425803316)

[המערכת - כולל תמונות מסך של קלט ופלט המערכת - 14 -](#_Toc425803330)

[משימות על פי זמנים והסבר על עמידה בזמנים - 19 -](#_Toc425803351)

[קשיים במהלך הפרויקט וכיצד נפתרו - 19 -](#_Toc425803360)

[ביבליוגרפיה - 20 -](#_Toc425803362)

תקציר

בעידן הרשתות החברתיות וכמויות המידע העצומות הקיימות ברשת ניתן לצבור מידע רב היכול לשמש להצלחת שיווק ופרסום בעולם כולו.

לפני תחילת מסע פרסום ניתן להשתמש במידע זה ככלי על מנת לדעת מה דעת הרוב על מפורסם בעיר מסוימת ובכך למנוע כישלון ובזבוז כספים על פרסום לא מוצלח.

StarTrace תעזור לבחור במפורסם המתאים ביותר למסע פרסום במיקום מסוים (עיר או מדינה) בעולם בהתבסס על ניתוח סנטימנטלי של תגובות משתמשים על אותו המפורסם ברשתות חברתיות.

בממשק למשתמש יש להזין שם של מפורסם ועיר ולאחר חישוב יוצגו למשתמש אחוזי הדעות החיוביות, השליליות והניטרליות על אותו המפורסם בעיר שנבחרה.

ממשק ה- WEB ימומש ע"י טכונלוגית AngularJS ביחד עם חבילת עיצוב Bootstrap, ע"מ ליצור מראה פשוט ונקי. שרת האפליקציה ישלח בקשות אל שרת מכונת הלמידה, אשר תנתח אותן לפי אלגוריתמי הניתוח הסנטימנטלי, ויקבל חזרה את תוצאות הניתוח. מכונת הלמידה תאסוף את המידע באמצעות הממשקים השונים שמציעות הרשתות החברתיות השונות (טוויטר, פייסבוק, אינסטגרם וכו'). הקוד של מכונת הלמידה ימומש בPython 2.7

הקדמה

כיום ברשת האינטרנט קיים מידע רב אודות מפורסמים ומעריציהם. המידע אודות המפורסמים מגיע בכמויות גדולות וממקורות שונים ומגוונים. לאיסוף ועיבוד המידע פוטנציאל עסקי רב.

הגדרת עולם הבעיה

כיום כל החברות בעולם אשר מפרסמות מוצרים לציבור הרחב צריכה מפורסם שיעמוד בראשה וייצג אותה כלפי הציבור עבור שלטי חוצות\קטלוגים\פרסומות וכדומה.

בלי פרסום זה וקמפיינים לשיווק המוצר, החברה והמוצר כנראה לא יממשו את פוטנציאל השיווק שיש להם.

מעבר לכך שיש צורך בשיווק נכון, הבחירה בפרזנטור שיהיה פני החברה עבור שיווק ופרסומים אלו היא לא פחות חשובה.

ישנם כמה סוגים של פרזנטורים:

1. פרזנטור מומחה – מקצועי בתחום הנמכר ונותן תחושה מקצועית וטובה לקונים מכיוון שמבין בתחום.
2. פרזנטור צרכן – ממליץ על המוצר בעקבות התנסות
3. ידוען\סלבריטי – שיכול לייצר אצל חלק ממעריציו גם תחושה טובה לגבי המוצר.

כאשר נרצה לבחור מפורסם להיות הפרזנטור של חברה – איך נדע במי לבחור?

כל בן אדם, כל תרבות וכל סביבה תעריץ סוגים שונים של מפורסמים כך שיכול להיווצר מצב שבסביבה מסוימת מפורסם אהוב נורא ובסביבה אחרת שנוא.

סוגיה זו יוצרת בעיה עבור חברות הפרסום שצריכות לבחור פרזנטור שיגרום לכמה שיותר אנשים ללכת אחריו ולקנות את המוצר.

הגדרת הבעיה ומה נעשה בתחום -

עד כה בחינה כמותית ומעמיקה של אפקטיביות פרסומות והשוואת ההשפעה של פרזנטור כזה או אחר בקטגוריה כזו או אחרת הייתה משימה כמעט בלתי אפשרית. ניתן היה לערוך בחינה שכזו בשתי שיטות שלכל אחת מהם חסרונות משלו. הראשונה, הצגת פרסומות בפני קבוצות מיקוד, מתודולוגיה אשר לא נתנה תשובות כמותיות מובהקות לסוגיות והשנייה באמצעות הצגת פרסומות בפני עוברים ושווים במרכזי קניות, מתודולוגיה שלא אפשרה להגיע למדגם כלל ארצי מייצג.

כיום, באמצעות הפאנל האינטרנטי של חברת פאנלס, ניתן לראשונה לבחון סוגיות הנוגעות לפרסומות על פני מדגם ארצי מייצג.

בסקר שערך מכון המחקר פאנלס, עבור אגוד חברות הפרסום ובית הספר של חברות הפרסום, הוצגו בפני כ- 451 נשאלים מגוון פרסומות מ- 5 קטגוריות כאשר כל פרסומת כוללת סוג שונה של פרזנטור: ."סלב", מומחה או צרכן (סה"כ 15 פרסומות).

על כל פרסומת נשאלו מספר שאלות בנוגע ל: האהדה לפרסומת, תפיסת האמינות שלה, העניין והסקרנות שהיא יצרה לגבי המוצר/השירות המפורסם, התרומה של הפרסומת להעדפת המוצר/השירות המפורסם.

מן הסקר ניתן לראות כאשר מחפשים ליצור **אהדה** למוצר מסוים, פרזנטור שהוא "סלב" מוביל בכל הקטגוריות. כלומר, על מנת להפוך את המוצר לפופולארי, יש להשיג פרזנטור שהוא "סלב".

אך כיצד ניתן לדעת מי הוא הסלב המתאים ביותר לשיווק המוצר אותו אנו רוצים להפוך לפופולארי?

נראה שאין דרך פשוטה למצוא מי מתאים מכיוון שאין את הכלים המתאימים למצוא איזה "סלב" הוא האהוד ביותר בסביבה מסוימת. ניתן להסתמך על דברים כמו רמת חשיפה או הופעות במדורי הרכילות, אך לעיתים דברים אלה עלולים להתפספס ובנוסף חסר מיקוד עבור ה"סלב" ברמת המיקרו, איפה הוא יותר פופולארי? בקרב אילו גילאים? האם יותר פופולארי בקרב נשים או גברים?

פתרון נוסף שקיים בעולם הדיגיטלי הוא Google Trends ו-Facebook Insights.

Google Trends יכול לתת מידע על כמה מחפשים ברחבי הרשת נושא מסוים או אדם מסוים ובאילו מקומות. Facebook Insights מציע לבעלי עמוד מסוים לקבל דוחות על כמות התנועה בעמוד שלהם ובכך יכול לספר לבעלי עמוד של "סלב" מסוים מהי רמת הפופולאריות שלו.

פתרון - הסבר על הפתרון המוצע וכיצד שונה מאחרים

מתוך שלושת הסוגים של פרזנטורים שדיברנו עליהם (צרכן\מומחה\ידוען), StarTrace באה לתת פתרון לסוג השלישי- בחינת אהדת הידוענים.

ניתן גם לתת פתרון עבור שני הסוגים הראשונים במידה והם גם דמויות הידועות לציבור.

מערכת StarTrace בונה על חוכמת ההמונים, ומחפשת שם את התשובה לשאלה שנשאלה בסוף הסעיף הקודם, מהי רמת הפופולאריות של "סלב" מסוים? היא עושה זאת באמצעות פניה אל הרשתות החברתיות בכדי להבין את רמת האהדה שלהם בקרב הציבור. באמצעות ניתוח סטאטוסים, Tweetים, תגובות ודיווחים על אותו ה"סלב", נוכל ללמוד מהי רמת האהדה שלו ברחבי הרשת. את רמת האהדה נוכל לאחר מכן להציע לחברות פרסום שונות אשר מחפשות את הפרזנטור הבא עבור המוצר שלהם. את החיפוש שלנו ניתן לבצע לפי מיקום ורשת חברתית ובהמשך תינתן האופציה לחפש גם לפי גילאים ומין.

הפתרון שלנו שונה מזה של Google Trends בכך שהוא יכול לסווג את ה"רעש" שהאדם אותו מחפשים עושה ברשת, בעוד ש-Google Trends בסך הכל יגיד **כמה** מחפשים אותו. מדדי הסיווג שלנו הם "חיובי", "שלילי" ו-"נייטרלי". בנוסף, אנו שונים מ-Facebook Insights בכך שאנחנו מאפשרים גם לאנשים שהם אינם מנהלי העמוד של ה"סלב"(ובכך למעשה מקורבים ברמה כלשהי אליו) לדעת מהי חוות הדעת עליו ברחבי העולם.

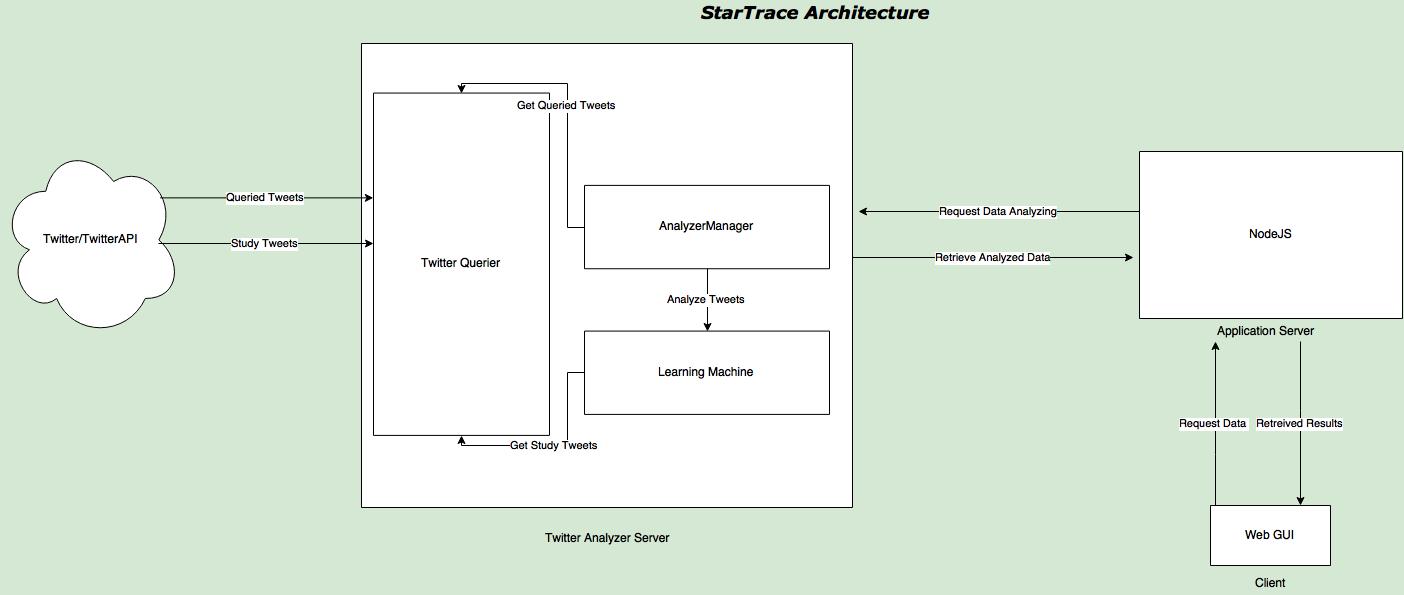
הצדקה אקדמאית

כדי להסיק תובנות בעלות פוטנציאל עסקי מתוך מאגרי הנתונים הנאספים מרשתות חברתיות על מפורסמים השתמשנו בדברים שלמדנו בקורס למידה חישובית וכריית נתונים.

את עולם כריית המידע ניתן לחלק לסוגי בעיות למידה שונות, במקרה שלנו זוהי בעיית סיווג. השתמשנו באחת הדרכים המומלצות לפתרון בעיות למידה מסוג זה, דרך זו בדרך כלל מגיעה לתוצאות סיווג גבוהות ומוצלחות מאוד. המסווג בו השתמשנו הוא Naive Bayes classifier, מסווג בייסיאני נאיבי הוא ממשפחת מסווגים הסתברותיים פשוטים המבוססים על יישום חוק בייס עם השערות בלתי תלויות חזקות (נאיביות) בין התכונות. המסווג לומד מסט דוגמאות שכבר תויגו (קיבלו ערך סיווג) ועל פי תכונותיהן שלמד לכל תיוג הוא יסווג את סט הדוגמאות הלא מתויגות.

מטרת הפרויקט

מטרת הפרויקט היא מציאת תובנות עסקיות בעזרת אלגוריתמי למידה על בסיס מידע רב שנאסף מרחבי האינטרנט. כדי לקבל החלטות עסקיות בתחום הפרסום נדרש להשתמש במידע של דעת הקהל הרחב על מפורסמים כדי שתהליך הפרסום יהיה מוצלח. נבנה מערכת שמספקת ידע לגבי דעת הקהל על מפורסמים על פי ערים בעולם על בסיס מידע.

תרשים בלוקים\מודולים כללי של המערכת

DR

כאשר באנו לפתח את המערכת חשבנו כיצד ניתן לכתוב מימוש פשוט וגמיש עבור האפליקציה שלנו, בנוסף רצינו לעבוד בטכנולוגיה אשר קיימות בה כבר ספריות הממשות שיטות סיווג וממשקים נוחים לרשתות החברתיות. בסופו של דבר ראינו שהטכנולוגיה הכי מתאימה לדרישות אלו היא Python בשל הקיום של ספריות שנוכל להיעזר בהן על מנת לסווג את המידע וכמובן מימושים איכותיים לממשקים מול הרשתות החברתיות.

בשלב הבא גיבשנו את הארכיטקטורה של המערכת. החלטנו שיהיה שרת ייעודי המוקדש ללמידה. שרת זה ימשוך נתונים מהרשתות החברתיות (מקור המידע החיצוני שלנו), יבצע עיבוד של המידע וישמור אותו אצלו. על שרת זה להיות מאופיין בדברים הבאים:

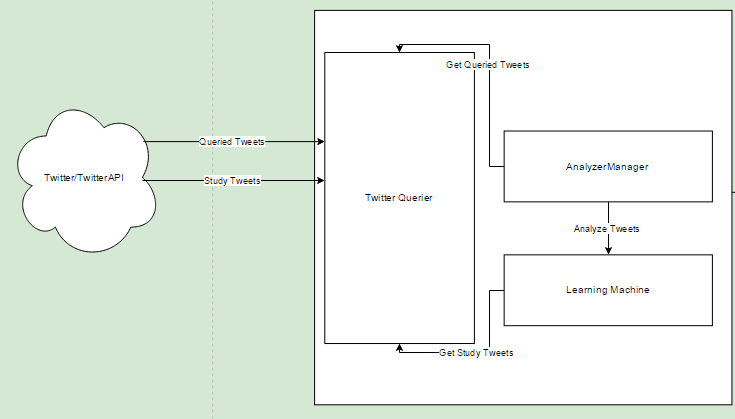
* מפרט טכני חזק. ביצוע פעולות למידה עשוי להיות פעולה מאוד כבדה, במיוחד כאשר מגיעות בקשות ממספר לקוחות שונים, לכן שרת זה זקוק להרבה זיכרון פנימי (RAM) ומעבד רב ליבות על מנת לעמוד במשימה.
* ייחצון ממשק RESTful – החלטנו שבכדי להפריד בצורה טובה בין השרת ללקוח, על השרת לייחצן ממשק RESTful אשר מאפשר ללקוח לשלוח בקשת POST מעל HTTP על מנת לקבל חזרה נתונים. כך למעשה שרת הלמידה לא צריך לדעת מי הלקוחות שלו, ויהיה ניתן לממש את צד הלקוח בטכנולוגיות שונות (Web, Mobile, Rich Client etc.)

לאחר מכן ניגשנו למימוש צד הלקוח. החלטנו לממש את צד הלקוח בשלב הראשון כשרת WEB אשר מכיל אתר המאפשר לבצע חיפוש על הסלברטיס המעניינים את המשתמש. הממשק בצד הלקוח הוא פשוט – הזנת שם המפוסם, הזנת המיקום ולחיצה על כפתור החיפוש (GO). בקשת החיפוש עוברת מהלקוח אל שרת ה-Web ומשם אל צד השרת.

התוצאות המוחזרות מוצגות בדף הראשי באופן דינאמי.

בחרנו בעיצוב קליל, פשוט ונח, ע"מ לא להלאות את המשתמש בפרטים מיותרים ולהתמקד במטרת החיפוש.

SRS

1. תרשים תהליכים ופעילויות
   1. צד שרת:
   2. צד לקוח: רץ על גבי שרת Node.js

The server processing the request

The server returns JSON to the **AngularJS Service**

HTML Page – **AngularJS Controller**  
Client Fill the Search Form

Results are shown dynamicaly by the returned JSON object

Query passed to the **AngularJS Service**. The service sends HTTP request to the server side  
  
The **AngularJS Service** returns to the **AngularJS Controller** the returned JSON

ב.      פירוט הדרישות

a.       חיפוש ברשת חברתית

i.      המערכת תאסוף מידע מרשת חברתית.

ii.      נתוני האיסוף יהיו מהשבוע האחרון.

iii.      המערכת תאסוף Tweetים, פוסטים תגובות ו-Likeים.

iv.      המערכת תחפש מידע רק עבור שמות מפורסמים.

v.      המערכת תחפש את המידע עבור מפורסמים בעיר מסוימת.

b.      סיווג המידע, למידה

i.      המערכת תיצור מאגר למידה.

ii.      בעזרת מאגר הלמידה, המערכת תיצור וקטורי סיווג.

iii.      וקטורי הסיווג יהיו עבור הנתונים שנאספו מהרשת החברתית.

iv.      כל Tweet וכדומה יסווג ע"פ מאגר הלמידה ווקטורי הסיווג.

v.      אלגוריתם הלמידה יהיה Navie Bayes.

c.       ניתוח המידע ו-יחצון התוצאות

i.      המערכת תנתח כל Tweet וכדומה לפי מאגר הלמידה.

ii.      המערכת תחליט ע"פ המאגר האם המידע הוא חיובי, שלילי או ניטרלי.

iii.      המערכת תחליט ע"פ המאגר האם הדעה הרווחת על מפורסם בעיר מסוימת היא חיובית, שלילית או ניטרלית.

iv.      המערכת תחזיר את תוצאות הניתוח כאחוזים.

v. המערכת תייחצן את התוצאות באמצעות ממשק RESTful

d.      הצגת המידע

i.      המערכת תציג את המידע בפשטות.

ii.      המערכת תציג את אחוזי ה-Tweetים החיוביים, שליליים וניטרליים מסך כל המידע שנאסף מהרשת החברתית, עבור כל מפורסם ומיקום.

iii.      המערכת תציג תמונה של המפורסם והעיר.

iv.      המערכת תציג את התוצאות גם בתור טבלת "עוגה".

מתודולוגיות וכלי פיתוח, חומרה ותוכנה

המערכת מחולקת לשני חלקים – צד שרת וצד לקוח.

1. חלק א' – צד שרת

צד השרת פותח לחלוטין ב-Python על מנת להשיג זריזות וגמישות בעבודה עם הממשקים. את הקוד אשר עוסק בלמידה פיתחנו באמצעות ספריות בשם SCM אשר מממשות מודלים של למידה חישובית. אל Twitter ניגשנו באמצעות ספריית python בשם TwitterAPI אשר יחצנה עבורנו את כל ממשקי החיפוש להם היינו זקוקים. על מנת להפוך מיקום משם של מקום לקואורדינציות השתמשנו בשירות רשת של GeoNames שמוחצן על ידי ספריית geopy, ספרייה שעוטפת ממשקי REST שונים ברחבים הרשת. את ממשק ה-REST פיתחנו באמצעות טכנולוגיה מבוססת Python בשם Flask, המאפשרת פיתוח אפליקציות אינטרנטיות פשוטות ומהירות. הבקשות הן בקשות POST אשר עוברות מעל HTTP.

1. חלק ב' – צד לקוח
   1. שרת ה-Web – Node.js

מה שמייחד שרתי אינטרנט הוא היכולת שלהם להתמודד עם מספר בקשות בו זמנית לאותו קוד. מספר גולשים למשל, מנסים להגיע בו זמנית לאותו הדף. במטרה לטפל בדרישה זאת שרתי האינטרנט משתמשים בתקשורת מבוססת threads. השרתים תומכים בריבוי threads ובדרך כלל לכל connection לשרת מוקצה thread. הבעיה בתקשורת זאת שהיא לא תמיד יעילה ודי קשה לשימוש, במיוחד שמבקשים לבצע גישה מהירה ומקבילית למשאבים ולקבצים: יכול להיות בזבוז זיכרון נרחב ויכולות להיות גם נעילות. בעיה זאת מחמירה ככל שהעומס על השרת גבוה יותר ונדרשות יכולות גבוהות מהמפתחים לבצע אופטימיזציות ושיפור ביצועים.  
לכן בחרנו לממש את שרת ה-Web עבור הפרוייקט ע"ג טכנולוגית Node.js, במטרה ליצור יכולת א-סינכרונית המאפשרת ביצועים גבוהים.  
node.js תוקפת ישירות את בעיה הנ"ל ומנסה לשחרר את צוואר הבקבוק מפעולות ה- I/O שחונקות את האפליקציה. אפליקציית node.js יכולה לטפל בסוגים שונים של בקשות I/O באותו הזמן משום שברירת המחדל של כל הפונקציות והספריות היא אסינכרונית. (יש גם מספר פונקציות סינכרוניות אבל הרוב המוחלט אסינכרוני).  
כל בקשות ה- I/O מהרשת מוגדרות כ- nonblocking וכל בקשות ה- I/O לקבצים מוגדרות אסינכרוניות. זאת אומרת, שהאפליקציה יכולה להמשיך לרוץ והגישה ל- I/O לא נחסמת.  
באמצעות שימוש ב- node ניתן להגיע לייעול מיטבי של הזיכרון תחת עומסים כבדים. מי שמשתמש ב- node לא צריך לחשוש מ- dead-locking של processes כיוון שאף פונקציה של node לא מבצעת בצורה ישירה פעולות I/O ולכן אין נעילות.  
  
שרת ה- Web בפרוייקט מקבל בקשות מהמשתמש, מעביר אותן לשרת ה-Phyton , המעבד את הנתונים שנשלחו אליו, ומחזיר אל שרת ה-Web את התוצאות בפורמט JSON.  
שרת ה-Web מחזיר את ה-JSON לצד הלקוח (AngularJS), והוא מציג את התוצאות.

* 1. AngularJS – צד הלקוח

Angularjs הינה פלטפורמה לפיתוח אפליקציות Web בצד הלקוח מבית היוצר של חברת גוגל, הפלטפורמה בנויה בשיטת MVC ובשונה מפלטפורמות פיתוח אחרות היא מסוגלת להשתלט על כל חלקי האפליקציה ולהכתיב את אופי העבודה כולו.  
Angularjs היא ספריית MVC הכוללת את המודלים Services, Controller ו – View.   
עקרון העבודה המנחה הוא לקיחת יותר החלטות עבור המשתמש דבר הבא לידי ביטוי במספר שורות הקוד הקצר יותר הנדרש.  
Angularjs שמה דגש על נושא בדיקתיות המערכת וניתן לכתוב באמצעותה תכניות בדיקות אינטגרציה ובדיקות מערכת בקלות יחסית.  
  
פלטפורמה זאת מאפשרת לנו, מפתחי הפרוייקט, להציג את תוצאות העיבוד בשרת בצורה נוחה ומאורגנת.

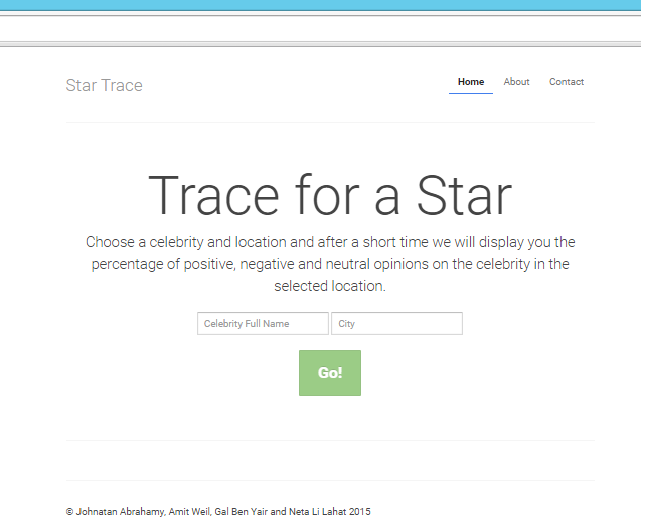
* + 1. פירוט המודולים בפרוייקט
       1. Controllers –  
          MainController – מודול זה אחראי לכל מה שמוצג בדף – כותרת, תיבות טקטס, הודעות למשתמש, תוצאות החיפוש, טבלאות ונתונים. דרכו מתבצעת שליחת הטקסט לחיפוש ל-Service.
       2. Services –   
          DataService – מודול זה אחראי לשלוח את את בקשת ה-http לשרת. מודול זה גם מקבל בחזרה את התוצאות (json) ומעביר ל-MainController.
       3. Views – התוכן הדינאמי המשתנה בהתאם למצב השימוש.  
          מצב הפתיחה – דף הפתיחה – בתוך הדף ישנו מקום לתוצאות החיפוש. כאשר תוצאות החיפוש מוחזרות, הן "נכתבות" לתוך הדף באופן דינאמי.  
          מצב ה"אודות" – דף האודות  
          מצב ה"צור קשר" – דף צור קשר
  1. BootStrap – ניראות האתר  
     FrameWork המספק סט בסיסי של כלי css לעיצוב אתרים הכולל ממשקים למשתמש, תפריטים, כפתורים, שדות וכו'. בנוסף, העיצוב הוא רספונסיבי.  
     FrameWork זה מכיל יישומי JS (בשילוב jQuery) לממשק משתמש.  
     מטרתו היא ליצור ממשק איכותי ולקדם אחידות בשימוש בכלים באתרים, לפשט את השימוש, ולהאיץ את תהליך הבנייה.

המערכת - כולל תמונות מסך של קלט ופלט המערכת

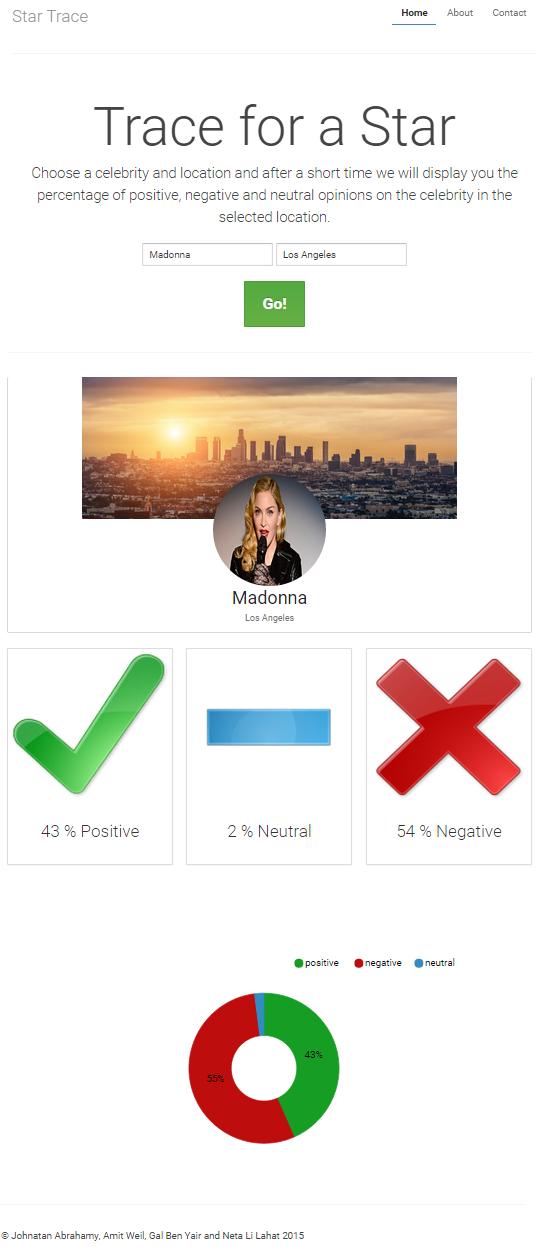
מסך הבית של המערכת:

התחברות המשתמש לאתר תביא אותו ישר לצפיהה בדף הHome.

* מאפשר הזנה של שם מפורסם ועיר בעולם עליהם ירצה המשתמש להריץ את סקירת התגובות.



לאחר הזנת הנתונים ולחיצה על "GO" מופעל האלגוריתם והמשתמש מקבל את תוצאות המערכת.

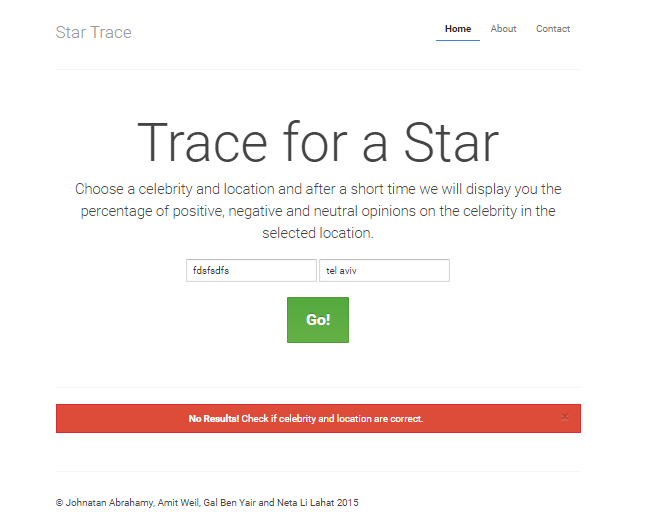


* מוצגת תמונה המפורסם ותמונה המקושרת לעיר שנבחרה
* התוצאות מתחלקות ל3 קטגוריות:

1. חיובי
2. שלילי
3. ניטרלי

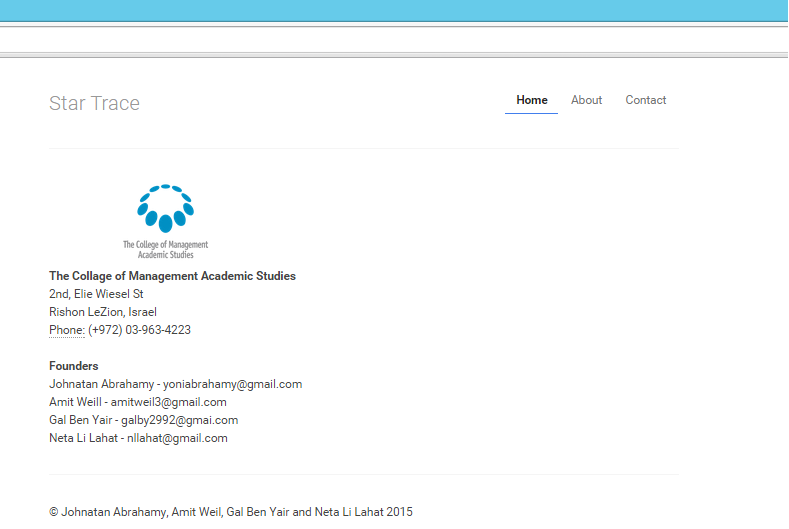
* ניתן לראות את התוצאות גם בצורת גרף.

במידה והמשתמש הזין נתונים (עיר\מפורסם) לא קיימים, המערכת לא תחזיר נתונים ותתקבל ההודעה הבאה:



מסך יצירת הקשר

בעת לחיצה על כפתור הcontact בתפריט המשתמש יגיע לדף יצירת הקשר הבא :



מסך הabout

מכיל פרטים כלליים על המוצר StarTrace .

מסך זה מציג גם את היכולת העסקית של המוצר עבור חברות שמחפשות מפורסמים.

משימות על פי זמנים והסבר על עמידה בזמנים

* בדה"ת לטכנולוגיות ודרכי איסוף נתונים מרשתות חברתיות שונות- ינואר 15
* בחירות טכנולוגיות ותחילת מימוש – פברואר 15
* פיתוח אופן הלמידה ובחינת פלט האלגוריתם – מרץ 15
* פיתוח ממשק למשתמש – אפריל 15
* הכנת תקצירים, ברושור ופוסטר לפרויקט - מאי 15
* הצגת הפרויקט – יוני 15
* ספר פרויקט – יולי 15

צורת העבודה שבה עבד הצוות למימוש הפרויקט הייתה חלוקת משימות בין חברי הצוות כדי שנעמוד בלוח הזמנים הנדרש לפרויקט, עמדנו בזמני העבודה שנדרשו.

קשיים במהלך הפרויקט וכיצד נפתרו

הקושי המרכזי במהלך הפרויקט היה למצוא API ידידותי אשר יאפשר לנו להוציא מידע על אנשים מרחבי הרשתות החברתיות. ראשית ניסינו לעבוד מול Facebook, אבל לאחר שגילינו שהם ביצעו שינויים ב-API שלהם שמונעים הוצאת מידע בסיסי על משתמשים החלטנו לעשות שינוי כיוון (Pivotting) ולהוציא מידע מ-Twitter במקום. כעת נתקלנו ב-API ידידותי יותר המאפשר לאחזר Tweetים ממשתמשים שונים ובנוסף לקבל Tweetים לפי מיקום. לאחר שאחזרנו את ה-Tweetים כל מה שנותר הוא לנתח אותם ולייחצן את התוצאת באמצעות ממשק REST מעל HTTP. קושי נוסף שנתקלנו בו הוא שה-API של Twiiter מאפשר להוציא רק מספר מסוים של Tweetים מתקופה שלא תעלה על חודש. פתרון שחשבנו על מנת לפתור את הבעיה הזו היא לשמור את הTweetים במסד נתונים משלנו, ובהמשך ליצור ממשק המבצע שילוב בין אחזור מידע מ-Twitter לבין עבודה מול מסד הנתונים שלנו על מנת להפיק ביצועי למידה טובים יותר ולהגיע לתוצאות איכותיות יותר.

ביבליוגרפיה

מאמר בנושא בעיית בחירת הפרזנטור - <http://www.allmarketing.co.il/?CategoryID=58&ArticleID=7500>

מאמר בנושא אלגוריתם למידה Navie Bayes וסיווג Text –

http://sebastianraschka.com/PDFs/articles/naive\_bayes\_1.pdf

מאמר בנושא ניתוח רשתות חברתיות –

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.372.1960&rep=rep1&type=pdf



# -------------------------------------------------------- #

# ======================== app.py ======================== #

# -------------------------------------------------------- #

from flask import Flask, jsonify, request, abort

from AnalyzingHandler import AnalyzingHandler

from ImageRetriever import ImageRetriever

app = Flask(\_\_name\_\_)

handler = handler = AnalyzingHandler()

@app.route('/learning/api/v1.0/analyzeTweets', methods=['POST'])

def analyzeTweets():

print "Entered function"

if not request.json or not 'keyword' in request.json:

abort(400)

#print "Passed if"

keyword = request.json['keyword']

location = request.json['location']

print "passed extraction"

results = handler.analyzeTweets(keyword, location=location)

imgRet = ImageRetriever()

results['keywordImages'] = imgRet.retreiveImage(keyword)

results['locationImages'] = imgRet.retreiveImage(location)

print results

return jsonify(results), 201

#end

#def index():

# return "Hello, World!"

#def setup\_app(app):

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

# -------------------------------------------------------- #

# ================= AnalyzingHandler.py ================== #

# -------------------------------------------------------- #

import collections

import argparse

import json

import datetime

import random

import os

import pickle

from datetime import timedelta

from TwitterQuerier import TwitterQuerier

from TwitterLearningMachine import TwitterLearningMachine

class AnalyzingHandler:

#start \_\_init\_\_

def \_\_init\_\_(self):

self.TwitterLM = TwitterLearningMachine('data/sampleTweets.csv')

#end

def analyzeTweets(self, keyword, location=None, until=None):

#Getting some tweets

querier = TwitterQuerier()

tweets = querier.getTweets(keyword, location=location)

# TODO: Call learning machine

results = results = {'positive':0, 'negative':0, 'neutral':0}

if len(tweets) > 0:

for tweet in tweets:

sentiment = self.TwitterLM.classify(tweet)

results[sentiment] = results[sentiment] + 1

# Normalizing the results:

for key in results:

results[key] = (results[key] \* 100) / len(tweets)

# TODO: Create a streamer that gets tweets and saves the to the database

return results;

#end LearnTweets

#end class

# Test Entry point

#analyzer = AnalyzingHandler()

#print analyzer.analyzeTweets('Beyonce', location='United States')

# -------------------------------------------------------- #

# =============== baseline\_classifier.py ================= #

# -------------------------------------------------------- #

import sys

import re

import classifier\_helper, html\_helper, pickle

reload(sys)

sys.setdefaultencoding = 'utf-8'

#start class

class BaselineClassifier:

""" Classifier using baseline method """

#variables

#start \_\_init\_\_

def \_\_init\_\_(self, data, keyword, time):

#Instantiate classifier helper

self.helper = classifier\_helper.ClassifierHelper('data/feature\_list.txt')

#Remove duplicates

self.lenTweets = len(data)

self.origTweets = self.getUniqData(data)

self.tweets = self.getProcessedTweets(self.origTweets)

self.results = {}

self.neut\_count = [0] \* self.lenTweets

self.pos\_count = [0] \* self.lenTweets

self.neg\_count = [0] \* self.lenTweets

self.time = time

self.keyword = keyword

self.html = html\_helper.HTMLHelper()

#end

#start getUniqData

def getUniqData(self, data):

uniq\_data = {}

for i in data:

d = data[i]

u = []

for element in d:

if element not in u:

u.append(element)

#end inner loop

uniq\_data[i] = u

#end outer loop

return uniq\_data

#end

#start getProcessedTweets

def getProcessedTweets(self, data):

tweets = {}

for i in data:

d = data[i]

tw = []

for t in d:

tw.append(self.helper.process\_tweet(t))

tweets[i] = tw

#end loop

return tweets

#start classify

def classify(self):

#load positive keywords file

inpfile = open("data/pos\_mod.txt", "r")

line = inpfile.readline()

positive\_words = []

while line:

positive\_words.append(line.strip())

line = inpfile.readline()

#load negative keywords file

inpfile = open("data/neg\_mod.txt", "r")

line = inpfile.readline()

negative\_words = []

while line:

negative\_words.append(line.strip())

line = inpfile.readline()

#start processing each tweet

for i in self.tweets:

tw = self.tweets[i]

count = 0

res = {}

for t in tw:

neg\_words = [word for word in negative\_words if(self.string\_found(word, t))]

pos\_words = [word for word in positive\_words if(self.string\_found(word, t))]

if(len(pos\_words) > len(neg\_words)):

label = 'positive'

self.pos\_count[i] += 1

elif(len(pos\_words) < len(neg\_words)):

label = 'negative'

self.neg\_count[i] += 1

else:

if(len(pos\_words) > 0 and len(neg\_words) > 0):

label = 'positive'

self.pos\_count[i] += 1

else:

label = 'neutral'

self.neut\_count[i] += 1

result = {'text': t, 'tweet': self.origTweets[i][count], 'label': label}

res[count] = result

count += 1

#end inner loop

self.results[i] = res

#end outer loop

filename = 'data/results\_lastweek.pickle'

outfile = open(filename, 'wb')

pickle.dump(self.results, outfile)

outfile.close()

'''

inpfile = open('data/results\_lastweek.pickle')

self.results = pickle.load(inpfile)

inpfile.close()

'''

#end

#start substring whole word match

def string\_found(self, string1, string2):

if re.search(r"\b" + re.escape(string1) + r"\b", string2):

return True

return False

#end

#start writeOutput

def writeOutput(self, filename, writeOption='w'):

fp = open(filename, writeOption)

for i in self.results:

res = self.results[i]

for j in res:

item = res[j]

text = item['text'].strip()

label = item['label']

writeStr = text+" | "+label+"\n"

fp.write(writeStr)

#end inner loop

#end outer loop

#end writeOutput

#start printStats

def getHTML(self):

return self.html.getResultHTML(self.keyword, self.results, self.time, self.pos\_count, \

self.neg\_count, self.neut\_count, 'baseline')

#end

#end class

# -------------------------------------------------------- #

# =============== build\_training\_set.py ================== #

# -------------------------------------------------------- #

import classifier\_helper

from classifier\_helper import \*

'''

inpfile = open("training.1600000.processed.noemoticon.csv", "r")

line = inpfile.readline()

maxCount = 100

count = 1

tweets = []

while line:

count += 1

if count > maxCount:

break

splitArr = line.split(',"')

unprocessed\_tweet = splitArr[5]

#tweet = process\_tweet(unprocessed\_tweet)

tweet = process\_tweet\_modified(unprocessed\_tweet)

tweets.append(tweet)

line = inpfile.readline()

#end while loop

'''

inpfile = open("baseline\_output.txt", "r")

line = inpfile.readline()

count = 1

tweetItems = []

opinions = []

while line:

count += 1

splitArr = line.split('|')

processed\_tweet = splitArr[0].strip()

opinion = splitArr[1].strip()

tweet\_item = processed\_tweet, opinion

if(opinion != 'neutral' and opinion != 'negative' and opinion != 'positive'):

print('Error with tweet = %s, Line = %s') % (processed\_tweet, count)

tweetItems.append(tweet\_item)

line = inpfile.readline()

#end while loop

tweets = []

for (words, sentiment) in tweetItems:

words\_filtered = [e.lower() for e in words.split() if len(e) >= 3]

tweets.append((words\_filtered, sentiment))

word\_features = get\_word\_features(get\_words\_in\_tweets(tweets))

set\_word\_features(word\_features)

training\_set = nltk.classify.apply\_features(extract\_features, tweets)

classifier = nltk.NaiveBayesClassifier.train(training\_set)

tweet = 'im so sad'

print classifier.classify(extract\_features(tweet.split()))

print nltk.classify.accuracy(classifier, training\_set)

classifier.show\_most\_informative\_features(20)

# -------------------------------------------------------- #

# ==================== config.json ======================= #

# -------------------------------------------------------- #

{

"consumer\_key": "ur0KdmqZYB1tYFSRMc5d7buP8",

"consumer\_secret": "ep5DHkG9SliOjNLDBHQgcGjoEatKoSZsYoYCzcNw4p8FX4nnJc",

"access\_token": "2920976197-qN8WgFwX9ZfZaJXofzp4of1melrUyg0ukIHXgCx",

"access\_token\_secret": "wYDcRKcjPW95eb8qW4cn9Poyk8kkam5xihOQguJSFh1LB"

}

# -------------------------------------------------------- #

# ================= ImageRetriever.py ==================== #

# -------------------------------------------------------- #

import collections

import argparse

import json

import datetime

import os

from datetime import timedelta

import urllib

import urllib2

import socket

class ImageRetriever:

#start \_\_init\_\_

#def \_\_init\_\_(self):

#end

def retreiveImage(self, keyword):

# Creating a dictionary for the url

data = {'v': '1.0', 'q' : keyword, 'userip' : socket.gethostbyname(socket.gethostname())}

queryString = urllib.urlencode(data);

#Getting some images

url = ('https://ajax.googleapis.com/ajax/services/search/images?' + queryString)

request = urllib2.Request(url, None, {'Referer1': 'www.startrace.com'})

response = urllib2.urlopen(request)

# Process the JSON string.

results = json.load(response)

# now have some fun with the results...

#

PhotoUrls = [];

rawPhotos = results['responseData']['results']

for photo in rawPhotos:

PhotoUrls.append(photo['unescapedUrl'])

return PhotoUrls

#end

#end class

# Test Entry point

#imager = ImageRetriever();

#print imager.retreiveImage('Brad Pitt')

#print results

#print results["responseData"]["results"][0]['unescapedUrl']

# -------------------------------------------------------- #

# ============== naive\_bayes\_classifier.py =============== #

# -------------------------------------------------------- #

import nltk.classify

import re, pickle, csv, os

import classifier\_helper, html\_helper

#start class

class NaiveBayesClassifier:

""" Naive Bayes Classifier """

#variables

#start \_\_init\_\_

def \_\_init\_\_(self, data, keyword, time, trainingDataFile, classifierDumpFile, trainingRequired = 0):

#Instantiate classifier helper

self.helper = classifier\_helper.ClassifierHelper('data/feature\_list.txt')

self.lenTweets = len(data)

self.origTweets = self.getUniqData(data)

self.tweets = self.getProcessedTweets(self.origTweets)

self.results = {}

self.neut\_count = [0] \* self.lenTweets

self.pos\_count = [0] \* self.lenTweets

self.neg\_count = [0] \* self.lenTweets

self.trainingDataFile = trainingDataFile

self.time = time

self.keyword = keyword

self.html = html\_helper.HTMLHelper()

#call training model

if(trainingRequired):

self.classifier = self.getNBTrainedClassifer(trainingDataFile, classifierDumpFile)

else:

f1 = open(classifierDumpFile)

if(f1):

self.classifier = pickle.load(f1)

f1.close()

else:

self.classifier = self.getNBTrainedClassifer(trainingDataFile, classifierDumpFile)

#end

#start getUniqData

def getUniqData(self, data):

uniq\_data = {}

for i in data:

d = data[i]

u = []

for element in d:

if element not in u:

u.append(element)

#end inner loop

uniq\_data[i] = u

#end outer loop

return uniq\_data

#end

#start getProcessedTweets

def getProcessedTweets(self, data):

tweets = {}

for i in data:

d = data[i]

tw = []

for t in d:

tw.append(self.helper.process\_tweet(t))

tweets[i] = tw

#end loop

return tweets

#end

#start getNBTrainedClassifier

def getNBTrainedClassifer(self, trainingDataFile, classifierDumpFile):

# read all tweets and labels

tweetItems = self.getFilteredTrainingData(trainingDataFile)

tweets = []

for (words, sentiment) in tweetItems:

words\_filtered = [e.lower() for e in words.split() if(self.helper.is\_ascii(e))]

tweets.append((words\_filtered, sentiment))

training\_set = nltk.classify.apply\_features(self.helper.extract\_features, tweets)

# Write back classifier and word features to a file

classifier = nltk.NaiveBayesClassifier.train(training\_set)

outfile = open(classifierDumpFile, 'wb')

pickle.dump(classifier, outfile)

outfile.close()

return classifier

#end

#start getFilteredTrainingData

def getFilteredTrainingData(self, trainingDataFile):

fp = open( trainingDataFile, 'rb' )

min\_count = self.getMinCount(trainingDataFile)

min\_count = 40000

neg\_count, pos\_count, neut\_count = 0, 0, 0

reader = csv.reader( fp, delimiter=',', quotechar='"', escapechar='\\' )

tweetItems = []

count = 1

for row in reader:

processed\_tweet = self.helper.process\_tweet(row[1])

sentiment = row[0]

if(sentiment == 'neutral'):

if(neut\_count == int(min\_count)):

continue

neut\_count += 1

elif(sentiment == 'positive'):

if(pos\_count == min\_count):

continue

pos\_count += 1

elif(sentiment == 'negative'):

if(neg\_count == min\_count):

continue

neg\_count += 1

tweet\_item = processed\_tweet, sentiment

tweetItems.append(tweet\_item)

count +=1

#end loop

return tweetItems

#end

#start getMinCount

def getMinCount(self, trainingDataFile):

fp = open( trainingDataFile, 'rb' )

reader = csv.reader( fp, delimiter=',', quotechar='"', escapechar='\\' )

neg\_count, pos\_count, neut\_count = 0, 0, 0

for row in reader:

sentiment = row[0]

if(sentiment == 'neutral'):

neut\_count += 1

elif(sentiment == 'positive'):

pos\_count += 1

elif(sentiment == 'negative'):

neg\_count += 1

#end loop

return min(neg\_count, pos\_count, neut\_count)

#end

#start classify

def classify(self):

for i in self.tweets:

tw = self.tweets[i]

count = 0

res = {}

for t in tw:

label = self.classifier.classify(self.helper.extract\_features(t.split()))

if(label == 'positive'):

self.pos\_count[i] += 1

elif(label == 'negative'):

self.neg\_count[i] += 1

elif(label == 'neutral'):

self.neut\_count[i] += 1

result = {'text': t, 'tweet': self.origTweets[i][count], 'label': label}

res[count] = result

count += 1

#end inner loop

self.results[i] = res

#end outer loop

#end

#start accuracy

def accuracy(self):

tweets = self.getFilteredTrainingData(self.trainingDataFile)

total = 0

correct = 0

wrong = 0

self.accuracy = 0.0

for (t, l) in tweets:

label = self.classifier.classify(self.helper.extract\_features(t.split()))

if(label == l):

correct+= 1

else:

wrong+= 1

total += 1

#end loop

self.accuracy = (float(correct)/total)\*100

print 'Total = %d, Correct = %d, Wrong = %d, Accuracy = %.2f' % \

(total, correct, wrong, self.accuracy)

#end

#start writeOutput

def writeOutput(self, filename, writeOption='w'):

fp = open(filename, writeOption)

for i in self.results:

res = self.results[i]

for j in res:

item = res[j]

text = item['text'].strip()

label = item['label']

writeStr = text+" | "+label+"\n"

fp.write(writeStr)

#end inner loop

#end outer loop

#end writeOutput

#start getHTML

def getHTML(self):

return self.html.getResultHTML(self.keyword, self.results, self.time, self.pos\_count, \

self.neg\_count, self.neut\_count, 'naivebayes')

#end

#end class

# -------------------------------------------------------- #

# ============== TwitterLearningMachine.py =============== #

# -------------------------------------------------------- #

import collections

import datetime

from sklearn import svm

from sklearn import \*

import nltk

from nltk.classify import \*

import re

import csv

import pprint

import numpy

import classifier\_helper

from classifier\_helper import \*

import pickle

import os.path

class TwitterLearningMachine:

#start \_\_init\_\_

def \_\_init\_\_(self, learningDataset):

#define machines

self.NBClassifier = None;

self.SVMClassifier = None;

#getting the stop words

self.stopWords = self.getStopWordList('data/feature\_list/stopwords.txt')

self.featureList = []

# init machines

self.initNBClassifier(learningDataset)

#end

#start replaceTwoOrMore

def replaceTwoOrMore(self, s):

#look for 2 or more repetitions of character

pattern = re.compile(r"(.)\1{1,}", re.DOTALL)

return pattern.sub(r"\1\1", s)

#end

#start process\_tweet

def processTweet(self, tweet):

# process the tweets

#Convert to lower case

tweet = tweet.lower()

#Convert www.\* or https?://\* to URL

tweet = re.sub('((www\.[^\s]+)|(https?://[^\s]+))','URL',tweet)

#Convert @username to AT\_USER

tweet = re.sub('@[^\s]+','AT\_USER',tweet)

#Remove additional white spaces

tweet = re.sub('[\s]+', ' ', tweet)

#Replace #word with word

tweet = re.sub(r'#([^\s]+)', r'\1', tweet)

#trim

tweet = tweet.strip('\'"')

return tweet

#end

#start getStopWordList

def getStopWordList(self, stopWordListFileName):

#read the stopwords

stopWords = []

stopWords.append('AT\_USER')

stopWords.append('URL')

fp = open(stopWordListFileName, 'r')

line = fp.readline()

while line:

word = line.strip()

stopWords.append(word)

line = fp.readline()

fp.close()

return stopWords

#end

#start getfeatureVector

def getFeatureVector(self, tweet, stopWords):

featureVector = []

words = tweet.split()

for w in words:

#replace two or more with two occurrences

w = self.replaceTwoOrMore(w)

#strip punctuation

w = w.strip('\'"?,.')

#check if it consists of only words

val = re.search(r"^[a-zA-Z][a-zA-Z0-9]\*[a-zA-Z]+[a-zA-Z0-9]\*$", w)

#ignore if it is a stopWord

if(w in stopWords or val is None):

continue

else:

featureVector.append(w.lower())

return featureVector

#end

#start extract\_features

def extract\_features(self, tweet):

tweet\_words = set(tweet)

features = {}

for word in self.featureList:

features['contains(%s)' % word] = (word in tweet\_words)

return features

#end

def initNBClassifier(self, learningDataset):

if os.path.isfile("NBClassifier.p"):

self.NBClassifier = pickle.load(open( "NBClassifier.p", "rb" ) )

print "Used ready!"

else:

inpTweets = csv.reader(open(learningDataset, 'rb'), delimiter=',', quotechar='"')

count = 0;

tweets = []

for row in inpTweets:

sentiment = row[0]

tweet = row[1]

processedTweet = self.processTweet(tweet)

featureVector = self.getFeatureVector(processedTweet, self.stopWords)

self.featureList.extend(featureVector)

tweets.append((featureVector, sentiment));

#end loop

# Remove featureList duplicates

self.featureList = list(set(self.featureList))

# Generate the training set

training\_set = nltk.classify.util.apply\_features(self.extract\_features, tweets)

# Train the Naive Bayes classifier

self.NBClassifier = nltk.NaiveBayesClassifier.train(training\_set)

# Saving the classifier

pickle.dump(self.NBClassifier, open("NBClassifier.p", "wb"))

print "Made A new one!"

def classify(self, tweet):

processedTweet = self.processTweet(tweet);

sentiment = self.NBClassifier.classify(self.extract\_features(self.getFeatureVector(processedTweet, self.stopWords)))

return sentiment.strip('|')

#end class

# Test area for learning machine

#LM = TwitterLearningMachine('data/medium\_size\_dataset.csv')

#tweet = 'Bananas'

#print LM.classify(tweet)

# -------------------------------------------------------- #

# ================= TwitterQuerier.py ==================== #

# -------------------------------------------------------- #

import collections

import argparse

import urllib

import urllib2

import json

import datetime

import random

import os

import pickle

from datetime import timedelta

from geopy.geocoders import GeoNames

from TwitterAPI import TwitterAPI, TwitterOAuth, TwitterRestPager

class TwitterQuerier:

#start \_\_init\_\_

#def \_\_init\_\_(self): // Remove if ctor is necessary

#end

#start getTweets

def getTweets(self, keyword, location=None , until=None):

params = {}

# Checking for params

if (until != None):

params['until'] = until

if (location != None):

geopoint = self.extractLocation(location)

# If no longtitude and latitude found, not including locations in search

if (geopoint != None):

params['geocode'] = "%f,%f,2000km" % (geopoint['latitude'], geopoint['longitude'])

return self.getData(keyword, params)

#end

#start extractLocation

def extractLocation(self, location):

geolocator = GeoNames(username='startrace')

geolocation = geolocator.geocode(location, timeout=10000)

geopoint = {}

if geolocation == None:

return None;

else:

geopoint['latitude'] = geolocation.latitude

geopoint['longitude'] = geolocation.longitude

return geopoint

#end

def parse\_config(self):

config = {}

# from file args

if os.path.exists('config.json'):

with open('config.json') as f:

config.update(json.load(f))

else:

# may be from command line

parser = argparse.ArgumentParser()

parser.add\_argument('-ck', '--consumer\_key', default=None, help='Your developper `Consumer Key`')

parser.add\_argument('-cs', '--consumer\_secret', default=None, help='Your developper `Consumer Secret`')

parser.add\_argument('-at', '--access\_token', default=None, help='A client `Access Token`')

parser.add\_argument('-ats', '--access\_token\_secret', default=None, help='A client `Access Token Secret`')

args\_ = parser.parse\_args()

def val(key):

return config.get(key)\

or getattr(args\_, key)\

or raw\_input('Your developper `%s`: ' % key)

config.update({

'consumer\_key': val('consumer\_key'),

'consumer\_secret': val('consumer\_secret'),

'access\_token': val('access\_token'),

'access\_token\_secret': val('access\_token\_secret'),

})

# should have something now

return config

def oauth\_req(self, url, http\_method="GET", post\_body=None,

http\_headers=None):

config = self.parse\_config()

consumer = oauth2.Consumer(key=config.get('consumer\_key'), secret=config.get('consumer\_secret'))

token = oauth2.Token(key=config.get('access\_token'), secret=config.get('access\_token\_secret'))

client = oauth2.Client(consumer, token)

resp, content = client.request(

url,

method=http\_method,

body=post\_body or '',

headers=http\_headers

)

return content

def getTweeterApi(self):

config = self.parse\_config();

api = TwitterAPI(config['consumer\_key'], #consumer\_key

config['consumer\_secret'], #consumer\_secret

config['access\_token'], #access\_token

config['access\_token\_secret']) #access\_token\_secret

return api

#end

def getData(self, keyword, params):

tweeter\_api = self.getTweeterApi()

#get the params

data = {'q': keyword, 'lang': 'en', 'count' : 150}

#Add if additional params are passed

if params:

for key, value in params.iteritems():

data[key] = value

results = tweeter\_api.request('search/tweets', data)

tweets = []

for item in results:

tweets.append(item['text']) # if 'text' in item else item)

return tweets

#end

#end class

# -------------------------------------------------------- #

# ====================== server.js ===================== #

# -------------------------------------------------------- #

// modules =================================================

var express = require('express');

var app = express();

var mongoose = require('mongoose');

var bodyParser = require('body-parser');

var methodOverride = require('method-override');

// configuration ===========================================

// config files

//var db = require('./config/db');

//mongoose.connect('mongodb://localhost/news');

var port = process.env.PORT || 8080; // set our port

// mongoose.connect(db.url); // connect to our mongoDB database (commented out after you enter in your own credentials)

// get all data/stuff of the body (POST) parameters

app.use(bodyParser.json()); // parse application/json

app.use(bodyParser.json({ type: 'application/vnd.api+json' })); // parse application/vnd.api+json as json

app.use(bodyParser.urlencoded({ extended: true })); // parse application/x-www-form-urlencoded

app.use(methodOverride('X-HTTP-Method-Override')); // override with the X-HTTP-Method-Override header in the request. simulate DELETE/PUT

app.use(express.static(\_\_dirname + '/public')); // set the static files location /public/img will be /img for users

// routes ==================================================

require('./app/routes')(app); // pass our application into our routes

// start app ===============================================

app.listen(port);

console.log('Magic happens on port ' + port); // shoutout to the user

exports = module.exports = app; // expose app

# -------------------------------------------------------- #

# ====================== MainCtrl.js ===================== #

# -------------------------------------------------------- #

angular.module('MainCtrl', []).controller('MainController', function($scope, Data) {

$scope.celebrity;

$scope.city;

$scope.getDataStatus = "beforeGetData";

$scope.isError = false;

$scope.dataObject = {}

$scope.getDataFromServer = function () {

$scope.isError = false;

//use dataService to get the json

$scope.getDataStatus = "getDataProccess";

$scope.dataObject = {}

//$scope.isError = true;

var reqObject = {

"keyword": $scope.celebrity,

"location":$scope.city

}

return (Data.query(JSON.stringify(reqObject))

.then (function (value) {

$scope.dataObject = value.data;

$scope.currCelebrity = JSON.parse(JSON.stringify($scope.celebrity));

$scope.currCity = JSON.parse(JSON.stringify($scope.city));

if ((($scope.dataObject.positive == 0) &&

($scope.dataObject.negative == 0) &&

($scope.dataObject.neutral == 0)) ||

(!($scope.dataObject))) {

$scope.isError = true;

}

else {

$scope.isError = false;

$scope.getDataStatus = "getDataSuccess";

// Pie chart

nv.addGraph(function() {

var chart = nv.models.pieChart()

.x(function(d) { return d.label })

.y(function(d) { return d.value })

.color(GetColors())

.labelThreshold(.05)

.showLabels(true)

.pieLabelsOutside(false)

.labelType("percent")

.donut(true)

.donutRatio(0.35);

d3.select("#chart svg")

.datum(InitData(value.data))

.transition().duration(1200)

.call(chart);

return chart;

});

}

},

function (error) {

console.log("error while getting data: " + error.textMessage);

$scope.dataObject = {};

$scope.isError = true;

$scope.getDataStatus = "getDataError"

}));

};

// define pie chart colors

function GetColors(dataObject) {

var colors = [

'rgb(22, 158, 36)', // positive

'rgb(190, 13, 13)', // negative

'rgb(48, 141, 193)' // neutral

];

return colors;

}

// define pie chart data

function InitData(dataObject) {

return [{"label" : "positive", "value" : dataObject.positive},

{"label" : "negative", "value" : dataObject.negative},

{"label" : "neutral", "value" : dataObject.neutral}];

}

/\*

$scope.dataObject = {

"positive":33,

"negative":33,

"neutral":33

}\*/

});

# -------------------------------------------------------- #

# ====================== DataService.js ===================== #

# -------------------------------------------------------- #

angular.module('DataService', []).factory('Data', ['$http','$q', function($http, $q) {

var data = undefined;

return {

query: function(jsonObj) {

return $http.post("http://localhost:5000/learning/api/v1.0/analyzeTweets", jsonObj);

}

}

}]);

# -------------------------------------------------------- #

# ====================== appRoutes.js ==================== #

# -------------------------------------------------------- #

angular.module('appRoutes', []).config(['$stateProvider', '$urlRouterProvider',

function($stateProvider, $urlRouterProvider) {

// For any unmatched url, send to /home

$urlRouterProvider.otherwise("/home");

$stateProvider

.state('home', {

url: "/home",

templateUrl: 'views/home.html',

controller: 'MainController'

})

.state('about', {

url: "/about",

templateUrl: 'views/about.html'

//controller: 'MController'

})

.state('contact', {

url: "/contact",

templateUrl: 'views/contact.html'

//controller: 'MController'

})

}]);