# הקדמה ללמידת מכונה

## הקדמה

למידת מכונה היא תת-תחום במדעי המחשב ובבינה מלאכותית העוסק בפיתוח אלגוריתמים שבאמצעותם ניתן ליצור תוכנות מחשב הלומדות מתוך דוגמאות כיצד להסיק תכונות לא ידועות על מידע חדש. למידת מכונה פועלת במגוון משימות חישוביות בהן התכנות הקלאסי אינו אפשרי. המקרה הנפוץ ביותר בו נרצה להשתמש בלמידת מכונה הוא בעיית זיהוי שמומחה אנושי מסוגל לפתור, אך לא מסוגל לכתוב את הכללים לזיהוי בצורה מפורשת או שהם משתנים עם הזמן ולא ניתנים לכתיבה מראש. העקרון המרכזי בלמידת מכונה הוא לתת למחשב קבוצה גדולה של מידע מתויג וכלים שבאמצעותם המחשב יוכל ללמוד את המידע ולפתח תוכנה שתדע להכריע בהסתברות גבוהה על כל מידע חדש.

## סוגי בעיות

**Classification** - נתונים k מחלקות. אנו מעוניינים לסווג כל אובייקט לאיזה מחלקה מ-k המחלקות הוא שייך. פעמים רבות מספר המחלקות הוא 2 המייצגים "כן" או "לא", אך גם יכולים להיות מספר רב של מחלקות. דוגמאות:

* בהינתן נתונים על אדם נרצה לדעת האם הוא עובד או מובטל.
* בהינתן תמונה של פרי נרצה לדעת מהו סוג הפרי.

**Regression** - בהינתן אובייקט x נרצה לנבא עבורו איזשהו ערך מספרי רצוף y. דוגמאות:

* בהינתן נתונים על אדם כלשהו נרצה לנבא מהו משקלו או גובהו.
* בהינתן גרסה של פלאפון נרצה לנבא מהו מחיר הפלאפון.

**Clustering** - בהינתן n אובייקטים לקבץ אותם למספר קבוצות, ושיוך דוגמה חדשה לקבוצה המתאימה ביותר. כל האובייקטים הנמצאים באותה קבוצה דומים זה לזה יותר מאשר לאובייקטים השייכים לקבוצות אחרות. לדוגמה, פילוח של לקוחות לפי התנהגות צרכנית ותכונות דמוגרפיות.

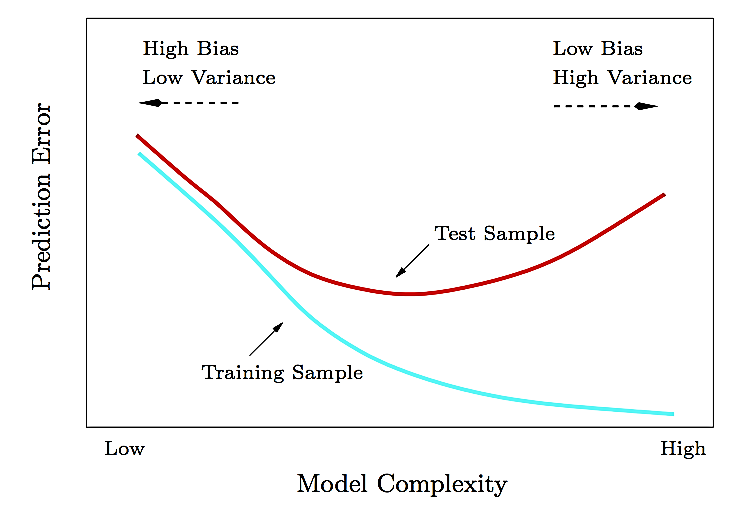
**Ranking** - בהינתן n אובייקטים לדרג אותם מהגבוה לנמוך לפי העדפות משתמש. לדוגמה, לדרג אתרי האינטרנט שנמצאו בחיפוש לפי העדפות המחפש כך שהטובים ביותר יופיעו קודם.

## מודל למידת מכונה

הרכיבים המרכזיים בפתירת בעיה באמצעות למידת מכונה הם:

1. **נתונים** - קבוצת אובייקטים כך שלכל אובייקט x יש תכונות מסוימות ותכונה מיוחדת y. מידע זה יחולק ל-train ו-test.
2. **פונקציית מודל** - פונקציה שאם נכניס אליה אובייקט עם כל התכונות תנבא מהו הערך שלו. כל פונקציה כזו מורכבת מפרמטרים . כפי שנראה מודל זה עשוי להיות מורכב מאוד.
3. **Loss Function** - פונקציה שבהינתן labels וחיזוי לכל אובייקט בדאטה, ומחזירה מספר המציין כמה אי-דיוק/טעות ממוצע יש בהשוואה ביניהם. ככל שהערך שמחזירה קטן יותר כך המודל טוב יותר. בדרך כלל פונקציה זו תהיה סכום השגיאות של כל האובייקטים ב-train בריבוע.
4. **אלגוריתם** - ישמש למציאת ה- Loss Functionהמינימלית על ידי קביעת פרמטרים מדויקים ככל הניתן. דוגמה לאלגוריתם נפוץ בתחום זה הוא Gradient Descent.

### Train, Test, Validation

בכל אחד מסוגי הבעיות לעיל כדי שנוכל גם לבדוק את המודל שאנו מפתחים, לא נשתמש בכל האובייקטים שברשותנו לפיתוח המודל, אלא נשאיר מספר אובייקטים (שאנו יודעים לאיזה מחלקה הם שייכים / מה הערך שלהם) בצד כדי שנוכל להעריך כמה המודל שפיתחנו הוא איכותי. נשווה בין ערכי האמת למה שהמודל ניבא. חלוקה זו של המידע הקיים הוא חלוקה לקבוצות train ו-test.

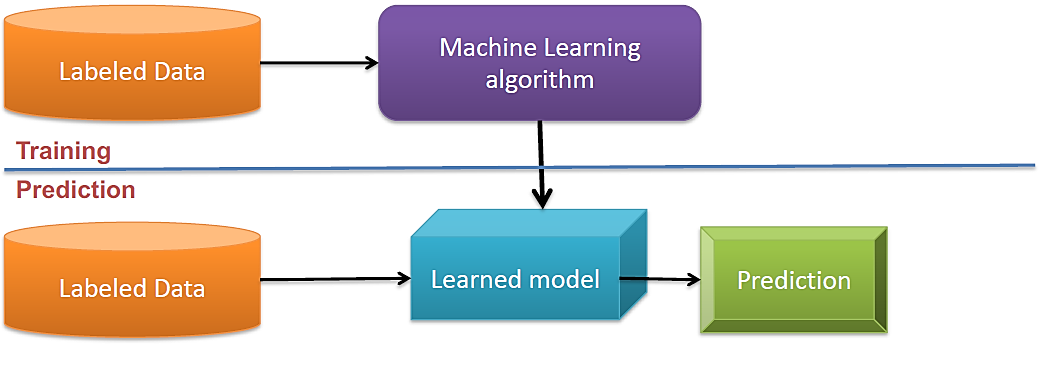
לא נרצה לבדוק את איכות המודל על המידע שעליו המודל התאמן, שכן בעיה מאוד נפוצה שיכולה לקרות היא Over-Fitting, שבה המודל יהיה מאוד מדויק על המידע שהתאמן עליו אך על מידע חדש יציג תוצאות מאוד גרועות (ראה תמונה). לכן נרצה לחלק את המידע עוד לפני פיתוח המודל ל-train ו-test כך שעל ה-train נתאמן ועל ה-test נבדוק 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 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.

### שלבי בניית המודל

לפי האמור לעיל כל בניית מודל בלמידת מכונה מחולק לשני שלבים: שלב הלמידה ושלב הבדיקה. בשלב הלמידה יש לנו צמדים , כאשר כל זהו אובייקט ה-i ב-train ו- זהו ה-label המתאים לו. המטרה היא למצוא את הפרמטרים האופטימליים בפונקציית המודל עבורם הערך בפונקציית ה-Loss יהיה מינימלי. בשלב הבדיקה, נקרא גם Generalization, מריצים את פונקציית המודל על כל האובייקטים ב-test שאותם המודל לא ראה ובודקים כמה הוא טעה על מידע זה. הטעות נמדדת על ידי ערך מספרי שנרצה שיהיה כמה שיותר קרוב ל-0.

## בעיות בפיתוח מודל

ראשית נציין שני מדדים חשובים בבדיקת איכות מודל.

**Bias** (הטיה) - הוא ההבדל בין החיזוי הממוצע של המודל לבין הערך הנכון אותו אנו מנסים לחזות. כמו ההגדרה של loss. מודל עם bias גבוה הוא מודל פשוט מדי או שמניח הנחות לא מדויקות על המידע. Bias גבוה תמיד מוביל לשגיאות גבוהות בנתוני ה-train וה-test.

**Variance** (שונות) - היא שגיאה הנובעת מרגישות לשינויים קטנים ב-train set. שונות גבוהה יכולה לגרום לאלגוריתם למדל את הרעש האקראי בנתוני האימון, ולא את הפלט הרצוי. מודל עם שונות גבוהה מקדיש תשומת לב רבה לנתוני ה-train ואינו מכליל נתונים אחרים שלא ראה קודם. כתוצאה מכך, מודלים כאלה מתפקדים טוב מאוד בנתוני ה-train אך בעלי שיעורי שגיאה גבוהים בנתוני ה-test.

### Underfitting

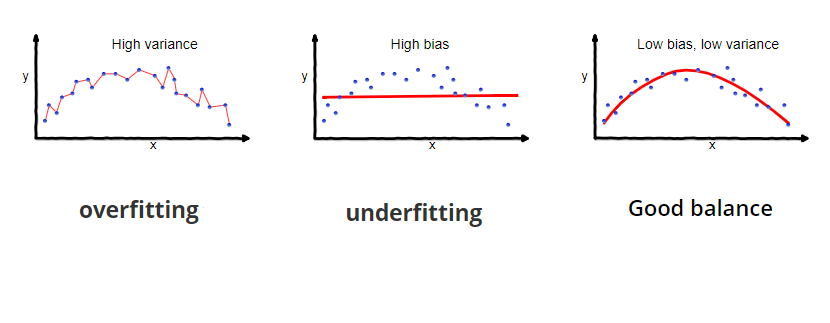
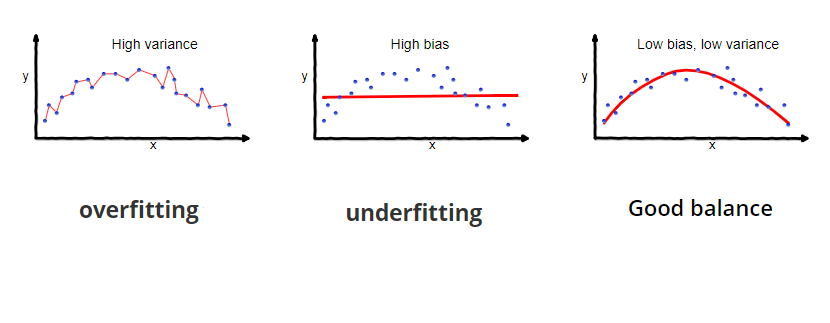
בעיה המתרחשת כאשר מודל אינו מסוגל ללמוד את הדפוס הבסיסי של הנתונים. למודלים אלה יש בדרך כלל bias גבוה ושונות נמוכה, המתבטאים בטעות גבוהה גם ב-train וגם ב-test. זה קורה כשיש לנו כמות קטנה מאוד של נתונים לבניית מודל מדויק או כשאנחנו מנסים לבנות מודל לינארי עם נתונים לא לינאריים. כמו כן, מודלים עם בעיה זו בדרך כלל פשוטים מאוד לתפיסת הדפוסים המורכבים בנתונים כמו רגרסיה לינארית ולוגיסטית.

הפתרונות לבעיה זו הם בדרך כלל הוספת מידע או להשתמש באלגוריתם למידה מורכב יותר שידע לזהות את הדפוסים במידע. מודל מורכב יותר גם יכול להיות לאמן יותר צעדים.

### Overfitting

בעיה המתרחשת כאשר המודל שלנו לוכד את הרעש במידע יחד עם הדפוס הבסיסי בנתונים. זה קורה כשאנחנו מכשירים את המודל שלנו הרבה על מערך נתונים רועש. למודלים אלו bias נמוך ושונות גבוהה, המתבטאים בטעות קטנה ב-train וטעות גדולה ב-test, כלומר המודל מותאם מדי ל-train אך לא למידע אחר. מודלים אלה מורכבים מאוד כמו עצי החלטה הנוטים להתאמת יתר.

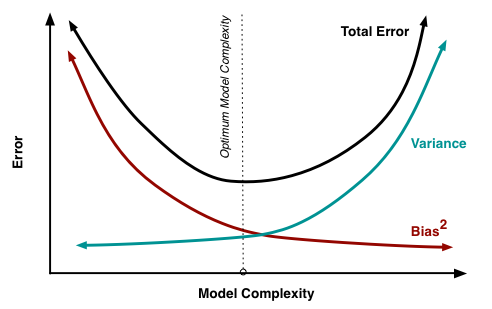
הפתרונות לבעיה זו הם בדרך כלל לפשט את המודל, להוריד תכונות או להשתמש במידע פחות רועש, כלומר שיש בו דפוס בולט. אפשרות נוספת, עליה נדון בהמשך, היא להפעיל רגולריזציה על המשקלים במודל.



### Bias-Variance tradeoff

על מנת למזער את הטעות הממוצעת של המודל (Mean Squared Error - MSE), עלינו למזער הן את ה-bias והן את השונות. אמנם זה לא טריוויאלי לעשות זאת שכן הם אחד על חשבון השני. מורכבות המודל היא הסיבה שיש tradeoff בין הטיה לשונות. אלגוריתם לא יכול להיות מורכב יותר ופחות מורכב בו זמנית. אם המודל שלנו פשוט מדי, כלומר יש לו מעט מאוד פרמטרים, ייתכן שיש לו bias גבוה ושונות נמוכה. מצד שני אם המודל יהיה יותר מורכב, כלומר יש מספר גדול של פרמטרים, אז יהיה לו שונות גבוהה ו-bias נמוך.

לכן כדי לבנות מודל טוב, עלינו למצוא את האיזון הנכון בין bias לשונות, מבלי להגיע ל-Overfitting או Underfitting, וגם למזער את השגיאה הכוללת. איזון אופטימלי של bias ושונות לעולם לא יגיע ל-Overfitting או Underfitting. חישוב האיזון הנכון ביניהם נעשה באמצעות הערך המינימלי של הנוסחה למטה, כאשר irreducible error מייצגת את הטעות שלא ניתן להימנע ממנה ונובעת מרעש של על המידע.



## סוגי למידה

נהוג לחלק את אלגוריתמי למידת המכונה למספר סוגים:

* למידה מונחית (supervised learning) - כל דוגמה מגיעה עם תווית סיווג. מטרת האלגוריתם היא ללמוד מהמידע המתויג כיצד לחזות את הסיווג של דוגמאות חדשות שאותן לא פגש בתהליך הלמידה. לדוגמה סיווג מיילים ל"חשוב" ו"לא חשוב" כאשר נתונים מיילים שכבר מסווגים. החסרונות בלמידה כזו שקשה להחליט מהו המידע הטוב ביותר ללמוד עליו ומהו האלגוריתם הטוב ביותר ללמוד איתו.
* למידה בלתי מונחית (unsupervised learning). מטרת האלגוריתמים היא למצוא ייצוג פשוט וקל להבנה של אוסף הנתונים. שיטות נפוצות מסוג זה הן חלוקה לקבוצות (clustering), והטלה ליריעות ממד נמוך כגון ניתוח גורמים ראשיים (PCA). החיסרון בלמידה זו שבדרך כלל מביאה לתוצאות פחות טובות מלמידה מונחית.
* למידה מונחית חיזוקים (reinforcement learning) - אלגוריתם הלמידה מקבל משוב חלקי על ביצועיו (רק לאחר סיום ביצוע המטלה) ועליו להסיק אילו מהחלטותיו הביאו להצלחה/כישלון. בסוף תהליך הלמידה האלגוריתם יודע בכל מצב מהי הפעולה הטובה ביותר. למידה זו משמשת כיום בעיקר בפיתוח תוכנה שתדע לשחק משחקים.

## סוגי מודלים

* מודל יוצר (generative model) - זהו מודל שבו מחשבים עבור כל מחלקה y מה ההסתברות שאובייקט x שייך למחלקה זו , וכן מה ההסתברות שאובייקט כלשהו שייך למחלקה y . דוגמה למודלים מסוג זה הם Naïve Bayes ו-Bayesian network.
* מודל מפלה (discriminative model) - הוא מודל שבו מחשבים רק מה ההסתברות שאובייקט x שייך למחלקה y . *דוגמה למודלים מסוג זה הם* Linear Regression, SVM, Decision Trees. *בדרך כלל מודל מפלה הוא יעיל יותר מאשר מודל יוצר.*

# עקרונות למידת מכונה

## PAC Learning

PAC learning (Probably approximately correct learning, ובתרגום חופשי: "למידת קרוב לוודאי בערך נכון") הוא מודל לניתוח מתמטי של בעיות מתחום הלמידה החישובית. מודל זה קובע מתי בעיה ניתנת לפתרון או לפתרון יעיל באמצעות למידת מכונה. המודל הוצג ב־1984 על ידי מדען המחשב הבריטי לסלי וליאנט.

במודל זה הלומד (אלגוריתם למידה) מקבל דוגמאות וצריך לבחור פונקציה מכלילה (הנקראת במודל זה "חוק") מתוך מחלקה מוגדרת של פונקציות אפשריות (מחלקת חוקים נקראת "היפותזה"). המטרה היא שבהסתברות גבוהה ("קרוב לוודאי"), הפונקציה הנבחרת תהיה בעלת שגיאת הכללה ("בערך נכון") נמוכה. אלגוריתם הלמידה נדרש להיות מסוגל ללמוד פונקציה בהינתן בחירה של יחס קירוב נדרש, מידת וודאות או התפלגות של דוגמאות. המודל הורחב בהמשך כדי לטפל גם ברעש על המידע, או דוגמאות המסווגות בצורה שגויה.

### הגדרה פורמלית

בהינתן בעיה P, נגדיר את X להיות מרחב כל הקלטים של הבעיה ו-Y מרחב כל הפלטים של הבעיה. עוד נסמן ב-H את קבוצת כל הפונקציות הממפות , כלומר מקבלות מופע של הבעיה ומחזירות פלט , לא בהכרח ש-y הוא הפלט הנכון עבור x. אם קיים אלגוריתם למידה A המקבל כקלט שני פרמטרים ומחזיר פונקציה , כך שבהסברות אזי טועה לכל היותר ב-על X, נאמר כי הבעיה P היא PAC Learnable. אם בנוסף קיים אלגוריתם A שעושה זאת בזמן פולינומי, אזי P היא Efficiently PAC Learnable.

### משפטים מרכזיים

יהי X מרחב קלטים של בעיה, D התפלגות על X, היפותזה H שהיא קבוצה **סופית** של פונקציות (חוקים) הממפות , ומדגם אקראי של n קלטים לפי D.

1. Generalization bound ללא טעות במדגם - אם קיימת פונקציה שהיא עקבית על S, כלומר לא טועה על אף קלט מ-S, אזי קיים כך שבהסברות לכל הפחות הטעות האמיתית של h על X כולו, שנסמן אותה ב-e(h), היא לכל היותר:

כלומר, אם יש קבוצה סופית של חוקים ומתוכם יש חוק שהוא טוב על המדגם, אזי בהסתברות מאוד גבוהה חוק זה הוא גם טוב על כל העולם.

1. Generalization bound כשיש טעות במדגם - נניח כי עבור פונקציה הטעות של h על S היא . אזי קיים כך שבהסברות לכל הפחות הטעות האמיתית של h על X, שנסמן אותה ב-e(h), היא לכל היותר:

כלומר אם יש קבוצה סופית של חוקים ומתוכם יש חוק שהטעות שלו על המדגם היא קטנה, אזי בהסתברות מאוד גבוהה הטעות שלו גם על העולם כולו היא קטנה.

ניתן לראות שיש חשיבות גדולה למספר החוקים בהיפותזה H. בסעיף הבא נלמד כיצד סופרים כמה חוקים בכל היפותזה.

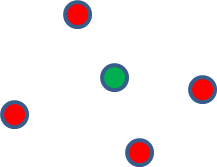
## VC Dimension

VC Dimension (Vapnik-Chervonenkis dimension) הוא מדד בתחום למידת מכונה המתאר את רמת כושר ההפרדה (הקיבולת, הסיבוכיות או העושר) של קבוצת חוקים שניתנת ללמידה באמצעות אלגוריתם למידה. מדד זה משמש, כמו שנראה במשפטים בהמשך, כדי לדעת את מספר החוקים בהיפותזה כלשהי, כדי שנוכל להציב ערך זה בנוסחאות שלמדנו בסעיף קודם. לפני שנסביר מהו מימד VC יש להסביר מהו "ניפוץ".

**ניפוץ (Shattering)** - בהינתן היפותזה H (קבוצה של חוקים/מסווגים) וקבוצה S המכילה n נקודות, נאמר כי H מנפצת את S אם יש חוק ב-H לכל סיווג אפשרי של S. כלומר, ב-H צריכים להיות חוקים שכל אחד מספק סיווג שונה על S.

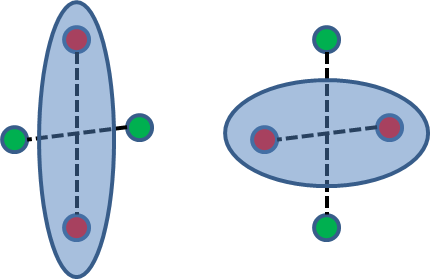
**מימד VC** - בהינתן היפותזה H, המימד VC של H הוא המספר המקסימלי של נקודות ש-H יכולה לנפץ עבור איזשהו סידור שלהם. במילים אחרות, אם H יכולה לנפץ איזשהו סידור של d נקודות אך לא יכולה לנפץ d+1 נקודות, אזי המימד VC של H הוא d. באופן דומה, כדי להוכיח שמימד VC של היפותזה H הוא d, יש להראות ש-H מסוגלת לנפץ **איזשהו סידור** של d נקודות (חסם תחתון) אך לא יכולה לנפץ d+1 נקודות (חסם עליון).

### דוגמאות

1. **מרווח חד-כיווני**: נתונים שורה של נקודות, כל חוק בהיפותזה זו הוא מיקום על שורה זו שבה כל הנקודות מצד ימין מסווגת 0 ומצד שמאל 1. המימד VC של מרווח חד-כיווני הוא 1.
2. **מרווח דו-כיווני**: כמו מרווח ח-כיווני, אלא שלכל מיקום אפשר להחליט איזה צד מסווג 0 ואיזה 1. מימד ה-VC של מרווח דו-כיווני הוא 2.
3. **קו דו-כיווני**: כל חוק בהיפותזה זו הוא קו שבו ניתן להחליט איזה צד מסווג 0 ואיזה 1. מימד ה-VC של מרווח דו-כיווני הוא 3.
4. **מלבן חד כיווני**: כל חוק הוא מלבן (בשני ממדים) שכל הנקודות בתוכו מסווגות 0, וכל הנקודות מחוץ מסווגות 1. מימד ה-VC של מלבן חד-כיווני הוא 4.

הוכחה: מלבן יכול לנפץ 4 נקודות (טריוויאלי), אך אינו יכול לנפץ 5 נקודות, מפני שעבור כל סידור של 5 נקודות לא נוכל לסווג את 4 הנקודות הכי קיצוניות לכל כיוון כ-0 ואת הנקודה החמישית כ-1.

1. **מעגל חד-כיווני**: כל חוק הוא מעגל (בשני ממדים) שכל הנקודות בתוכו מסווגות 0, וכל הנקודות מחוץ מסווגות 1. מימד ה-VC של מלבן חד-כיווני הוא 3.

הוכחה: מעגל יכול לנפץ 3 נקודות (טריוויאלי), אך אינו יכול לנפץ 4 נקודות. אם הנקודות מסודרות כך שיש נקודה אחת שמוכלת בצורה הקמורה של שלושת האחרות, אזי מעגל לא יכול לסווג את שלושת הנקודות החיצוניות 0 ואת הפנימית 1. אם הנקודות מסודרות בצורה כללית, אזי אם יש מעגל שיכול להכיל שתי נקודות מנוגדות לא יכול להיות שיש מעגל שמכיל רק את שני הנקודות האחרות (ראה תמונה).

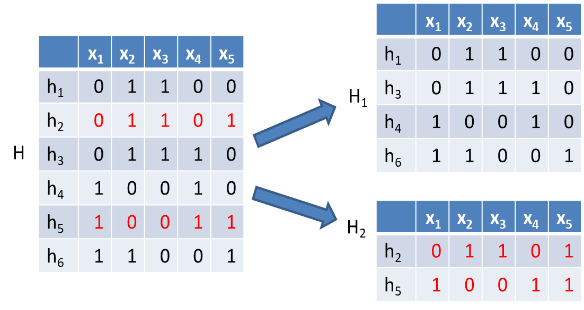
1. **כדור חד-כיווני**: כל חוק הוא כדור (בשלושה ממדים) שכל הנקודות בתוכו מסווגות 0, וכל הנקודות מחוץ מסווגות 1. מימד ה-VC של מלבן חד-כיווני הוא 4.

### משפט Sauer

בהינתן היפותזה H שהמימד VC שלה הוא d וקבוצה S עם n נקודות, נסמן את להיות כל הסיווגים השונים ש-H יכולה לתת ל-S. לפי משפט Sauer מתקיים:

כלומר נוכל לחשב חסם עליון למספר הסיווגים השונים ש-H יכול לתת ל-S. באמצעות חסם זה נוכל להציב ערך זה בנוסחאות Generalization Bound שלמדנו ב-PAC learning כדי לקבל את הטעות האמיתית של חוק כלשהו ב-H על S.

**מסקנה**: אם המימד VC קטן מ-n ויש חוק שהטעות שלו על המדגם קטנה, אזי בהסתברות גבוהה נקבל שהטעות של h על העולם כולו היא קטנה. ככל שהמימד VC קטן יותר כך הטעות על העולם קטנה יותר ונקבל חוק טוב יותר. במילים אחרות, למידה היא אפשרית אם החוק הוא פשוט, כלומר יש מעט חוקים.

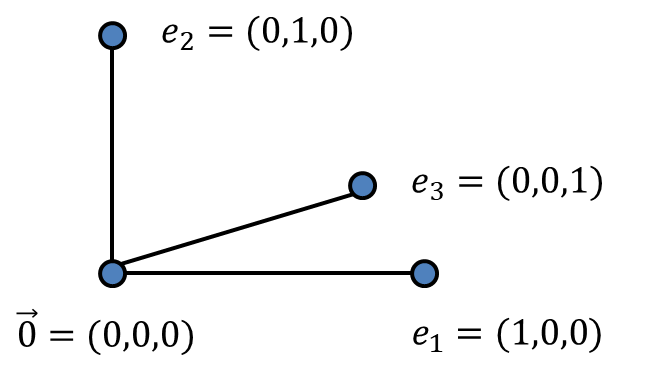


**הוכחת המשפט:** ניקח את כל החוקים ב-H שנותנים סיווג שונה על S, ונחלק אותם לשני קבוצות ו- בדרך הבאה: אם יש שני חוקים שמסווגים שונה **רק** את הנקודה האחרונה , אזי נעביר את החוק שסיווג את כ-1 ל-. כל השאר יהיו ב-. נשים לב שעבור הקבוצה מספר התיוגים השונים ב- ו- לא משתנה: , אולם הממד VC של הוא לכל היותר d-1: . כעת, נוכיח את משפט Sauer באינדוקציה על מספר הנקודות n ב-S.

* בסיס - עבור n=d יש לכל היותר סיווגים שונים, שאכן קטן יותר מ-.
* הנחה - נניח כי המשפט נכון עבור n-1.
* צעד - נוכיח עבור n. לפי הנחת האינדוקציה ומשפט פסקל מתקיים:

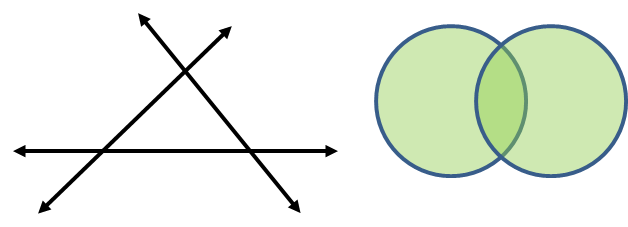
### משפטים חשובים

1. **מישור ב-d-1 ממדים**: כל חוק הוא מישור ב-d-1 ממדים בעולם של d ממדים . המישור מבדיל בין הנקודות המסווגות 0 לנקודות המסווגות 1. מימד ה-VC של מישור ב-d-1 ממדים הוא d+1.

הוכחה: מישור ב-d-1 ממדים יכול לנפץ d+1 נקודות. נסדר d נקודות במרחק של יחידה אחת מראשית הצירים, כך שכל נקודה נמצאת בציר אחר. נקודה נוספת נציב בראשית הצירים. עבור כל תת-קבוצה S של d+1 נקודות אלו יש מישור בעל d-1 מימדים שמכיל כל הנקודות בקבוצה זו ולא נקודות אחרות. עבור d+2 נקודות, לפי משפט רדון (Radon) כל קבוצה של d+2 נקודות ב- ניתנת לחילוק לשתי קבוצות זרות כך שהצורות הקמורות הנוצרות על ידן נחתכות. אין אף מישור שיכול לסווג את כל הנקודות מחוץ לחיתוך כ-0 ואת הנקודות בתוך החיתוך 1.

1. **מישור עם מרווח:** נסמן את רדיוס המעגל המקיף את כל הנקודות במדגם ב-R, ואת המרווח המינימלי בין המישור לנקודה כלשהי ב-. מספר החוקים המקסימלי ב-H על S הוא .

המסקנה העולה משני המשפטים שציינו עתה, היא שאם d קטן אזי מישור (חוק ליניארי) הוא מסווג מאוד טוב, אולם אם d מאוד גדול אזי נקבל באמצעות משפט Sauer חסם לא יעיל על . לכן במקרים כאלו מישור שהוא רחוק מן הנקודות במדגם הוא מסווג טוב. ככל שהמישור מתרחק מן הנקודות כך הטעות שלו על העולם תהיה קטנה יותר והמסווג יהיה טוב יותר.

1. **איחוד חוקים**: יהיו קבוצות חוקים עם אותו מימד VC , ויהיה איחוד כל קבוצות חוקים אלו, . אזי ניתן לתת חס עליון למימד VC של H:

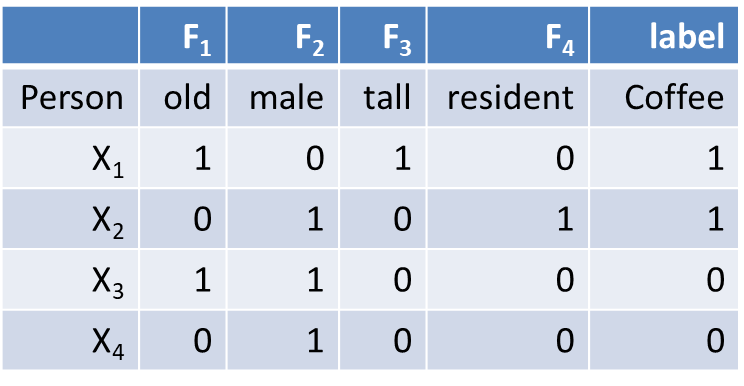
דוגמה לאיחוד של קבוצות חוקים הוא איחוד של מספר קווים או מספר מעגלים.

# Winnow & Perceptron

## אלגוריתם Winnow

זהו אלגוריתם שמטרתו למצוא מסווג ליניארי בינארי. נתונים מדגם של n אובייקטים כך שלכל אובייקט יש k תכונות כלשהן , ו-label המסווג את האובייקט כ-1 או 0 (המייצגים בדרך כלל תשובה לשאלת כן או לא). מטרת האלגוריתם היא למצוא וקטור W של k משקלים, כך שאם נכפיל וקטור זה בכל אובייקט x עם k תכונות (ולא רק אלו שנתונים לנו), יעזור לנו לסווג את x האם שייך למחלקה 1 או שייך למחלקה 0.

ההנחה המרכזית של אלגוריתם Winnow היא שמתוך כל k התכונות יש r תכונות בלבד שהן קריטיות והשאר אינן חשובות, כך שאם לאובייקט כלשהו יש ערך שונה מ-0 **בלפחות אחד** מהתכונות הקריטיות, אזי נסווג אותו כ-1, אך אם בכל התכונות הערך של האובייקט הוא 0 נסווג אותו כ-0. אם הנחה זו לא מתקיימת על המדגם האלגוריתם לא יעבוד. מטרת האלגוריתם היא אם כן לזהות את כל התכונות הקריטיות ולנפות את כל שאר התכונות. מכן שמו של האלגוריתם Winnow (לנפות).

דוגמה: בטבלה הבאה ניתן לראות שהתכונות ו- הן התכונות הקריטיות.

### תיאור האלגוריתם

האלגוריתם מאתחל את כל המשקלים ל-1, ולאחר מכן עובר בלולאה על כל האובייקטים. עבור כל אובייקט מכפיל אותו בוקטור W, אם הסכום שהתקבל גדול שווה ל-n מסווג אותו כ-1, אחרת מסווג אותו כ-0. לאחר סיווג של כל אובייקט בודקים את ה-label של האובייקט. אם סיווגנו נכון עוברים לאובייקט הבא, אך אם טעינו בודקים איזה סוג טעות זו. אם סיווגנו 1 אבל הוא באמת 0, משמע שכל התכונות בהן יש לאובייקט זה 1 הן אינן חשובות ולכן נאפס את המשקלים בהן. אך אם סיווגנו 0 והוא באמת ,1 נכפיל את כל המשקלים פי 2 בכל התכונות שבהן שונה מ-0 כדי שבאיטרציה הבאה יהיה יותר סיכוי שנסווגו כ-1. כך חוזרים על פעולה זו עד שהמודל שיתקבל מסווג נכון את כל המדגם.

1. initialize = 1.
2. For each object :

* if classify as 1
* else classify as 0
* if we classified incorrectly:

A - if we classify 1 but was 0:

Set for all

B - if we classify 0 but was 1:

Set for all

1. return step 2 until no mistake is found on all objects

### סיבוכיות

נוכיח כי מספר הטעויות המקסימלי שהאלגוריתם מבצע על המדגם הוא לכל היותר . כתוצאה מכך נגיע למסקנה שמספר הפעמים שחוזרים על שלב 2 הוא , ולכן הסיבוכיות של אלגוריתם Winnow היא .

הוכחה:

* בכל טעות מסוג B מכפילים משקל של לפחות תכונה קריטית 1 פי 2. אחרי טעויות שבהן הכפלנו משקל של תכונה קריטית פי 2 נקבל שהמשקל של תכונה זו לבדה גדול מ-n ולכן לא נוכל לטעות בה יותר. כיוון שיש r תכונות, טעויות מסוג B יקרו לכל היותר פעמים.
* בכל טעות מסוג A מפחיתים את סכום המשקולות ב-W בלפחות n, ואילו טעות מסוג B מוסיפה לכל היותר n לסכום המשקולות. כיוון שסכום המשקולות יכול להגיע לכל היותר ל- לפי החלק הראשון בהוכחה, אזי על A ניתן לחזור לכל היותר גם כן פעמים, כי לא ניתן לאתחל משקל לערך שלילי לפי האלגוריתם.

## אלגוריתם Perceptron

כמו Winnow מטרת אלגוריתם Perceptron היא למצוא מסווג ליניארי בינארי, אלא שבנוסף מנסה שהמרווח של המישור הליניארי, המשמש כמסווג, מן הנקודות במדגם יהיה כמה שיותר גדול. המניע לכך הוא, שכמו שלמדנו בעקרונות למידת מכונה, ככל שהמסווג רחוק יותר מן הנקודות כך הטעות שלו על העולם תהיה קטנה יותר והמסווג יהיה טוב יותר.

האלגוריתם מסתמך על תכונה באלגברה ליניארית שאם מכפילים נקודה במרחב בוקטור אורתוגונלי למישור מקבלים את המרחק של הנקודה מהמישור. אם הערך חיובי הנקודה בצד אחד של המישור, אם שלילי הנקודה בצד שני של המישור ואם 0 הנקודה על המישור.

### תיאור האלגוריתם

האלגוריתם עובר בסבבים על כל האובייקטים במדגם עד שמסווג נכון את כל האובייקטים. עבור כל סבב האלגוריתם שומר וקטור משקלים . בהתחלה מאתחל כל המשקלים ב- ל-0. עבור כל אובייקט, אם המכפלה שלו בוקטור משקלים נוכחי גדול מ-0 נסווג אותו כ-1, אחרת נסווגו כ-0. לאחר סיווג של כל אובייקט בודקים את ה-label שלו. אם סיווגנו נכון עוברים לאובייקט הבא, אך אם טעינו בודקים איזה סוג טעות זו. אם סיווגנו 1 אבל הוא באמת 0, נגדיר  *כדי שבסבב הבא יפחת הסיכוי שנסווג אותו 1*. אך אם סיווגנו 0 והוא 1, נגדיר  *כדי שבסבב הבא* יגדל הסיכוי שנסווגו כ-1. בכל סבב בו גילינו טעות ראשונה, מסיימים את הסבב ועוברים לסבב הבא.

1. set
2. for round
3. for each object :

* If classify as 1
* else classify as 0
* if we classified incorrectly:

A - if we classify 1 but was 0:

Set

B - if we classify 0 but was 1:

Set

exit round t (on first mistake)

1. if no mistakes this round, exit algorithm

### סיבוכיות

נוכיח כי מספר המקסימלי של סבבים עבור הפעלה של האלגוריתם Perceptron על מדגם S היא לכל היותר , כאשר הוא המרווח המקסימלי של מישור שעקבי על S מאיזושהי נקודה ב-S. מכאן שסיבוכיות האלגוריתם היא .

הוכחה:

נסמן ב- את הוקטור המאונך למישור עם מרווח אותו אנו רוצים למצוא.

1. *נוכיח כי מתקיים .*

*עבור טעות מסוג A אגן מתקיים: .* שהרי מחזיר את המרחק מהמישור שהוא לכל הפחות .

באותו אופן, עבור טעות מסוג B מתקיים: .

1. *נוכיח כי מתקיים: .*

*עבור טעות מסוג A מתקיים: . שהרי אנו מניחים שכל הנקודות* x *הן על מעגל היחידה או בתוכו, ולכן . בנוסף, כי אחרת לא היינו מסווגים את* x *להיות 1.*

באותו אופן, עבור טעות מסוג B מתקיים: .

כעת, נסמן את המספר המקסימלי של סבבים באלגוריתם Perceptron ב-. לפי טענה 1 יוצא שבכל סיבוב מוסיפים ל- לפחות , ולכן לאחר סבבים מתקיים: **.** ולפי טענה 2 יוצא שבכל סיבוב מוסיפים ל- לפחות 1, ולכן לאחר סבבים מתקיים: . מכאן שמתקיים:

# SVM - Support Vector Machines

## הסבר האלגוריתם

K-NN

## הסבר האלגוריתם

אלגוריתם KNN (k-Nearest Neighbors) הוא אלגוריתם המשמש כמסווג לא ליניארי, אך יכול גם לפתור בעיית רגרסיה. הרעיון המרכזי באלגוריתם זה, שבשלב האימון הוא זוכר מדגם מייצג של אובייקטים עם ה-label שלהם, ועבור כל אובייקט x חדש שרוצה לסווג, האלגוריתם צריך למצוא את k האובייקטים הכי קרובים אליו (k השכנים שלו), ואז לסווג את x אל המחלקה הנפוצה ביותר בקרב k השכנים הקרובים. שובר שוויון יכול להיות סכום המרחקים הכי קטן או השכן הכי קרוב.

כדי למצוא את k השכנים הקרובים צריך להגדיר ל-K-NN פונקציית מרחק. יש שני סוגים מרכזיים של פונקציית מרחק:

1. L1 Distance (Manhattan) - זהו סכום ההפרשים בערך מוחלט של כל f התכונות של שני האובייקטים.
2. L2 Distance (Euclidean) - זהו שורש הסכום של ההפרשים בריבוע של כל f התכונות של שני האובייקטים.
3. L - זהו ההפרש המקסימלי מבין כל f התכונות של שני האובייקטים.

אלגוריתם זה נחשב לפשוט מאוד. מצד אחד זמן האימון שלו הוא , אך הזמן שלוקח לו לסווג אובייקט הוא . זהו זמן ריצה גרוע מאוד. אנו מעדיפים שזמן האימון יהיה ארוך, כיוון שמבצעים אותו פעם אחת בלבד, ואילו זמן הסיווג יהיה מהיר.

## קביעת היפר-פרמטרים

ההיפר-פרמטרים של אלגוריתם K-NN הם הערך k ופונקציית המרחק. k צריך להיות אי-זוגי כדי למנוע מקרים של שוויון בין מחלקות. כדי לוודא אילו ערכים הכי טובים יש לחלק את הדאטה כך שיהיה חלק של test שבו נבדוק את איכות המודל הסופי. יש כמה דרכים כיצד לחלק את הדאטה ולקבוע את ההיפר-פרמטרים.

* **train-validation-test**. נקבע ערכים שרירותיים ל-k, נמצא k שכנים הכי קרובים מה-train ונעריך את המודל על ה-validation. ניקח את ערכי ה-k הכי טובים על ה-validation, ולבסוף נבדוק את איכות המודל על ה-test.
* **Cross-Validation** - נחלק את הדאטה למספר חלקים שווים הנקראים folds ול-test. עבור כל ערך של k שנרצה לבדוק נעבור בלולאה כמספר ה-folds. בכל איטרציה fold אחר ישמש כ-validation ושאר ה-folds ישמשו כ-train. רמת הדיוק של ערך k יהיה ממוצע רמות הדיוק של כל האיטרציות. נבחר את ה-k שבו ממוצע אמת הדיוק היה מקסימלי. לבסוף נבדוק את איכות המודל על ה-test. שיטה זו שימושית בעיקר שאין הרבה דאטה.

K-NN עוזר להפחית רעש שכן לעומת 1-NN שכן מתחשבים בכמה נקודות ולא אחד.

## K-NN לרגרסיה

בהינתן דוגמה חדשה, האלגוריתם מחזיר ערך מאפיין, לדוגמה, ממוצע ערכים של ה-label ב-k השכנים הקרובים ביותר.

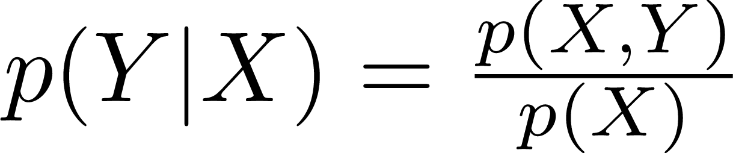
Naïve Bayes

## הסבר האלגוריתם

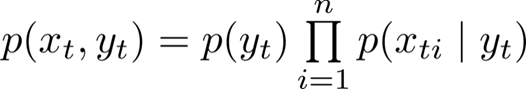
Naïve Bayes זהו אלגוריתם המייצר מסווג (Classifier) שבו הסיווג למחלקות אינו מתחשב בתלות שבין התכונות (ולכן נקרא נאיבי). מסיבה זו הוא יכולה לעבוד גם עבור אובייקטים שבהם **חסרים תכונות**. נסמן את כל סוגי המחלקות ב-. הרעיון באלגוריתם זה הוא לחשב עבור כל אובייקט שאנו רוצים לסווג למחלקה, מהי ההסתברות שהוא שייך לכל מחלקה, ולבחור את המחלקה עם ההסתברות הגבוהה ביותר, נסמן אותה ב-. במילים אחרות נרצה לחשב:

.

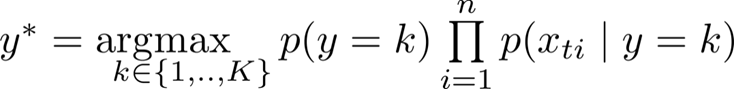
כאשר הוא ההסתברות שאובייקט שייך למחלקה .



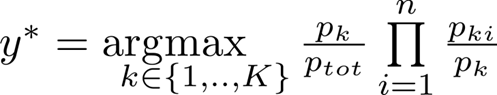
* לפי חוק בייס מתקיים: .
* ולפי נוסחת ההסתברות השלימה מתקיים:
* אנו מניחים באלגוריתם זה שתמיד אין תלות בין תכונות שונות. אנו מניחים זאת למרות שזה כמובן לא תמיד נכון, מפני שזה יכול להוביל לשגיאות, וגם כי בפועל זה מביא תוצאות טובות. לכן נקבל:



=

* **משילוב המשוואות נקבל את נוסחת Naïve Bayes:

כאשר היא ההסתברות שאובייקט כלשהו שייך למחלקה , ו-היא ההסתברות שהתכונה i ב- שייכת למחלקה .

כדי לפשט את המודל, שלא נצטרך לחשב את כל הנתונים בנפרד עבור כל אובייקט, נחשב את שלושת הערכים הבאים כדי לזרז את החישוב. הראשון משותף לכל המחלקות , ואילו השניים האחרים מיוחדים לכל מחלקה :

* - מספר האובייקטים שכבר נתונים לנו.
* - סכום כל האובייקטים השייכים למחלקה k.
* - לכל תכונה i ולכל מחלקה k, נחשב בכמה אובייקטים ממחלקה k התכונה i מופיעה.

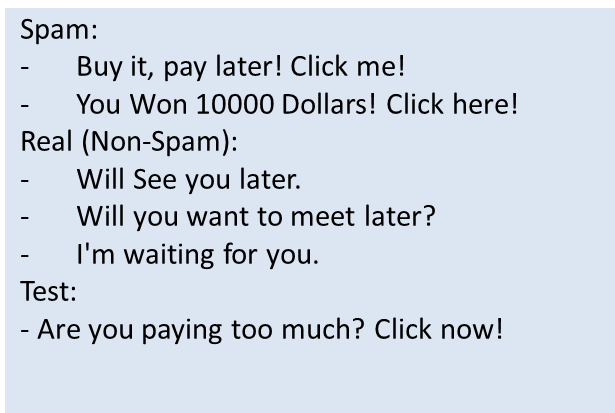
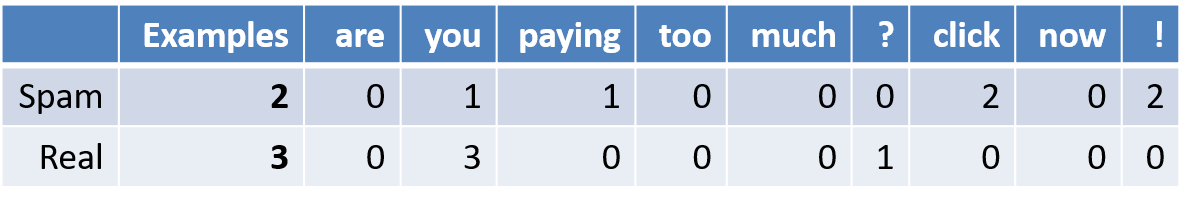
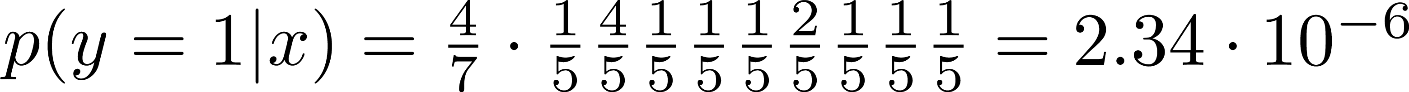
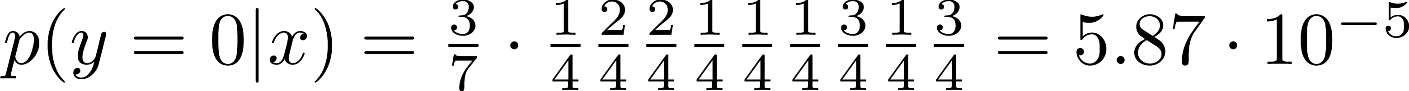
כעת כל שנותר הוא לחשב עבור כל אובייקט:

הערה - כאשר נתונים המון אובייקטים, כיוון שמכפילים הסתברויות, יכול להיות מצב שכל הערכים יצאו קטנים מיכולת הדיוק של המחשב, ולכן כל ההסתברויות יהיו 0. מסיבה זו, בפועל בדרך כלל במקום להכפיל הסתברויות, נפעיל פונקציית log לכל הסתברות ונסכום את כל ערכים שהתקבלו.

## Laplace's Smoothing

בעיה נוספת שיכולה להתעורר מהנוסחה שתיארנו היא כאשר נקבל שהסתברות עבור עמודה מסוימת היא 0. דבר זה יכול לאפס את המשוואה. כדי לפתור בעיה זו נוסיף לנתונים עוד אובייקט לכל מחלקה, שבו מופיעים כל הערכים האפשריים. פעולה זו תוסיף k ל-, 1 לכל , 1 ל- במונה מחוץ למכפלה, ו-2 ל- של המכנה במכפלה.

**דוגמה:** בדיקה האם הודעה היא ספאם או אמיתית



## דוגמה ב-Python

נמשיך את הדוגמה מסעיף קודם על סיווג הודעות אמיתיות לספאם. נניח שהמידע שמור ב-RDD בתור (הודעה, מחלקה).

input\_data = sc.parallelize([("hello there", 0), ("hi there", 0), ("go home", 1), ("see you",1), ("bye to you", 1)])

#We need to find , and all :

>>> pk = input\_data.map(lambda (message, cls): (cls, 1)).reduceByKey(lambda a,b: a+b).collectAsMap()

>>> ptot = sum(pk.values())

>>> pki = input\_data \

.flatMap(lambda (message, cls): list(set([(cls,w) for w in message.split()]))) \

.map(lambda (cls, word): ((cls,word), 1)) \

.reduceByKey(lambda a,b: a+b).collectAsMap()

>>> import numpy as np

>>> query = "hello hi"

>>> class\_probs = [pk[k]/float(ptot)\*np.prod (np.array([pki.get((k,i),0)/float(pk[k]) for i in query.split()])) for k in range(0,2)]

>>> print(class\_probs)

[0.10000000000000001, 0.0]

>>> y\_star = np.argmax(np.array(class\_probs))

>>> print(y\_star)

[0]

Linear Regression

## הסבר האלגוריתם

זהו אלגוריתם שפותר בעיית רגרסיה, כלומר מחזיר ערך מטווח רציף ולא בדיד. רגרסיה ליניארית היא שיטה מתמטית המאפשרת לנו לבנות פונקציה ליניארית, כך שבהינתן אובייקט חדש x, שאנו יודעים את **כל** k התכונות שלו, נוכל לנבא מהו הערך y עבור x.

כל מייצג את המשקל (weight) שאנו נותנים לכל יחידה מהתכונה ה-i, ו-b הוא ערך מינימלי כלשהו (bias), שמטרתו להוסיף כוח חישוב שאינו תלוי בתכונות. הסיבה שאנו רוצים דווקא פונקציה שהיא ליניארית היא מפני שאנו מחפשים חוק פשוט, שלפי משפט Sauer הוא חוק טוב.

הרעיון המרכזי הוא שהפונקציה הליניארית שאותה אנו מחפשים היא הפונקציה שסך הסטיות של כל m האובייקטים בדאטה ממנה הוא מינימלי. ככל שיש לנו יותר מידע, כלומר יש בידינו המון אובייקטים נתונים, כך יכולות הניבוי של המודל יהיו טובים יותר. אמנם יש איזשהו גבול של מספר אובייקטים שכבר לא ישפר את המודל.

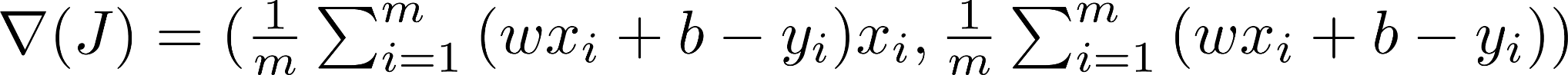
נתחיל עם רגרסיה ליניארית פשוטה שבה מספר התכונות הוא , ולאחר מכן נלמד כיצד לבנות משוואה עבור מודל עם k תכונות.

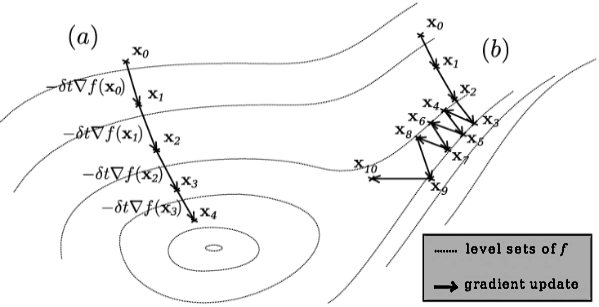
## רגרסיה ליניארית פשוטה

כאשר לכל אובייקט יש תכונה אחת בלבד, אנו מחפשים פונקציה פשוטה מהצורה , שבה נוכל להציב את התכונה של x ולקבל את הערך הצפוי y אותו אנו רוצים לנבא. w ו-b הם קבועים שבהמשך נלמד כיצד לחשב אותם. כאמור לעיל, אנו מחפשים פונקציה שעבור סכום המרחקים של הערך של כל אובייקט מהערך שהפונקציה צופה לו הוא מינימלי. ניתן לתאר את המרחק בערך מוחלט או בהעלאה בריבוע של ההפרש. נבחר בהעלאה בריבוע מפני שהוכח הסתברותית שמקבלים תוצאה טובה יותר על ידי העלאה בריבוע, וגם בגלל שערך מוחלט הוא לא גזיר ויכול להיות גם לא רציף. בנוסף, נחלק ב-2m כדי לקבל סטייה ממוצעת בריבוע (החלוקה ב-2 כי מצטמצמת עם הנגזרת). אם כן, מטרתנו היא למצוא w ו-b שמצמצמים את ה-Loss Function, שנסמן אותה , והיא:

דרך אחת לעשות זאת היא לגזור את הפונקציה ולהשוות ל-0, אמנם עבור מודלים מסוימים שבהם יש המון מידע וכן כהכנה לשיטה הבאה שנלמד, Logistic Regression, משתמשים בשיטת Gradient Descent.

## Gradient Descent

וקטור הגרדיאנט זהו וקטור שמייצג את הנגזרת של פונקציה עם מספר משתנים, ומסומן ב-. בכל כניסה בוקטור נמצא הנגזרת של הפונקציה לפי משתנה אחר. גודל הוקטור כמספר המשתנים בפונקציה. וקטור הגרדיאנט עבור פונקציית ה-Loss ב-Linear Regression הוא:

הרעיון בשיטת ה- Gradient Descentהוא להתחיל בערכים שרירותיים כלשהם של w ו-b, ובלולאה לשנות ערכים אלו בצעדים של קצב למידה ביחס לערך הגרדיאנט לפי ערכי w ו-b הנוכחיים, עד שמגיעים למצב של התכנסות (convergence). התכנסות זה מצב שבו הצבתם של w ו-b הנוכחיים בוקטור הגרדיאנט ייתן לנו את וקטור ה-0 או שואף לכך. כאשר מגיעים למצב של התכנסות מצאנו את ערכי w ו-b המינימליים.

חשוב ש- לא יהיה גדול מדי, שכן אז עלולים לפספס את המינימום ובצעד חזור שוב לפספס, וכך אף פעם לא להגיע למינימום. לכן צריך ש- יהיה קטן יחסית, בדר"כ באזור 0.001. על בסיס שיטה זו בנויים שיטות למידה רבות כמו רגרסיה לוגיסטית ולמידה עמוקה בכללותה.

האלגוריתם שבו נשתמש כדי למצוא ערכי w ו-b ברגרסיה ליניארית עבורם נקבל מינימלי הוא:

* בחר w ו-b רנדומליים כלשהם (בדרך כלל 0 לשניהם).
* קבע איזשהו שהוא גודל הצעד על w ו-b.
* בלולאה עד שמשיגים התכנסות (convergence) או מספר סופי של צעדים:
* חשב את .
* חשב את .
* עדכן את w להיות .
* עדכן את b להיות .
* החזר את w ו-b.

## דוגמה ב-Python

דוגמה לחישוב משוואה שתנבא מחיר של פלאפון גלאקסי S5, כאשר ידוע מחירם של גרסאות אחרות של גלאקסי. בנוסף, תוכנית זו תדפיס את ערכי רכיבי הגרדיאנט של w ו-b כל 200 איטרציות, ולבסוף תדפיס את המחיר המנובא.

import numpy as np

galaxy\_data = np.array([[2,70],[3,110],[4,165],[6,390],[7,550]]) #Galaxi versions and their price

w = 0

b = 0

alpha = 0.01

for iteration in range(10000):

deriv\_b = np.mean(1\*((w\*galaxy\_data[:,0]+b)-galaxy\_data[:,1]))

deriv\_w = np.mean(galaxy\_data[:,0]\*((w\*galaxy\_data[:,0]+b)-galaxy\_data[:,1]))

b -= alpha\*deriv\_b

w -= alpha\*deriv\_w

if iteration % 200 == 0 :

print("it:%d, grad\_w:%.3f, grad\_b:%.3f, w:%.3f, b:%.3f" %(iteration, deriv\_w, deriv\_b, w, b))

print("Estimated price for Galaxy S5: ", w\*5 + b)

## רגרסיה ליניארית מרובה

כאשר יש k תכונות לכל אובייקט ולא רק תכונה אחת, השיטה מאוד דומה אלא שכעת כל אובייקט אינו ערך בודד אלא וקטור , כאשר כל מייצג את התכונה ה-j של האובייקט . וכן w כבר לא יהיה ערך בודד אלא וקטור , כאשר כל מייצג את המשקל שאנו נותנים לכל יחידה מהתכונה j. הוא למעשה מישור שהמכפלה שלו ב- היא המרחק של האובייקט מהמישור. מטרתנו היא למצוא ו-b שמצמצמים את , כך שמרחק כל האובייקטים בדאטה מהמישור הוא מינימלי.

ישנם וריאציות שמוסיפים תכונה נוספת (אינדקס 0) ל-k התכונות שתייצג את b וכך מפשטים את המשוואה. בדרך כלל לא נעשה זאת אך יש כאלו שכן וצריך להכיר את זה. לשם כך צריך להגדיר בכל האובייקטים: , וכן . כעת המטרה היא לצמצם את:

האלגוריתם שתיארנו לעיל נשאר אותו הדבר אלא שכעת בוקטור הגרדיאנט יש k+1 כניסות, 1 לכל תכונה ועוד b. כל הכניסות נראות אותו הדבר כמו בוקטור הגרדיאנט שבנינו לעיל. בכל איטרציה של הלולאה, עבור כל , יש לחשב את הנגזרת החלקית שלו ולהציב את , ואז ולעדכן . אותו הדבר בכל איטרציה גם עבור b.

## איכות המודל

כדי לבדוק אם מודל רגרסיה ליניארית איכותי, נשווה בין המודל שלנו למודל שתמיד מנבא את הערך הממוצע של כל ערכי ה-Y (labels). ההשוואה תהיה בין הערכים המוחזרים מפונקציית ה-Loss הסופי של שני המודלים. כאמור לעיל, הערך המוחזר מפונקציית Loss מייצג את הסטייה הממוצעת של ניבוי המודל מ-Y. ככל שהסטייה של המודל שלנו קטנה מהסטייה של מודל שתמיד מנבא ממוצע כך המודל שלנו יותר טוב. ההשוואה האמיתית צריכה להיות כאשר הסטייה של המודל שלנו מחושבת על ה-test ואילו סטיית המודל שתמיד מנבא ממוצע מחושבת על ה-train.

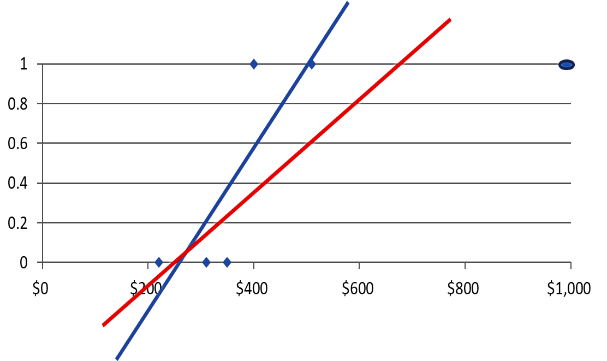
## סוגי רגרסיה נוספים

עבור דאטה שבו הסיווג תלוי בגורם נוסף של זמן, כמו שהסיווג הוא מחזורי או תלוי ברצך של זמן (time series), יש מודלים נוספים לרגרסיה כמו ARMA ו-ARIMA המתחשבים בנתונים אלו.

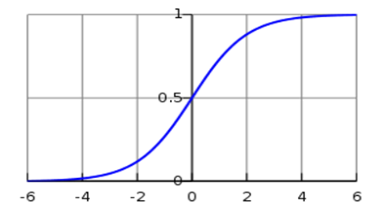
# Logistic Regression

## הסבר השיטה

רגרסיה לוגיסטית זוהי שיטה המנסה להתאים את הרגרסיה הליניארית כדי שתוכל לפתור בעיית Classification, כלומר במקום לנבא ערך לאובייקט נרצה לסווג אותו למחלקה מתאימה. בשלב ראשון נסביר כיצד לבצע התאמה זו עבור בעיית סיווג של שתי מחלקות. בעיות אלו הן לרוב בעיות שבאות לענות על שאלת כן או לא. לדוגמה, האם אדם הוא מועסק או לא או האם לקוח יבצע רכישה או לא. בהמשך נלמד כיצד לבצע את ההתאמה עבור בעיות סיווג של מספר מחלקות.



המחשבה הראשונית היא לייצג את שתי המחלקות באמצעות 0 ו-1, כאשר 0 התשובה היא לא ו-1 היא כן. במצב כזה נוכל להפעיל את שיטת הרגרסיה הליניארית על אובייקט x, כמו שלמדנו לעיל, ואזי אם הערך y שהתקבל קרוב יותר ל-1 מאשר ל-0 נעריך ששייך למחלקה 1, אחרת נסווג אותו למחלקה 0. אמנם בשיטה כזו במקרים שבהם יש אובייקט עם ערך מאוד גבוה או מאוד נמוך ביחס לקבוצה של אובייקטים שהוא נמצא איתם באותה מחלקה, כל המשוואה הליניארית תינטה אל אותו אובייקט. טעות זו יכולה להוביל לשגיאות חמורות.



לכן במקום פונקציה ליניארית נרצה למצוא פונקציה שבה סך הסטיות של כל m האובייקטים ממנה הוא מינימלי אך גם תצמצם את הפגיעה באיכות המשוואה במקרי קיצון שתיארנו. פונקציה כזו היא Sigmoid Function, שבה ככל שערכי ה-x גבוהים היא הולכת ומתקרבת ל-1, וככל שערכי x הולכים וקטנים היא מתקרבת ל-0.

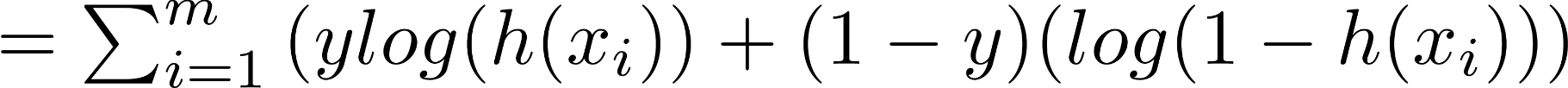
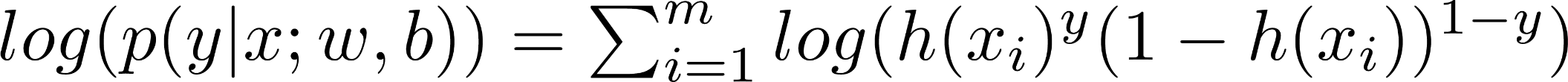
לאחר שנמצא את w ו-b באמצעות שיטת Gradient Descent, נציב אותם בפונקציית סיגמואיד h(x). כל אובייקט x שנרצה לסווג אותו למחלקה נציב אותו בפונקציה שהתקבלה. הערך שיחזור מהפונקציה הוא מה ההסתברות שהאובייקט שייך למחלקה 1, . אם ערך זה גדול מ-0.5 נעריך ששייך למחלקה 1, אחרת נסווג אותו למחלקה 0.

### חישוב פונקציית Loss

כאמור, *ההסתברות שהחיזוי עבור אובייקט יהיה 1 או 0 הוא*  ו- *בהתאמה. מכאן שההסתברות שהחיזוי עבור אובייקט יהיה הוא: .*

נרצה לחשב עבור חלק ה-train בדאטה מה ההסתברות שהחיזוי שלנו עבור כל אובייקט יהיה שווה בדיוק לדאטה. לשם כך נצטרך להכפיל את החישוב לעיל עבור כל אובייקט . נרצה כמובן למקסם ערך זה. .

אם נפעיל פונקציית log על הביטוי לעיל ונשתמש בחוקי log, נוכל לקבל סכום במקום מכפלה.



כעת כיוון שמקובל למזער ולא למקסם נוסיף מינוס בהתחלה. בנוסף, נחלק ב-m כדי לקבל סטייה ממוצעת לכל אובייקט, ולא שאם נגדיל מספר האובייקטים אזי גם הסטייה תגדל. סך הכל פונקציית ה-loss אותה נרצה למזער עבור w ו-b היא:

נזכיר שאם לכל אובייקט יש k תכונות ולא רק תכונה אחת, אזי כל אובייקט אינו ערך בודד אלא וקטור , כאשר כל מייצג את התכונה ה-j של האובייקט . וכן w כבר לא יהיה ערך בודד אלא וקטור , כאשר כל מייצג את המשקל שאנו נותנים לכל יחידה מהתכונה j.

### חישוב הגרדיאנט

נוכל למצוא w ו-b מינימליים באמצעות Gradient Descent. וקטור הגרדיאנט עבור נראה אותו דבר, אלא שיש להחליף את הפונקציה הליניארית בפונקציית סיגמואיד . וקטור הגרדיאנט יראה כך:

האלגוריתם שבו נשתמש כדי למצוא ערכי w ו-b עבורם נקבל מינימלי זהה לאלגוריתם שלמדנו ברגרסיה ליניארית:

* קבע איזשהו שהוא גודל הצעד על w ו-b.
* בלולאה עד שמשיגים התכנסות (convergence):
* חשב את , *כאשר ב-*מציבים את w ו-b הנוכחיים.
* חשב את , *כאשר ב-*מציבים את w ו-b הנוכחיים.
* עדכן את w להיות .
* עדכן את b להיות .
* החזר את w ו-b.

## השוואה ל- Naïve Bayes

Naïve Bayes הוא מודל יוצר (generative model) שבו מחשבים עבור כל מחלקה y מה ההסתברות שאובייקט x שייך למחלקה זו , וכן מה ההסתברות שאובייקט כלשהו שייך למחלקה y . לעומת זאת, Logistic Regression הוא מודל מפלה (discriminative model) שבו מחשבים רק מה ההסתברות שאובייקט x שייך למחלקה כלשהי y *. בדרך כלל מודל מפלה הוא יעיל יותר מאשר מודל יוצר.*

## דוגמה ב-Python

יש לנו בסיס נתונים של אנשים ואנחנו רוצים לשלוח להם הצעות עבודה, אך לשם כך עלינו להעריך מי מועסק ומי לא. אנו מכירים מידע זה רק על חלק קטן מהנתונים. ברצוננו לבנות Classifier הקובע אם משתמש מועסק או לא, בהתבסס על מגדר (0-זכר, 1-נקבה), גיל ושנות הניסיון של המשתמש.

import numpy as np

data\_x = np.array([[1,28,4],[1,60,34],[1,25,3],[0,54,20],[0,24,2],[0,39,12],[0,30,4],[1,36,10],[1,26,1],[0,44,9]])

data\_y = np.array([1,1,1,1,1,1,1,0,0,0]) # data\_x is the gender, age, and experience. data\_y 1 means employed

def h(x,w,b): # sigmoid function

return 1 / (1+np.exp(-(np.dot(x,w) + b)))

w = np.array([0.,0,0])

b = 0

alpha = 0.001

for iteration in range(100000):

deriv\_b = np.mean(1\*( (h(data\_x,w,b)- data\_y)))

deriv\_w = np.dot((h(data\_x,w,b) - data\_y), data\_x)\*1/len(data\_y)

b -= alpha\*deriv\_b

w -= alpha\*deriv\_w

print("User [1, 49, 8] probability of working: ", h(np.array([[1, 49, 8]]),w,b))

## ריבוי מחלקות

כאשר יש מחלקות שאנו רוצים לסווג אליהן, ולא רק 2, משתמשים בשיטה שנקראת Softmax כדי לסווג אובייקט x למחלקה מתאימה. שיטה זו מתוארת בפרק הבא.

## איכות המודל - F-Measure

כדי לוודא את איכות המודל נבנה Confusion Matrix על התוצאות שנלקחו מהרצת המודל על ה-validation או על ה-test. במטריצה זו נציין כמה אובייקטים המודל ניבא נכון וכמה לא ניבא נכון מתוך החיוביים ומתוך השליליים. המטריצה משמאל מתאימה לבעיית Classification עם שתי מחלקות. נגדיר:

* Recall – מהו האחוז שהגדרנו שהם מתאימים למחלקה מבין כל אלו שאכן מתאימים למחלקה. בציור לעיל נחשב:

.

* Precision – מהו האחוז של המתאימים למחלקה מבין אלו שהמודל זיהה שהם מתאימים למחלקה. בציור לעיל נחשב:

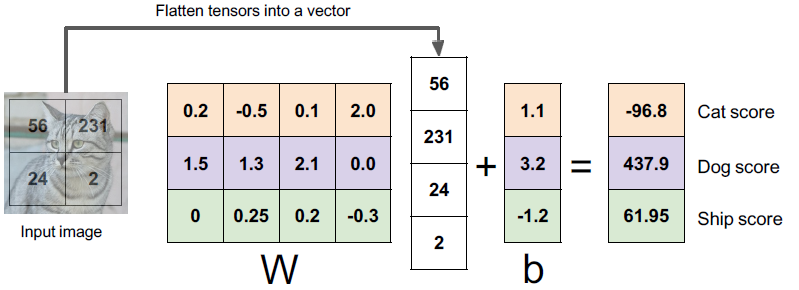
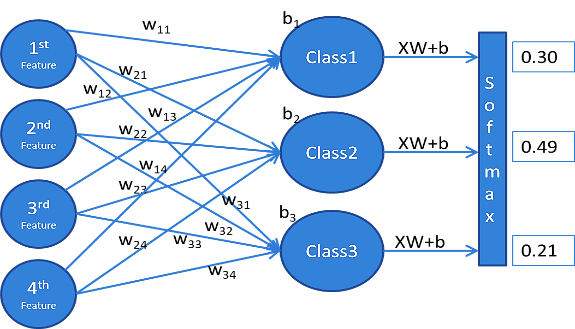
השיטה שבה נמדוד את האיכות של המודל נקראת F-Measure המחשבת ממוצע הרמוני באמצעות שני הערכים שהגדרנו לעיל. ככל שערך זה גדול יותר כך המודל מדויק יותר.

# Softmax

## הגדרה

כאשר ישנם k מחלקות שאנו רוצים לנבא עבור אובייקט לאיזה מהן הוא שייך, נשתמש בשיטת Softmax כדי לעשות זאת. בשיטה זו בונים מודל כמו Logistic Regression, אלא שכאן המודל ינבא מה ההסתברות שהוא שייך לכל אחד מ-k המחלקות. כלומר, לכל מחלקה i, המודל יחזיר מה ההסתברות שאובייקט uשייך ל-i, כך שסכום כל ההסתברויות הוא בדיוק 1. נשייך את u למחלקה עם ההסתברות הגבוהה היותר.

כיצד מודל Softmax עובד? נסמן את הדאטה X, את מספר האובייקטים בדאטה n, מספר התכונות f ומספר המחלקות k. במקום ש-W יהיה וקטור באורך f ו-b ערך בודד, כמו ב-Logistic רגיל, **מייצרים W שהוא מטריצה מסדר גודל של** , **כלומר מספר השורות הוא מספר התכונות ומספר העמודות הוא מספר המחלקות, ו-b הוא וקטור באורך k**. כעת הכפלה של הדאטה ב-W והוספת b ייתן לנו מטריצה בגודל , כלומר יש k ערכים לכל אובייקט. k ערכים אלו נקראים “logits”. כדי לקבל את k ההסתברויות לכל אובייקט, נחלק כל אחד מ-k הערכים בסכום k הערכים.



לסיכום, החישוב שהמודל יבצע כדי לקבל את וקטור ההסתברות y המכיל את ההסתברות שכל אובייקט שייך למחלקה i הוא:

פונקציה זו נקראת פונקציית Softmax. נשים לב שכאשר Softmax הוא Logistic Regression רגיל, לכן Softmax הוא הכללה של Logistic Regression. ב-TensorFlow נוכל להשתמש בפקודות הבאות כדי לקבל את k ההסתברויות לכל אובייקט ב-X:

W = tf.Variable(tf.zeros([features,categories]))

b = tf.Variable(tf.zeros([categories]))

y = tf.nn.softmax(tf.matmul(X, W) + b)

## פונקציית Loss - Cross Entropy

בשיטה זו פונקציית ה-Loss היא האנתרופיה, המייצגת שיעור אי-ודאות של מאורע. חישוב האנתרופיה למאורע X שיש לו מספר אפשרויות עם הסתברות שונה לכל אפשרות הוא:

אמנם כאן ההסתברויות אינן בהכרח תואמות את ההתפלגות האמיתית, לכן אנו נחשב cross entropy, שבה **מצליבים** את ההסתברויות שקיבלנו עם ההסתברויות האמיתיות. בנוסף, נרצה לחשב את האנתרופיה הממוצעת לכן נחלק במספר האובייקטים n. חישוב פונקציית ה-Loss עבור כל מחלקה i הוא:

כאשר זוהי ההסתברות האמיתית שאובייקט x שייך למחלקה i, ו- זה ההסתברות ש-x שייך למחלקה i לפי המודל שלנו. נשים לב שכיוון שאנו יודעים בוודאות לאיזה מחלקה שייך כל אובייקט, היא 1 עבור אותה מחלקה ו-0 עבור שאר המחלקות. לכן חשוב לשמור את ערכי האמת לכל אובייקט בוקטור בגודל k בו יש 1 בכניסה i אם האובייקט שייך למחלקה i, ו-0 בכל שאר הכניסות (one-hot encoding). למעשה, החישוב של פונקציית ה-Loss עבור כל מחלקה i הוא אם כן:

### בעיות חישוב

אם פונקציית ה-Softmax תחזיר עבור אובייקט במחלקה מסוימת 0 או 1 אזי נקבל בחישוב פונקציית Loss ערך NaN (Not a Number). מאוד קל להגיע למצב זה שכן יכול להגיע בקלות לאינסוף או 0 ואז הניבוי יהיה 0 או 1. כדי לפתור זאת נשתמש בפונקציה מותאמת של TensorFlow שמחשבת בבת אחת את פונקציית ה-Softmax עם פונקציית ה-Loss, ואז ניתן להפריד בין ה-logits מהחזקה של e ולקבל חישוב מדויק שלא מגיע לטעות כזו. פונקציה זו היא:

loss = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits\_v2(y\_, z))

## תבנית למודל Softmax

categories = 10

x = tf.placeholder(tf.float32, [None, features])

y\_ = tf.placeholder(tf.float32, [None, categories])

W = tf.Variable(tf.zeros([features,categories]))

b = tf.Variable(tf.zeros([categories]))

z = tf.matmul(x, W) + b

loss = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits\_v2(y\_, z))

update = tf.train.GradientDescentOptimizer(0.001).minimize(loss)

…

# שיפורים כלליים למודל למידת מכונה

## רגולריזציה

במודלים רבים נותנים "משקלים" לכל תכונה. רגולריזציה זוהי שיטה שבה אנו נותנים "עונש" למודל על כל משקל שהוא יקבע. מטרתנו היא להוריד כוח מהמודל ולגרום לו להיות זהיר יותר בנתינת משקלים גדולים מדי לתכונות אלא אם כן זה מאוד משתלם. המטרה כאן היא למנוע מהמודל להיכנס ל-overfitting, כלומר שהמודל טוב מדי על ה-train אך פחות טוב על ה-validation. נעשה זאת על ידי הוספה לפונקציית Loss סכום כלשהו התלוי במשקלים שהמודל קבע במצב נוכחי. ככל שהמשקל יהיה גדול יותר כך ה-Loss יגדל, ולכן יהיה יותר זהיר. יש שתי שיטות מרכזיות לרגולריזציה. שתי השיטות מתבססות על נוסחאות p-Norm ועל המשקלים W.

Spam:

* Buy it, pay later! Click me!
* You Won 10000 Dollars! Click here!

Real (Non-Spam):

* Will See you later.
* Will you want to meet later?
* I am waiting for you.

Test:

- Are you paying too much? Click now!

### Lasso

בשיטה זו מוסיפים לפונקציית Loss את סכום כל המשקלים בערך מוחלט כפול קבוע . הוא היפר-פרמטר, הנקרא “regularization strength”, שיש לבצע מספר ניסויים כדי למצוא את הערך הטוב ביותר עבורו.

Lasso מאוד אגרסיבית עבור תכונות עם משקלים קטנים ועלולה לגרום למודל לאפס את המשקלים עליהם כך שלא ישפיעו כלל. בעייתיות נוספת של שיטה זו היא שקשה לגזור ערך מוחלט. יש לכך מספר פתרונות:

1. להשתמש Coordinate Gradient Descent. לא ממומש ב-TensorFlow.
2. להשתמש באופטימייזר Adam.
3. להפחית את גודל הצעד ב- Gradient Descentככל שמתקדמים עם הצעדים - alpha\*=0.99.

### Ridge

בשיטה זו מוסיפים לפונקציית Loss את סכום כל המשקלים בריבוע חלקי 2, ומכפילים בקבוע .

מחלקים ב-2 כדי שיצטמצם עם הנגזרת. Ridge תגרום למודל להפחית משקולות גבוהים אך לא תאפס את הנמוכים, אלא תגרום למשקלים להיות מבוזרים פחות או יותר בצורה אחידה. משתמשים ברגולריזציה זו כאשר רוצים לגרום למודל להתחשב גם בתכונות פחות חשובות. ב-TensorFlow ניתן להשתמש בפונקציה המובנית tf.nn.l2\_loss(W) שמבצעת את אותה פעולה.

### Elastic Net

שילוב של Lasso עם Ridge.

## 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.