למידת מכונה - עבודת גמר

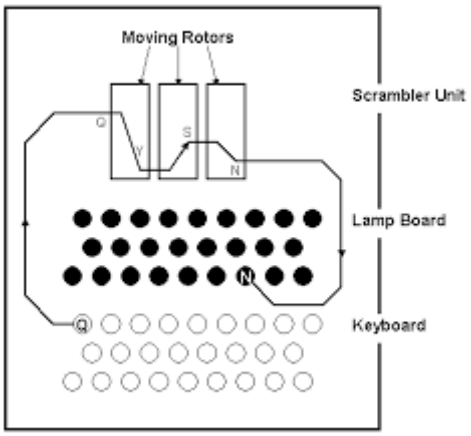
מגישים:
ספיר בוחבוט 316416429
אלמוג דוד 207749441

נושא הפרויקט: פיצוח צופן האניגמה בכלים מודרניים

מכונת האניגמה

מכונת האניגמה היא מכשיר הצפנה מפורסם בו השתמשו הגרמנים, בעיקר במהלך מלחמת העולם השנייה, כדי לאבטח תקשורת צבאית. המכונה דומה למכונת כתיבה ופועלת באמצעות הזנת אותיות דרך מקלדת, אשר מוצפנות לאחר מכן על סמך תצורתם של מספר סלילים ולוח חיבורים לפני שהן מוצגות כאות אחרת.

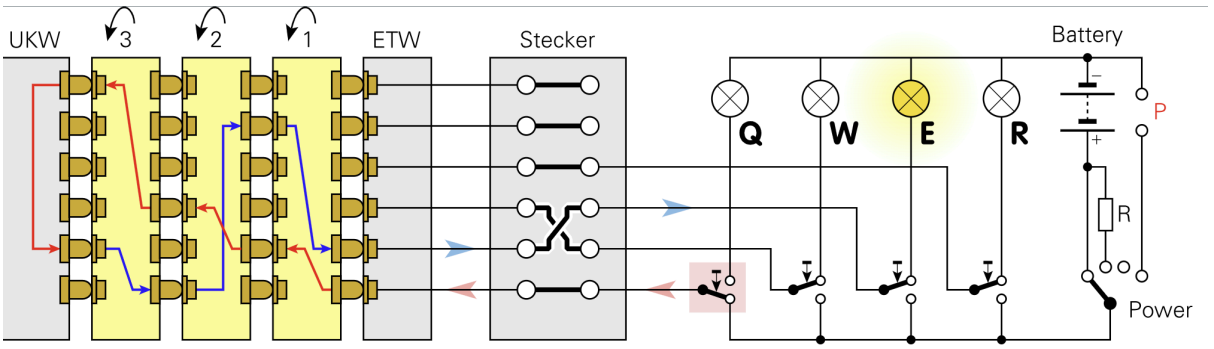
תצורה זו אפשרה מספר עצום של הגדרות אפשריות, מה שהפך את ההצפנה של האניגמה לקשה מאוד (וכמעט בלתי אפשרית) לפענוח.



מכונת האניגמה מורכבת משלושה חלקים מרכזיים:

- שלושה רוטורים (Rotors), יש חמישה רוטורים אפשריים, בוחרים שלושה מתוכם ובוחרים לשלושה הנ"ל את הסדר בו הם נכנסים למכונה ובנוסף את נקודת תחילת העבודה שלהם (יש לכל רוטור 26 נ"ק התחלה כמספר האותיות באנגלית).
- משקף (Reflector) שבעצם מחבר אות אחת לשניה - את הסדר ניתן לקבוע ולשנות אך תמיד חייב להיות שכל אות תהיה מחוברת לאות אחרת.
- לוח תקע (Plugboard) שבו ניתן לערבל את האותיות כמו שתרצה, כלומר הדבר אינו חובה אך אפשר לחבר שתי אותיות α ו-β כך שאם נכנס למכונה α היא בעצם תכניס אליה את β וההפך.

דוגמת ריצה: כאשר אות α הייתה נלחצת במכונה, האות הייתה עוברת דרך לוח השקעים (PB) - שם הייתה עוברת פרמוטציה לפי החיבורים בו ($PB(\alpha) = \beta$), לאחר מכן היא הייתה נכנסת לרוטור אחד ($R1$) שם הייתה עוברת פרמוטציה ($R1(\beta) = \beta'$) לאחר מכן הייתה נכנסת אל רוטור שתיים ($R2$) שם הייתה עוברת פרמוטציה ($R2(\beta') = \beta''$), באופן דומה היא תיכנס אל רוטור שלוש ($R3(\beta'') = \beta'''$).
לאחר מכן האות תיכנס אל המשקף (REF) שם היא תתחלף עם הזוג שהוצמד לה ($REF(\beta''') = \delta$), באופן דומה האות תחזור דרך שלושת הרוטורים $R1(R2(R3(\delta))) = \gamma$ ובסוף האות תעבור שוב דרך לוח השקעים ($PB(\gamma) = \mu$) לפני שתצא חזרה כ- μ.



כעת נחשב את מספר הקומבינציות האפשריות לקונפיגורציות התחלה אפשריות של המכונה:

- עבור הרוטורים (ישנם חמישה אפשריים, מתוכם צריך לבחור שלושה בסדר כלשהו):
$5 \cdot 4 \cdot 3 = 60$
- עבור נקודות ההתחלה של כל רוטור (כל רוטור מתחיל את הסיבוב שלו מנקודה מסוימת בין 0-25):
$26 \cdot 26 \cdot 26 = 26^3 = 17,576$
- עבור לוח החיבורים (Plugboard) ישנה אופציה לחבר בין זוג אותיות - או לא לחבר כלל ולכן:
$\frac{26!}{6!\,10!2^{10}} = 150,738,274,237,937,250$

כך שבסה"כ אנחנו מקבלים: 158,962,555,217,826,360,000 אפשרויות קונפיגורציה למכונה (יותר מ-158 קווינטיליון).

הפתרון - מלחמת העולם השניה

במלחמת העולם השניה הצליחו "לפתור" את האניגמה באופן הבא - ישנו פגם מולד יחיד בתוך המכונה, מכיוון שמדובר בפרמוטציות במעגל סגור, לא יתכן שבריצה של אניגמה האות שנכנסה כקלט תצא גם כפלט, לכן בהינתן משפט ידוע מראש - ניתן לשלול המון אופציות.

בעלות הברית ניצלו את החולשה הנ"ל ואת העובדה שכל איגרת של הגרמניים הייתה נגמרת במשפט "הייל היטלר" ובכך כבר הצליחו להוריד משמעותית את מספר הקונפיגורציות האפשריות - בהתאם לך הם בנו את מכונת ה"בומב" שבוחנת שלל אופציות אפשריות מתוך הקונפיגורציות שנשארו ואז הייתה משאירה עשרות עד מאות אפשריות אחרונות - אותן כבר היו בודקים באופן ידני.

רעיון הפרויקט

תחילה רצינו לנסות לפתור את האניגמה בעזרת כלים מודרניים, לראות איך אפשר לפתור את הצופן הזה בעזרת אלגוריתמים של למידת מכונה (כלומר בהינתן משפט מוצפן כלשהו - מצא את המשפט המקורי).
הבעיה כאן היא מהותית: בכל לחיצה על אות - הקונפיגורציה של המכונה משתנה - לכן עבור טקסט באורך x תווים - המכונה צריכה ללמוד x קונפיגורציות אפשריות מתוך כל האופציות שהוצגו למעלה.

ה-Data Set

את ה-data set בנינו בעצמנו: לאחר המילואים, כדי "לחזור לכושר" של כתיבת קוד - ספיר כתב פרויקט אימפלמנטציה של המכונה האגדית ב-Cpp, תוך שימור כל המרכיבים שלה וכל מספר הקונפיגורציות העצום שיש לה. את המכונה כתב בקפידה תוך התייחסות לכל הניואנסים הקטנים שבנו אותה.

לאחר מכן הוסיף למכונה אופציה של קריאה מטקסט (לפי path של קובץ) וריצה של המכונה על כל התוכן של הטקסט - לכן אם המכונה מתחילה במצב 0 של קונפיגורציה כלשהי, לכל אות במיקום ה- i בטקסט, המכונה תקודד את האות כאשר היא במצב i של קונפיגורציה 0 (כלומר בקונפיגורציה ההתחלתית (מצב ה-0) רוץ i טרנספורמציות של הקונפיגורציה של המכונה - זה מצב ה-i של המכונה).

העברנו את השיטה הנ"ל לכתיבת התוצאות לתוך קובץ טקסט כאשר בכל שורה בקובץ המילה הראשונה היא המילה המקורית (w) והמילה השניה היא המילה המקודדת ($Enigma(w)$) עם רווח (space) ביניהן.

אך לא הצלחנו לאמן את המודל לפי ה-dataset הזה (אחוזי הצלחה שנעים בין 0.01%-0.5%).

לאחר בדיקה מעמיקה (וארוכה) התבהרה הבעיה (!): מבחינת כל מודל שאימנו וניסינו - התחלנו לאמן אותו עם מילה אשר קודדה החל מ-קונפיגורציית 0 ($initial - config$) כלשהי של המכונה, מילה הבאה אשר נכנסה לאימון של המודל קודדה החל מ-קונפיגורציית 1 (שהיא x טרנספורמציות של המכונה החל ממצב ה-0, כאשר x היא אורך המילה הראשונה)

ובהכללה כל מילה i ב-dataset קיבלה קונפיגורציה $\sum_{x=0}^{i-1} initial - config.step(sizeof(word(x)))$

ובמילים אחרות: לכל מילה (בפוטנציאל) קונפיגורציה שונה לגמרי מקודמתה.

לכן, ביקשנו לקחת שתי הנחות לגביי ה-data שאיתו אנחנו עובדים:

- כל מילה שהוצפנה, מתחילה את ההצפנה שלה במצב ה-0 של המכונה.
- קונפיגורציית ה-0 של המכונה תהיה זהה בכל ה-Data Set.

באופן הזה עדיין נצטרך לזהות קונפיגורציה התחלתית אחת מיני קווינטליונים - אך לפחות היא לא תשתנה ממילה למילה.

מרכיבי ה-Dataset

בהתחלה ה-dataset נבנה על גביי ערכי ויקיפדיה של ה-enigma - הסתכם בסך הכל ל-16Kמילים.

ניסינו כמה וכמה ריצות על כמה וכמה מודלים שונים - אך לכל היותר הגענו לביצועים של 70% הצלחה - לאחר התייעצות, הכרנו את Tiny Shakespeare - אוסף של כל המחזות של שייקספיר, האוסף הנ"ל מכיל יותר מ- 40K שורות ממחזות של שייקספיר - דבר שהעלה את גודל ה-dataset ליותר מ-200K מילים סך הכל.

ניקוי והסרת הרעש מה-Dataset

ההוספה התלוותה עם קשיים - היינו צריכים להוסיף ווליידציות כאלה ואחרות. תחילה ניסינו באופן ידני למחוק את כל התווים הבעייתיים מהקובץ העצום אך בכל פעם שחשבנו שסיימנו, הפעלנו את המודל על ה-data ובכל פעם מחדש הוא נתקע כתוצאה מ-data לא נכון, בעיקר תווים (chars) שהוא לא ידע להתמודד איתם (אותיות גדולות, אותיות לא באנגלית, מספרים, תווים מיוחדים וכו'). לבסוף הבנו שאין מנוס אלא מראש למנוע את המצב הנ"ל על ידי ניקוי ה-data כבר בשלב יצירת ה-dataset שנמצא בקוד האניגמה המקורי.

האלגוריתמים בהם השתמשנו

AdaBoost (Adaptive Boosting) using Decision Tree

מבוא:
Adaboost הוא אלגוריתם למידה אנסמבלי (המשלב מספר מודלים אינדבידואליים) המשמש למשימות סיווג ורגרסיה. מטרתו העיקרית היא לשלב מספר מסווגים חלשים, בדרך כלל מודלים פשוטים, למסווג או מנבא חזק אחד. המטרה היא לשפר את הביצועים החזויים על פני מסווג חלש יחיד על ידי התמקדות איטרטיבית במקרים שסווגו בצורה שגויה בנתוני האימון.

Adaboost עובד באמצעות תהליך איטרטיבי. בתחילה, לכל מופע אימון מוקצה משקל שווה. מסווג חלש, לרוב עץ החלטות עם עומק מוגבל (גדם), מאומן על הנתונים המשוקללים הללו. לאחר האימון, Adaboost מעריך את הביצועים של המסווג החלש. לאחר מכן הוא מקצה משקלים גבוהים יותר למקרים שסווגו בצורה שגויה, מה שמניע את המסווג החלש הבא להתמקד יותר במקרים הקשים הללו. תהליך זה חוזר על עצמו עבור מספר קבוע מראש של איטרציות או עד להשגת מודל מושלם. המודל הסופי משלב את התחזיות של כל הלומדים החלשים, כאשר תרומתו של כל לומד חלש משוקללת לפי ביצועיו.

Adaboost עם עצי החלטה כמסווגים חלשים יעיל במיוחד עבור משימות סיווג בינארי. הוא עובד היטב עם מאגרי נתונים המכילים שילוב של תכונות מספריות וקטגוריות. עם זאת, ניתן ליישם את Adaboost גם על משימות רגרסיה והוא יכול לעבוד עם סוגים שונים של נתונים. יכולת ההסתגלות והביצועים החזקים שלו הופכים אותו למתאים למגוון רחב של יישומי למידת מכונה, במיוחד כאשר מתמודדים עם מערכות יחסים מורכבות בתוך הנתונים.

במימוש שלנו:
מאחר והאלגוריתם משלב מספר חוקים חלשים כדי ליצור חוק חזק, נוכל להשתמש בכל חילוף אות ברמה נתונה (אינדקס אותיות - כלומר כל אינדקס ישמש כרמה מסוימת) כחוק חלש ולשלב אותם. כך בתקווה האלגוריתם יצליח ללמוד את דפוס התצורה של מכונת ה-Enigma.

יש בעיה אחת שאנחנו יכולים לחשוב עליה מראש - אפילו במקרה שהאלגוריתם הזה יעבוד בצורה מושלמת, הוא עדיין תמיד חסום על המילה הארוכה ביותר במערך הנתונים - מכיוון שהכללים שלנו נקבעים רק לגודל המקסימלי של מערך הנתונים.

ניקח את המידע במאגר שלנו ואת כל המילים נאתחל לאורך המילה לגודל קבוע - מכיוון ש-AdaBoost מצריך feature vector באורך קבוע. נחלק את ה-data לאימון ולמבחן ונאמן את המודל - אימון המודל יתבצע X פעמים כאשר X הוא אורך המילה הארוכה ביותר במאגר האימון שלנו -> נרצה שבכל איטרציה, AdaBoost ילמד את צורת הפרמוטציה של אותו האינדקס (בכל אינדקס 26 אותיות ממופות אל 26 אותיות אחרות).

SVM - Support Vector Machine

מבוא:

SVM הוא אלגוריתם למידה מפוקח המשמש למשימות סיווג ורגרסיה. המטרה העיקרית של SVM היא למצוא את המישור האופטימלי המפריד בצורה הטובה ביותר בין המחלקות השונות במרחב התכונות. בסיווג, SVM שואפת ליצור גבול החלטה שממקסם את המרווח בין מחלקות, ולמקסם את ההפרדה בין נקודות נתונים של מחלקות שונות. ניתן להשתמש ב-SVM גם למשימות רגרסיה, שבהן הוא שואף למצוא מישור המתאים ביותר לנתונים תוך מזעור שגיאות.

SVM הופך את נתוני הקלט למרחב בעל ממדים גבוהים יותר שבו ניתן למצוא היפר מישור כדי להפריד בין המחלקות. הוא עושה זאת באמצעות פונקציית ליבה, המאפשרת ל-SVM למפות באופן מרומז את תכונות הקלט לתוך מרחב ממדי גבוה יותר מבלי לחשב את הטרנספורמציה באופן מפורש. ברגע שהוא נמצא במרחב בעל הממדים הגבוהים יותר, SVM מוצא את המישור הממקסם את המרווח בין המחלקות. נקודות הנתונים הקרובות ביותר להיפר מישור נקראות וקטורי תמיכה, והן קובעות את המיקום והכיוון של המישור. SVM מבקש למצוא את המישור הממזער את שגיאת הסיווג תוך מקסום השוליים, מה שהופך אותו לעמיד בפני חריגים ויעיל בטיפול במערכי נתונים מורכבים.

SVM יעיל גם עבור מאגרי נתונים הניתנים להפרדה ליניארית וגם עבור מאגרי נתונים שלא ניתנים להפרדה ליניארית. הוא עובד היטב עם נתונים במימדים גבוהים והוא שימושי במיוחד כאשר מספר התכונות גדול ממספר הדגימות. SVM מתאים למשימות סיווג עם תוצאות בינאריות ורב מחלקתיות. הוא יכול להתמודד עם נתונים מספריים וקטגוריים, מה שהופך אותו למגוון עבור סוגים שונים של מאגרי נתונים. עם זאת, ייתכן ש-SVM לא יתאים למאגרי נתונים גדולים מאוד בשל המורכבות החישובית שלו, והוא עשוי לדרוש בחירה קפדנית של היפרפרמטרים לביצועים מיטביים. בסך הכל, SVM מתאים למשימות בהן רצויה הפרדה ברורה בין מחלקות וכאשר עוסקים במאגרי נתונים בגודל בינוני.

במימוש שלנו:

אחת הבעיות העיקריות בהם נתקענו בעת מימוש מודל SVM הייתה ש-ההתייחסות לבעיה כבעיית סיווג, בה כל אות מקורית ממופה לאות מקודדת, אינה מתאימה כלל (בעקבות החשיבות של מיקום האות במילה מאחר וכל אינדקס וכל אות מקבלות משמעות שונה בתהליך ההצפנה).

לכן, נדרשנו לחשוב על הגדרה שונה באופן משמעותי.

בהתחלה ניסינו קידוד מילה למילה - זו הייתה בעיית סיווג בעייתית שכן במשימת סיווג או רגרסיה טיפוסית, y (המסמן את הערך הצפוי = label) צפוי להיות מערך 1D, שבו כל אלמנט מתאים לתווית או ערך בודד לכל דגימת קלט. עם זאת, במקרה שלנו, כל דגימת קלט (מילה) משויכת למספר תוויות (התווים המקודדים), מה שהופך אותה לבעיית סיווג מרובה תוויות.

לכן, ננסה לפשט את הבעיה - התמקדות בקידוד תו לתו ולא במילים שלמות. גישה זו מפחיתה משמעותית את המורכבות. גישה זו מתייחסת לכל תו במערך הנתונים כנקודת נתונים נפרדת, כאשר תכונת הקלט היא התו המקורי, ופלט היעד הוא התו המקודד.

Regression - Random Forest Regressor

מבוא:

Random Forest הוא אלגוריתם למידה אנסמבלי (המשלב מספר מודלים אינדבידואליים) רב תכליתי המשמש לסיווג, רגרסיה ומשימות אחרות. מטרתו היא לשפר את הביצועים של עץ החלטה בודד על ידי יצירת "יער" של עצי החלטה ושילוב תחזיותיהם. Random Forest שואפת להפחית overfitting ולהגביר את החוסן על ידי אימון עצי החלטה מרובים על תת קבוצות אקראיות של נתוני האימון ושילוב תחזיותיהם באמצעות הצבעה או מיצוע (ממוצע התוצאות).

Random Forest עובד על ידי יצירת אנסמבל של עצי החלטה. במהלך האימון, עצי החלטה מרובים גדלים תוך שימוש בקבוצות משנה שונות של נתוני האימון והתכונות. כל עץ מאומן באופן עצמאי, ובכל פיצול נלקחת בחשבון תת-קבוצה אקראית של תכונות, מה שעוזר לקשור את העצים ולהפחית overfitting. במהלך חיזוי, תחזיות העצים הבודדות משולבות באמצעות מיצוע (לרגרסיה) או הצבעה (לסיווג) כדי לייצר את התחזית הסופית. גישת אנסמבל זו מביאה בדרך כלל למודל מדויק וחזק יותר בהשוואה לעץ החלטות בודד.

Random Forest מתאים היטב למגוון רחב של מערכי נתונים ומשימות. זה יכול לטפל גם בבעיות סיווג וגם בבעיות רגרסיה, והוא יעיל עם נתונים במימד גבוה המכילים מאפיינים מספריים וקטגוריים כאחד. Random Forest עמיד בפני חריגים, ערכים חסרים ונתונים רועשים, מה שהופך אותו למתאים עבור מערכי נתונים בעולם האמיתי עם דרגות שונות של איכות. הוא יכול להתמודד עם מערכי נתונים לא מאוזנים והוא נוטה פחות ל-overfitting בהשוואה לעצי החלטה בודדים. עם זאת, ייתכן ש-Random forest אינו הבחירה הטובה ביותר עבור משימות הדורשות מודלים ניתנים לפירוש, מכיוון שמכלול עצי ההחלטה יכול להיות מורכב לפירוש.

במימוש שלנו:
מכיוון שבעיה זו אינה כרוכה בחיזוי של משתנה יעד רציף (continuous target variable) - אלא בחיזוי של משתנים רבים יחד (קידוד רצף אותיות שאינם נעים ברצף לינארי - אלא משתנים בהתאם למיקומם במילה) נצטרך Multi-output Regression, כאשר המטרה היא לחזות מספר משתני יעד במקום רק אחד.
שיטה זו היא בחירה טובה מכיוון ש-Multi-output Regression עובדת בצורה הטובה ביותר כאשר משתנים קשורים זה לזה - ובמקרה שלנו האניגמה משנה את התצורה שלה בכל אינדקס (עקב תנועת הרוטורים) כך שבכל אות היית מצפה לפלט שונה.

לאחר מספר ניסיונות, מצאנו שמימוש הרגרסיה שמביא לנו את התוצאה הטובה ביותר הוא Random Forest Regressor כאשר אלגוריתם Random Forest משלב תשובות של עצי החלטה רבים - על מנת להקטין overfitting ולשפר את הדיוק. כל עץ החלטה (Decision Tree) בעל שונות גבוהה, אך כאשר אנו משלבים את כולם במקביל, השונות המתקבלת היא נמוכה מכיוון שכל עץ החלטה מקבל מתאמן היטב על sample data מסוים - כתוצאה מכך הפלט של Random Forest אינו מתבסס על עץ אחד אלא על המון עצים.

מה שעשינו זה להמיר את התווים המקודדים במספרים שלמים לייצוג מטריצה בינארית המתאים לרגרסיה. כל תו (כולל הריפוד) מיוצג כווקטור בינארי עם כל האפסים מלבד האינדקס של התו, המוגדר ל-1.

השתמשנו בקידוד one-hot, תהליך המשמש להמרת נתונים לפורמט שניתן לספק לאלגוריתמים של למידת מכונה כדי לעשות עבודה טובה יותר בחיזוי. זה עובד על ידי המרת כל ערך לעמודה קטגורית חדשה ומקצה

ערך 1 או 0 (נכון/לא נכון) לעמודות הללו בכל שורה. בבניית המודל, השתמשנו ב-100 עצים על מנת ליצור את היער של המודל.

LSTM - Long Short-Term Memory

מבוא:

LSTM הוא סוג של ארכיטקטורת רשת עצבית חוזרת (RNN) שנועדה להתגבר על בעיית הגרדיאנט הנעלם ב-RNNs מסורתיים. מטרת LSTM היא ללכוד ביעילות תלות ויחסים ארוכי טווח בנתונים עוקבים, מה שהופך אותו למתאים במיוחד למשימות הכוללות נתונים מסדרות זמן, עיבוד שפה טבעית (NLP), זיהוי דיבור ועוד. שלא כמו RNNs מסורתיים, רשתות LSTM יכולות לשמור על מידע על פני רצפים ארוכים, מה שהופך אותן למתאימות היטב למשימות שבהן ההקשר והזיכרון הם חיוניים.

LSTM פועל על ידי החדרת תאי זיכרון מיוחדים, או "שערים", המווסתים את זרימת המידע דרך הרשת. שערים אלו מורכבים משער כניסה, שער שכחה ושער פלט, כל אחד אחראי על שליטה בהיבטים שונים של תא הזיכרון. שער הקלט קובע כמה מידע חדש מותר להיכנס לתא הזיכרון, שער השכחה מסדיר את השמירה או המחיקה של מידע קיים, ושער הפלט שולט בחשיפה של תוכן תא הזיכרון לשאר הרשת. ארכיטקטורה זו מאפשרת לרשתות LSTM לזכור או לשכוח מידע באופן סלקטיבי לאורך זמן, מה שמאפשר להן ללכוד תלות ארוכת טווח בנתונים עוקבים בצורה יעילה יותר.

רשתות LSTM מצטיינות במשימות הכרוכות בנתונים עוקבים או סדרתיים, כגון זיהוי דיבור, תרגום שפה, ניתוח סנטימנטים וחיזוי שוק המניות. הם יעילים במיוחד כאשר עוסקים בנתונים שמפגינים תלות ארוכת טווח או דפוסים זמניים. רשתות LSTM יכולות לעבד סוגים שונים של נתונים עוקבים, כולל טקסט, אודיו ונתונים מספריים של סדרות זמן. הם מתאימים היטב למשימות שבהן הקשר ומידע היסטורי ממלאים תפקיד מכריע בביצוע תחזיות או ביצירת תפוקות. עם זאת, אימון רשתות LSTM יכול להיות אינטנסיבי מבחינה חישובית, במיוחד עבור מאגרי נתונים גדולים, והם עשויים לדרוש כוונון קפדני של היפרפרמטרים לביצועים מיטביים.

במימוש שלנו:

כבר כשעלתה המחשבה לפרויקט הזה, התחלנו לחפש - איזה מודל הכי מתאים למשימה שכזאת?
במסגרת הקורס לא למדנו למידה עמוקה אך נראה שעם הפריצה של מודלי השפה לחיינו בשנה האחרונה, משימה כזאת נראתה כאילו היא נועדה למודל שכזה - בכל הבדיקות שהרצנו, הוצע להשתמש ב-LSTM.

אך בכל זאת היינו צריכים עזרה והכוונה, לעשות פרויקט שכזה עם מודל שאנחנו לא מכירים בתחום דומה אך קצת שונה (למידה עמוקה) זאת אינה החלטה קלה (הידע התבסס על יוטיוב וקורס אינטרנטי שעשיתי במהלך המילואים).

דיברנו עם ליעד על הפרויקט בכללי וביקשנו הכוונה - הוא הפנה אותנו לשני מקורות (1, 2) של מידע לגביי פתרון האניגמה בעזרים של למידת מכונה - בשני המקורות התייחסו לפתרון הבעיה על ידי שימוש ב-RNN ובאופן ספציפי יותר - LSTM בשל הקשר המאסיבי של מודל זה אל בעיות שפה (דימוי שפה, קול-טקסט, ניתוח סנטימנט ועוד...)

באופן פרטני, ל- LSTM יש את היכולת לשמור על מצב על פני כמה Batch's, כלומר, מצב המכונה לאחר הצפנת מילה אחת יכול לעבור למילה הבאה, תוך חיקוי הפעולה הרציפה של מכונת Enigma (לכן התגלה כבחירה מאוד נכונה לפתרון הבעיה).

במודל בו השתמשנו היו:

- שכבת הטבעה להמרת ייצוגים שלמים של אותיות להטבעות וקטוריות צפופות.
- שכבת LSTM שמטרתה ללכוד את התלות של הרצפים, בהתחשב בתנועת הרוטור של האניגמה.
- שכבה צפופה עם הפעלת softmax למיפוי הפלט של ה-LSTM להתפלגות הסתברות על אותיות פלט אפשריות.

השוואת ביצועים

ניתן לראות את מימוש הקוד ותוצאות ההרצה גם במחברת Break The Enigma Project

	LSTM	Regression **	SVM	AdaBoost
Fit time (seconds)	236.70	145.94	1278.259	393.99
Accuracy	99.36%	98.59%	29.83%	92.95%

** מבחן הרגרסיה פעל אך ורק על חצי מה-dataset (כ-200 אלף דגימות). ניסינו מספר מימושים שונים של רגרסיה (כמובן את אלו המתאימים לבעיה מהסוג הזה) חלקם לא עבדו עם ה-dataset המלא ולכן השתמשנו במצומצם יותר - לבסוף מצאנו שעבור המימוש הנבחר, גם עם dataset קטן יותר הביצועים היו טובים יותר.

מסקנות

אפשר לראות שאלגורתמים שייעודם קרוב או נועדו לפתור בעיות מהסוג שאנחנו מתמודדים איתו - הגיעו לתוצאות מרשימות: מהאלגוריתמים שנלמדו בכיתה, בולטים לטובה:

- ראשון לטובה הוא AdaBoost שמחזיר תוצאות טובות ב-93% מהמקרים - כאשר החסרונות שלו הם אי העמידות במילים "ארוכות" (ככל שהמילה ארוכה יותר הוא נוטה לטעות - כנראה בשל האופי של שימור החוקים שלו).
- בנוסף ה-Regression עם תוצאות טובות משל עצמה 98% הצלחה אך עם פגם אחד מהותי - כוח העיבוד הנדרש על מנת לאמן מודל שכזה הוא גדול מאוד (בחלק מהניסויים שעשינו הגענו לשימוש של 12GB של זיכרון RAM מה שהביא לקריסות חוזרות של ה-Colab).

חשוב לציין שגם במימוש AdaBoost וגם במימוש של Regression עושים ניצול של עצים כדי למקסם את תוצאה ה-predict - כך באופן עקיף גם העצים שנלמדו בכיתה התבררו כיעילים לבעיה מסוג זה (בסה"כ זה הגיוני בהינתן הטבע של הבעיה לחפש את הפרמוטציה הבאה הנכונה בכל שלב).

אלגוריתמים שמטרתם רחוקה מהמטרה אותה אנחנו מחפשים הביאו את התוצאות הכי פחות טובות:

- אלגוריתם SVM עומד גם על זמן האימון הארוך ביותר (יותר מ-21 דקות !!) ובנוסף על התוצאות הנמוכות ביותר עם 29% הצלחה.
  אולי הטרנספורמציה שלנו של הבעיה אל תוך האלגוריתם הייתה שגויה, אך בסך הכל נראה ש-SVM נועד בייסודו לפתרון בעיות מסוג אחר.

לבסוף נציין את האלגוריתם המצטיין, נשים לב שהאלגוריתם לא נלמד בכיתה - והוא גרסה מתקדמת של RNN - אלגוריתם מבוסס למידה עמוקה. בכל המקורות שהסתכלנו הוא הוצג כסוג של סטנדרט ונקודת התחלה טובה לכל פרוייקטים sequence 2 sequence ולכן ניסינו אותו:

- אלגוריתם LSTM הצליח להביא את התוצאות הטובות ביותר עם 99% הצלחה, כמו כן השימוש שלו בזיכרון RAM לא היה מאסיבי ולכן הוא יכל לעבור על ה-dataset המלא (לעומת Regression שהפיל את ה-Colab).
  בנוסף האלגוריתם קיבל זמן אימון של 236 שניות - כלומר בהשוואה ל-AdaBoost זמן האימון של LSTM היה מהיר ב-158 שניות ומדויק יותר ב-6% (דיוק של 99% אל מול 93%).

מקורות

פירוט מקורות:

1. Break The Enigma (קישור אל המחברת של הפרויקט).
2. Enigma-Machine Implement in Cpp (קישור לפרויקט הגיט האב איתו יצרנו את הדטא).
3. How we cracked the Enigma code using Artificial Intelligence
4. 158,962,555,217,826,360,000 (Enigma Machine) - Numberphile
5. Enigma machine
6. Bombe
7. How did the Enigma Machine work?
8. Flaw in the Enigma Code - Numberphile
9. Recurrent Neural Networks (RNNs), Clearly Explained!!!
10. Recurrent neural network
11. What are recurrent neural networks?
12. What is LSTM (Long Short Term Memory)?
13. Random Forest Regression
14. Multi-output Regression in Machine Learning
15. Learning the Enigma with Recurrent Neural Networks
16. Enigma encoding machine
17. Cryptanalysis of the Enigma
18. Cracking Enigma in 2021 - Computerphile
19. Tiny Shakespeare - Data Set
20. William Shakespeare