# Data Preprocessing

## הקדמה

בלמידת מכונה מפעילים אלגוריתם למידה על מידע מתויג ונתון בידינו כדי ללמד את המודל כיצד לחזות נכון עבור מידע לא מתויג. המידע שעליו אנו לומדים צריך להיות מייצג בצורה טובה את כל המידע מסוג זה שנמצא בעולם. במידה והמידע לא מייצג אזי המודל ידע לחזות נכון רק עבור סוג מסוים של אובייקטים ולא בכל המקרים.

## תכונה ריבועית (Quadratic Feature)

בפיתוח מודל, כאשר אנו נותנים לכל תכונה משקל, אנו מקבלים משוואה ליניארית שאם נציב בה את כל את כל התכונות של אובייקט חדש x תחזיר לנו ניבוי של ערך y המתאים לו. החיסרון בשיטה זו הוא שתמיד נקבל משוואה ליניארית שיכולה להיות לא מדויקת לעומת פרבולה שהיא יותר גמישה. כדי לקבל משוואה ריבועית בלי לשנות את המודל, מספיק להוסיף עבור כל תכונה i תכונה חדשה i’ שתקבל את הערך . לדוגמה, אם במודל יש לנו תכונה אחת לכל אובייקט, אזי המשוואה שנקבל לאחר הוספת התכונה הריבועית היא: , שהיא אכן פרבולה.

לא תמיד נרצה להוסיף הרבה תכונות ריבועיות. הסיבה לכך היא שיכול להיווצר מצב של התאמת יתר (Over-Fitting) שעלולה להביא לתחזיות גרועות מאוד. מתברר שהוספת תכונות מלאכותיות רבות אכן יפחית את הטעות (מרחק ממוצע של אובייקט מהמשוואה) בקבוצת האובייקטים ב-train, אך בשלב מסוים הטעות בקבוצת ה-test תתחיל לגדול.

## נורמליזציה

מנרמלים את הדאטה על ידי כך שלכל תכונה k מחשבים ממוצע וסטיית תקן , ואז מעדכנים כל ערך בתכונה זו:

כלומר לכל ערך בדאטה מחסירים את הממוצע ומחלקים בסטיית תקן של התכונה שלו. פעולה זו גורמת לכך שהערכים בכל תכונה מתפלגים נורמלית סטנדרטית, כלומר הממוצע שלהם הוא 0 וסטיית התקן היא 1. ישנם מספר סיבות מדוע נרצה לנרמל את הדאטה:

* בלמידה עמוקה, כאשר הערכים תמיד חיוביים אזי הגרדיאנט הוא או חיובי או שלילי. במצב כזה התנועה לכיוון הגרדיאנט תהיה תמיד בצורת זיג-זג ולא באופן ישיר, כלומר תהליך הלמידה לא יעיל. נורמליזציה של הדאטה סביב ה-0 פותרת בעיה זו.
* במודלים שאינם למידה עמוקה, אתחול ה-W וה-b ל-0 בתחילת הרצת אלגוריתם הלמידה יעזור לנו למצוא את המינימום יותר מהר, שכן זה הממוצע לכל התכונות.
* לקיחת צעד לכיוון של כל מימד הוא יותר סטנדרטי ולא תלוי ביחידות שבהן נמדדת התכונה.
* חשוב ביותר כאשר משתמשים ברגולריזציה - הוספת המשקולות ל-Loss היא אחידה לכל התכונות.
* ניתן להשוות חשיבות של תכונות על ידי השוואה בין המשקולות שלהן מאחר שהערכים אינם תלויים ביחידות שונות. מאפשר גם להשוות בין Dataset שונים.
* מונע חישוב של מספרים גדולים מאוד שיכולים להביא לשגיאה.

חשוב מאוד לשים לב שלא מנרמלים את ה-label, שכן אז הסטייה הממוצעת גם כן מנורמלת, וכדי לקבל אותה בצורה המקורית יש להכפיל בשונות . נרמול הדאטה יבוצע לפני הרצת האלגוריתם, אולם ב-MB-GD אפשר לנרמל כל קבוצה לאחר שנבחרת (Batch Normalization).

### Min-Max Scaling

עוד סוג של נרמול הוא , כאשר ו- הם הערכים המקסימליים והמינימליים בתכונה של x. נרמול זה גורם לכך שכל הערכים בדאטה יהיו בין 0 ל-1.

## Train, Test, Validation

לאחר פיתוח מודל נרצה להיות מסוגלים לבדוק את המודל שאנו מפתחים. בדיקת המודל תיעשה על מידע שבו אנו יודעים את הערך y אותו המודל מנסה לנבא. נשווה בין ערכי האמת למה שהמודל ניבא.

לא נרצה לבדוק את איכות המודל על המידע שעליו המודל התאמן, שכן בעיה מאוד נפוצה שיכולה לקרות היא Over-Fitting, שבה המודל יהיה מאוד מדויק על המידע שהתאמן עליו אך על מידע חדש יציג תוצאות מאוד גרועות (ראה תמונה). לכן נרצה לחלק את המידע עוד לפני פיתוח המודל ל-train ו-test כך שעל ה-train נתאמן ועל ה-test נבדוק את איכות המודל. אמנם גם בחלוקה זו יכולה להיות בעיה, שבכל ניסוי נבדוק את הטעות ב-test ונשפר, כך שנקבל את המודל עבורו הטעות ב-test מינימלית. אמנם אז ה-test יאבד את המשמעות שלו, שכן יכול להיות שאנו מתאימים את המודל רק עבור ה-test ולא עבור מקרה כללי. כל המטרה ב-test היא שבודקים את הטעות בו פעם אחת בסוף הפיתוח של המודל כדי להעריך כמה המודל טוב ולא לפתח את המודל באמצעותו. הפתרון לכך הוא לחלק את כל האובייקטים שיש בידינו לשלושה חלקים:

1. **train** - באמצעותו לומדים ומפתחים את המשוואה הליניארית על ידי הפעלת שיטת Gradient Descent שתוארה לעיל.
2. **validation** - מטרתו היא לעזור לנו לקבוע את מספר התכונות המלאכותיות שנכניס למודל. נשחק עם מספר התכונות וסוגן ואז נבדוק את הטעות בקבוצת ה-validation. נבחר את התכונות עבורן הטעות בקבוצת ה-validation היא מינימלית.
3. **test** - שעליו מריצים את המודל הסופי פעם אחת. מטרתו היא להעריך את טיב המודל.

אם אין הרבה אובייקטים החלוקה המקובלת היא 60% ל-train, 20% ל-validation, ו-20% ל-test. אמנם כאשר יש הרבה אובייקטים אפשר להפחית את כמות האובייקטים ב-test וב-validation. לדוגמה, אם יש מיליון אובייקטים אין צורך שה-test יהיה 200,000 אלא אפשר להסתפק גם ב-2000, וכן ניתן להסתפק ב-10,000 ב-validation. ב-train תמיד נרצה שיהיו כמה שיותר. ה-train וה-test צריכים להיות די זהים ברמת ההתפלגות. אם ה-train מייצג מידע מסוג אחד וה-test מידע מסוג אחר, אין לדעת האם המודל יעבוד גם ב-test שכן זהו מידע שחדש לו.

## Imbalanced Data

כאשר יש המון מידע ממחלקה אחת אך יש רק מעט מידע ממחלקה אחרת, המודל יכול ללמוד לחזות נכון רק עבור המחלקה הראשונה, שכן עיקר ה-Loss נובע משגיאות על מחלקה זו. אולם בדרך כלל נהיה מעוניינים להיות מדויקים יותר עבור המחלקה השנייה. לדוגמה, כאשר נרצה לחזות האם לאדם יש מחלה כלשהי, לרוב האנשים התשובה תהיה שלילית אך אנו מעוניינים לדייק יותק דווקא עבור אלו שכן חולים. יש מספר דרכים להתמודד עם דאטה לא מאוזן:

1. Min-Batch - נחלק את קבוצת ה-train למחלקה שמיוצגת הרבה ולמחלקה שמיוצגת מעט. נשתמש ב-min-batch כך שמידע אותו נריץ יהיה כל פעם חלק אחר מהמחלקה שמיוצגת הרבה אך נצרף אליה כל פעם את המידע של המחלקה שמיוצגת מעט.
2. Under-sampling - לא להשתמש בכל המידע שיש לנו אלא לדגום ב-train ממחלקת הרוב קבוצה שווה לגודל מחלקת המיעוט וזה יהיה ה-train החדש שלנו.
3. Over-sampling - נשכפל את המידע במחלקה הקטנה מספר פעמים. יש דרך קלה ויעילה לעשות זאת באמצעות טכניקת SMOT (Synthetic Minority Oversampling Technique).
4. Loss function intervention - בדר"כ 1 יהיה המחלקה הקטנה ו-0 המחלקה הגדולה. נרצה "להעניש" את המודל על כל אובייקט שינבא 0 ובפועל הוא 1. לכן נכפיל בפונקציית Loss את הרכיב שסוכם טעות זו בקבוע , כלומר נכפיל במנה של מספר האובייקטים הכולל ב-train במספר האובייקטים השייכים למחלקת המיעוט. **זוהי השיטה המומלצת לבעיית דאטה לא מאוזן**. בספריות למידת מכונה לכל מודל יש פרמטר שאחראי לאזן את הדאטה בצורה זו.
5. Reduce prediction threshold - במקום לקבוע ש0.5 זהו הסף שמעליו נשייך למחלקה 1 ומתחתיו נשייך למחלקה 0, נקבע סף נמוך יותר כך שיותר אובייקטים ינובאו שייכים למחלקה 1. זוהי שיטה לא מומלצת.