****

**MACHINE LEARNING PROJECT**

**part 2**

**המחלקה להנדסת תעשייה וניהול בן גוריון**



June 30, 2020

חאלד סובח

שירה בן צבי

תוכן עניינים

[הכנת הנתונים, הערות ודגשים 3](#_Toc44581069)

[Decision Trees 3](#_Toc44581070)

[כיוון פרמטרים 3](#_Toc44581071)

[Neural Networks 6](#_Toc44581072)

[כיוון פרמטרים 6](#_Toc44581073)

[K-Means 8](#_Toc44581074)

[שיטת אשכול נוספת 9](#_Toc44581075)

[השוואה בין מודלים 10](#_Toc44581076)

[המודל הנבחר 11](#_Toc44581077)

[Random Forest 11](#_Toc44581078)

[Gradient Boosting 12](#_Toc44581079)

[נספחים 14](#_Toc44581080)

# הכנת הנתונים, הערות ודגשים

* בחלקו הראשון של הפרויקט, הפכנו את ערכי משתנה המטרה, על מנת שיתאים למסקנות ששיערנו מהנתונים וכן לסט המקורי אותו ראינו באינטרנט. בחלק הזה, נעדיף להשאיר את משתנה המטרה כמו שהיה נתון בהתחלה על מנת לשמור על תאימות.
* כמו כן נשתמש באותו טיפול בנתונים שעשינו בחלק הקודם(טיפול בחריגים, יצירת משתנים חדשים וכו').
* את סט הנתונים נחלק לסט אימון(70%), וסט בחינה(30%). נשתמש בסט האימון בבניית המודל ומציאת ערכי ההיפר-פרמטרים הטובים ביותר, ונבדוק את המודל על סט הבחינה. בעזרת הפקודה set.seed(29), נוכל לוודא שאנחנו עובדים על אותו סט אימון, אימות ובחינה.

# Decision Trees

תמונה שמכילה שעון

התיאור נוצר באופן אוטומטיעץ החלטה הינו מודל לסיווג נוסף בו אנו נעזרים כאשר מטרתנו לסווג רשומות למחלקות ידועות (supervised problem) בצורת חלוקה רקורסיבית. מבחינת הנתונים העץ יודע להתמודד הן עם משתנים רציפים והן עם משתנים קטגוריאליים.

.1בעזרת החבילה rpart, נבנה עץ החלטה עבור סט האימון, ונחשב את אחוזי הדיוק עבור סט האימון וסט הבחינה ונקבל:

train Accuracy = 0.8551724

test Accuracy = 0.7704918

ניתן לראות שקיבלנו אחוזי דיוק טובים על סט האימון. לעומת זאת, יש הבדל משמעותי בינם לבין אחוזי הדיוק עבור סט הבחינה. אפשר להסביר את זה ע"י:overfitting, מצב שבו המודל לומד הרבה (התאמה מדוקדקת מידי של המודל על פי הנתונים שקיבל, וככל שיש יותר רעש, מצב של התאמת יתר יכול להתרחש) על סט האימון, ועל כן מצליח פחות בבצוע תחזיות עבור סט הבחינה.

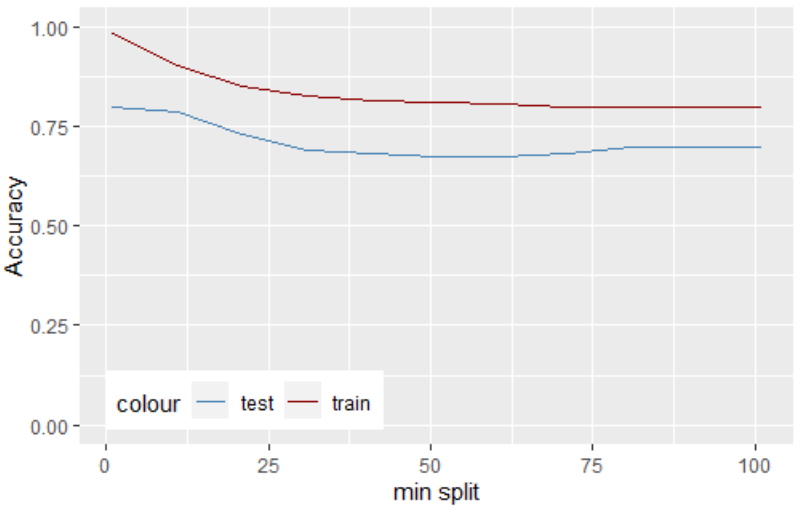
חשוב לציין שעבור סט אימון אחר (למשל, אם עשינו את חלוקת סט הנתונים לסט אימון וסט בחינה עבור set.seed(!=29), יש סיכוי לקבל עץ אחר, מכיוון שהעץ יהיה בנוי עבור נתונים אחרים. ולכן, לצורך שיפור הדיוקים , ומניעת מצב ההתאמת יתר, נעשה K-folds-CV, וגם שינוי בפרמטרים של המודל, וננסה לכוון אותם כדי לקבל את עץ ההחלטה האופטימלי.

## כיוון פרמטרים

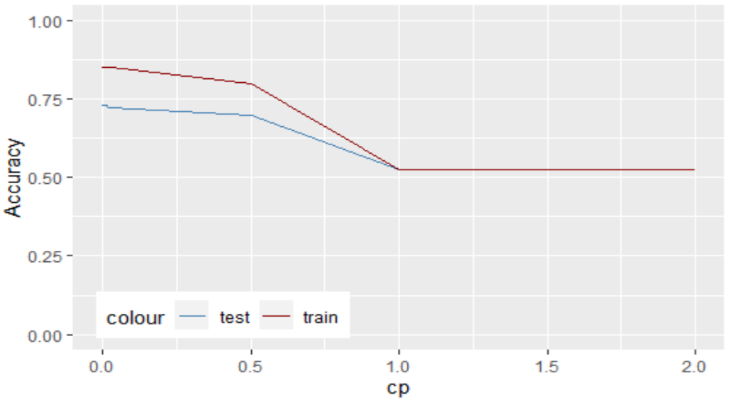
עכשיו, ננסה לכוון את הפרמטרים של המודל כדי לשפר את אחוזי הדיוק שקיבלנו בסעיף הקודם.

נחלק את סט הנתונים שלנו לעשרה, כל פעם בודקים ערך מסוים של ההיפר-פרמטר שרוצים לשפר, נבנה מודל בעזרת 90% מסט האימון ובודקים את הדיוק עבור סט האימון, וסט הבחינה (10% מסט האימון), נחזור על התהליך הזה 10 פעמים (K-folds=10), ובסוף נעשה את הממוצע של הדיוקים של סט הבחינה וסט האימון. נחזור על התהליך הזה עבור כל ערכי ההיפר-פרמטר שרוצים לבדוק.

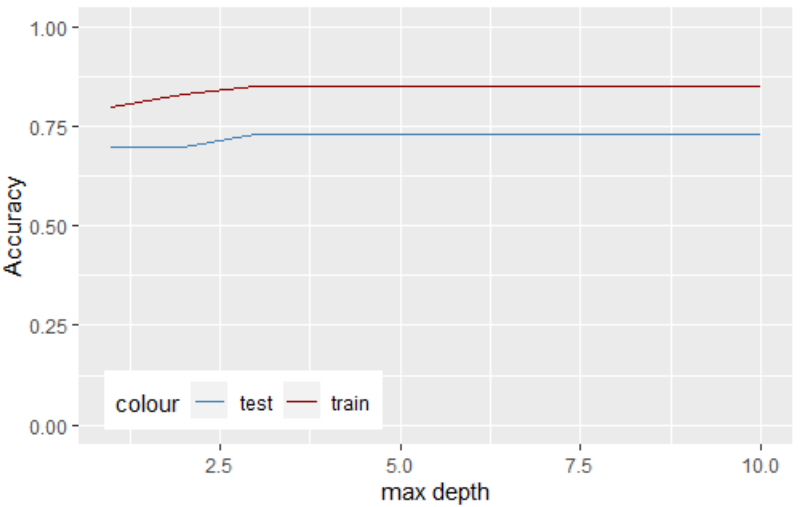
**minsplit**: המספר המינימלי של תצפיות שחייבות להתקיים בצומת כדי לפצל.

כאשר מגדילים פרמטר זה, העץ מוגבל יותר מכיוון שהוא צריך לקחת בחשבון דוגמאות נוספות בכל צומת.

ניתן לראות שככל ש-minsplit יותר גדול, הדיוק יקטן יותר (עבור אחוזי הדיוק של סט הבחינה, יקטן מ-0 עד-60, ואחר כך יעלה אבל בפחות מההתחלה). כלומר, עבור ה-minsplit, אנחנו נעדיף שישאף כמה שיותר ל-0.

**cp**: פרמטר מורכבות: פיצול שאינו מקטין את חוסר ההתאמה הכללי של גורם cp אינו מנסה. התפקיד העיקרי של הפרמטר הזה הוא לחסוך זמן מחשוב על ידי גיזום פיצולים שכמובן לא כדאי מהגרף, אנחנו נרצה שהפרמטר cp יהיה כמה שיותר נמוך ושואף ל-0.

**Maxdepth**: העומק המרבי של כל צומת של העץ הסופי, כאשר צומת השורש נספר כעומק 0.

הגדרת פרמטר זה תגרום לכך שהעץ "יפסיק לצמוח" כאשר העומק יהיה שווה לערך שנקבע בעומק המרבי.

אפשר לראות מהגרף, שעבור maxdepth < 4, אחוזי הדיוק ישתפרו ככל שהפרמטר יגדל, ואחרת, הדיוקים לא ישתפרו ויישארו כמו שהם. ולכן אפשר להגיד שהערך הטוב ביותר לפרמטר הזה הינו קרוב ל-4.

בדקנו לכל היפר-פרמטר בנפרד, מה הערכים הטובים ביותר שיכול לקבל עבור העץ האופטימלי. אבל, אנחנו צריכים לתאם בין שלושת ההיפר-פרמטרים ולבדוק איזו קומבינציה תשפר את עץ ההחלטה כמה שיותר (לדוגמה: בחרנו עומק = 3, אבל אולי עבור cp לא אפס נקבל דיוקים יותר טובים). בשביל לעשות זאת, ניעזר בפקודה tune.rpart, שמוציאה את הקומבינציה הטובה ביותר. נבדוק: minsplit=seq(1,41,2), cp=c(0,0.0005,0.001,0.002,0.005,0.01,0.02),maxdepth=1:6

ונקבל: minsplit = 9, cp = 0.02, maxdepth = 4.

תמונה שמכילה אובייקט, שעון

התיאור נוצר באופן אוטומטי**נאמן עכשיו את עץ ההחלטה עם הקונפיגורציה הטובה ביותר שמצאנו**, ונקבל:

tuning train Accuracy = 0.8827586

tuning test Accuracy = 0.7868852

אפשר לראות, שכיוון ההיפר-פרמטרים שיפר את הדיוקים שלנו עבור סט האימון וגם כן, עבור סט הבחינה.

נבדוק אחת התצפיות בסט הבחינה ידנית:

> Heart\_test[2,]

age gender cp trestbps chol fbs restecg thalach exang oldpeak slope ca thal

46 0 0 138 243 0 0 152 1 0 1 0 2

Y BloodPressureRange GoodMaxHeart

1 3 0

cp=0=0 -> thalach=152>148 -> thal=2=3 -> Y=1=1

ניתן לראות שהסיווג אכן תואם למציאות.

ממבנה עץ ההחלטה, ישנם 4 משתנים שעבורם ניתן לדעת אם המטופל חולה או לא, והם: סוג כאב בחזה, פעימות בדקה, תוצאת בדיקת לחץ Thalium, וגם המדרון בקטע השיא ב-ST.

תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטימהחלק הקודם, חשבנו שהמשתנה ca-מספר העורקים העיקריים החסומים, יהיה משתנה חשוב עבור סיווג המטופלים, בפרט עבור ca>1.

עבור הפלט של חשיבות המשתנים של עץ ההחלטה הסופי, נוכל לראות את החשיבות של כל משתנה על העץ. ישנם משתנים שחשבנו שיעזרו בדיוקים שלנו, אך כמעט לא היה להם חשיבות. למשל, ציפינו, וגם כן לפי הספרות שראינו, הגיל יכול לעזור בסיווג אם המטופל חולה או לא (ככל שעולה הגיל כך עולה הסיכוי לחלות במחלת לב). כמו כן המשתנה שיצרנו היה כמעט ללא חשיבות בכלל על עץ ההחלטה, טווח לחץ הדם (BloodPressureRange).

עבור המודלים הבאים, נעשה נרמול למשתנים הרציפים (ממוצע וסטיית תקן) עבור סט הנתונים, ניעזר בפונקציה class.ind() כדי לטפל במשתנים הקטגוריאליים. ואחר כך נחלק לסט בחינה וסט אימון.

# Neural Networks

כדי להריץ את המודל ההתחלתי, אנחנו צריכים להגדיר בנוסף ל-data, formula, את ההיפר-פרמטר size, שהוא מספר היחידות בשכבה המוסתרת. אנחנו בחרנו להתחיל בשכבה אחת, עם נוירון אחד חבוי, ואנחנו ננסה בסעיף הבא לכוון את הפרמטר הזה.

ברשת שלנו יש 35 נוירונים בשכבת הכניסה(כולל B1), כדי להריץ את רשת נוירונים, היינו צריכים להפוך את המשתנים הקטגוריאליים, ולכן קיבלנו יותר נוירונים בשכבת הכניסה. בנוסף לכך, יש לנו רק שכבה אחת חבויה ברשת, שיש בה רק נוירון אחד.(יותר משכבה אחת חבויה זה DNN).

עבור הרשת שהרצנו, נחשב את אחוזי הדיוק עבור סט הבחינה וסט האימון ונקבל:

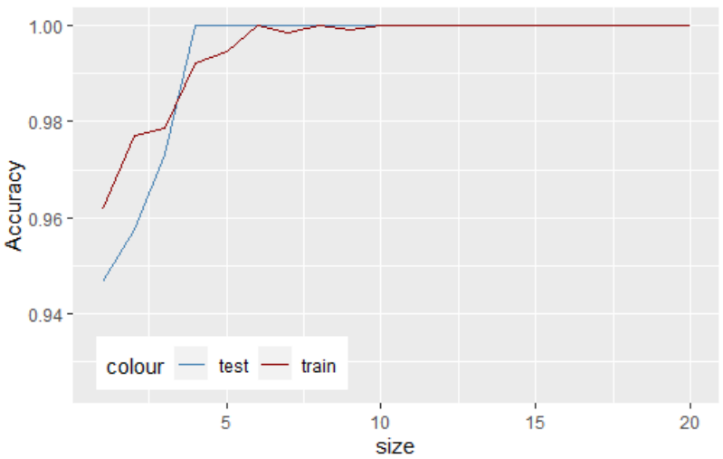
train accuracy: 0.937931

test accuracy: 0.7704918

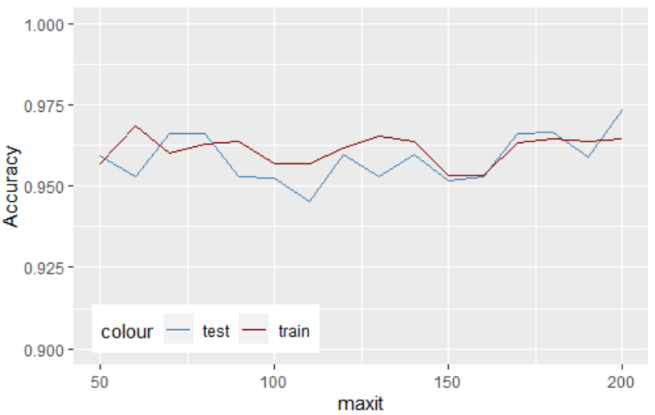
המודל נתן תוצאות דיוק גבוהות עבור סט האימון. לעומת זאת, אחוזי הדיוק עבור סט הבחינה נמוך ב- 16%. לכן, אנחנו ננסה לכוון את ההיפר-פרמטרים של המודל, כדי לשפר את הדיוקים, ולא להגיע למצב של התאמת יתר כמו שקיבלנו.

## כיוון פרמטרים

size: מספר היחידות בשכבה המוסתרת) מספר הנוירונים הנסתרים צריך להיות בין גודל שכבת הקלט לגודל שכבת הפלט.(

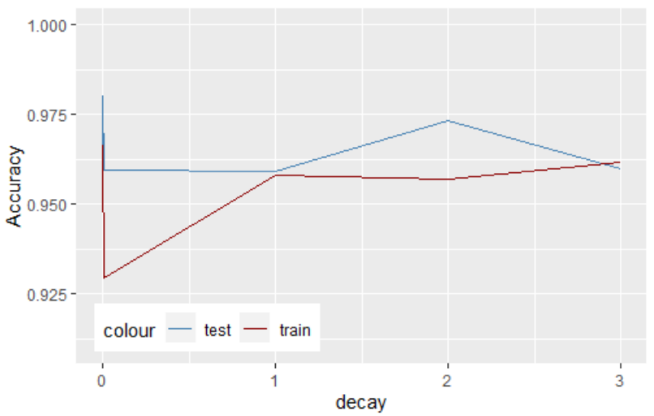
שימוש במעט מדי נוירונים בשכבות הנסתרות יביא לתת-התאמה. מצד שני, יותר מדי נוירונים בשכבות הנסתרות עלולים לגרום להתאמה יתר. גם כן, בעיה שנייה יכולה להתרחש גם כשנתוני האימונים מספיקים. מספר גדול מאוד של עצבנים בשכבות הנסתרות יכול להגדיל את הזמן שלוקח לאמן את הרשת. משך זמן האימונים יכול לגדול עד כי אי אפשר לאמן מספיק את הרשת העצבית. ברור שיש להגיע לפשרה כלשהי בין יותר מדי לבין מעט מדי נוירונים בשכבות הנסתרות.

ניתן לראות, שעבור 4 נוירונים נסתרים, הדיוק שלנו יהיה הטוב ביותר, וכאשר מוסיפים עוד נוירונים לשכבה, הדיוק עבור סט הבחינה לא ישתנה.

 המספר המרבי של האיטרציות. ברירת מחדל הוא 100 :**maxit**

אנחנו רוצים לבדוק איך מספר האיטרציות משפיע על הדיוקים של הרשת.

כאשר מספר האיטרציות גדול מ-~190, הדיוק חוזר ועולה יותר ממה שהיה עבור 75, 175. ולכן כאשר ננסה למצוא את הקונפיגורציה האופטימלית, ננסה טווח גדול של מספר האיטרציות, מכיוון שזה יכול להשתנות מטווח לטווח.

**Decay**: פרמטר לירידה במשקל.

פרמטר אשר מונע התאמת יתר. עדיף לבחור ערכים שקטנים מ-0.5 לבדיקה, אבל אנחנו גם רצינו לראות מה קורה אם נקבע את הפרמטר להיות יותר מחצי.

מהגרף, עבור הערכים השואפים ל-0, נקבל את אחוזי הדיוק הגבוהים ביותר.

כעת, אחרי שבדקנו עבור כל שינוי היפר-פרמטר בנפרד כיצד הוא משנה את דיוקי רשת הנוירונים, ננסה למצוא קונפיגורציה אופטימאלית עבור שלושת ההיפר-פרמטרים שהצגנו. נמצא אותה באותה דרך שבדקנו איך השינוי בהיפר-פרמטר מסוים משפיע על הדיוק, אבל הפעם נעבור על כל הקומבינציות עבור שלושת ההיפר-פרמטרים, ונבחר את הקומבינציה שתיתן את הדיוק הגבוה ביותר עבור הבחינה.

נבדוק:

**size=c(21,27), maxit=seq(100,300,20), decay=c(0,0.00001,0.0001,0.001,0.01)**

הקונפיגורציה שבחרנו היא: size=24, maxit=100, decay=0

הרשת האופטימלית שהגענו אליה מורכבת מ-35 נוירונים בשכבת הכניסה, והם המשתנים שלנו כולל המשתנים הקטגוריאליים, ו-B1. גם ברשת זו יש שכבה אחת נסתרת, אבל עכשיו יש בה 24 נוירונים חבויים. ככל שיש יותר נוירונים, המודל יילמד יותר טוב, אך יש להיזהר לא ליפול להתאמת-יתר (בדקנו עד 27 נוירונים חבויים בשכבה החבויה, אבל עבור 24, וההיפר-פרמטרים האחרים קיבלנו את הדיוק הכי טוב, והפשוט ביותר).

נחשב את אחוזי הדיוק עבור סט האימון וסט הבחינה ונקבל:

tune train accuracy: 1

tune test accuracy: 0.7868852

אחוז הדיוק עבור סט האימון עלה ל- 100%. שזה דבר חיובי. אבל, כיוון הפרמטרים כמעט ולא שינה את אחוזי הדיוק עבור סט הבחינה, שזה הדבר החשוב ביותר.

# K-Means

עכשיו נעבור למודל שונה מהמודלים הקודמים: האלגוריתם של K-Means הינו אלגוריתם שעובד בצורה לא מונחית (unsupervised), כך שהוא לא מקבל את משתנה המטרה, אלא מנסה לחלק את הנתונים שהוא קיבל ל-K clusters (כך שאנחנו חייבים "להגיד לו" כקלט כמה clusters יש).

האלגוריתם מחשב את המרחק האיקלידי בין נקודות (תצפיות) ולכן גם פה ננרמל את המשתנים הרציפים, וגם המשתנים הקטגוריאליים נהפוך אותם למשתני דמה.

מכיוון שמדובר באלגוריתם לא מונחה, שלא מקבל את משתנה המטרה, אנחנו לא נחלק את הסט הנתון לסט אימון וסט בחינה.

**נריץ עכשיו את המודל עם ערכי ברירת המחדל**, וה-K שמתאים לבעיה שלנו, שהוא- 2 (חולים לעומת לא חולים). נזכיר שוב שאנחנו נותנים למודל את כל המשתנים שלנו, אך בלי משתנה המטרה(Y). במודל הזה אנחנו לא נעשה cross-validation, מכיוון שמדובר במודל לא מונחה.

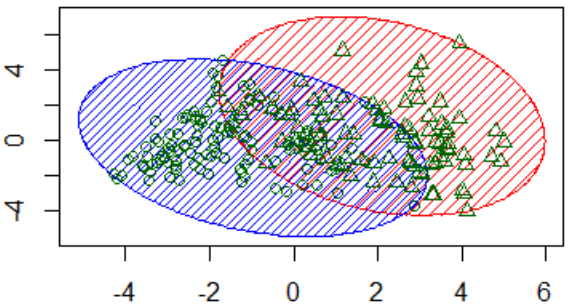
**נבדוק עכשיו בתוצאות המודל:**

נסתכל איך המודל סיווג את התצפיות, ונחשב את אחוז הדיוק עבור ה-Y שיש לנו:

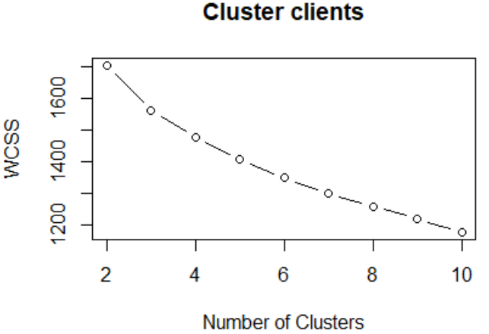
תמונה שמכילה מד

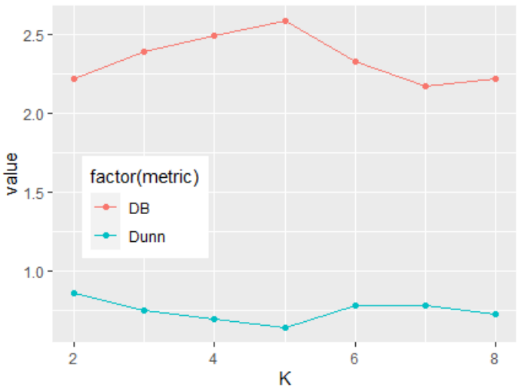
התיאור נוצר באופן אוטומטיאנכי: תוצאות המודל. אופקי: ה-Y

האלגוריתם מספק לנו תוצאות די נמוכות בהשוואה למודלים שהצגנו קודם. סיבה לכך יכולה להיות שהאלגוריתם מתחיל מנקודה אקראית ומתחיל לחלק ל-K קבוצות וייתכן שיש K אופטימלי יותר שיניב את אחוזי דיוק יותר טובים ממה שקיבלנו.

****בתרשים מוצג איך האלגוריתם חילק את התצפיות לשתי קבוצות:

**בלי להתחשב בכך שאנו יודעים את מספר ה-clusters, ננסה לבחור את ערך ה-K המתאים ביותר:**

****יש כמה מדדים לבדיקת ה-K האופטימלי. למשל, **WCSS**, (משרטטים את הקשר בין מספר האשכולות לתוך סכום הריבועים (WCSS) ואז אנו בוחרים את מספר האשכולות בהם השינוי ב- WCSS מתחיל להתיישר (שיטת המרפק)). לפי הגרף, התלבטנו בין לבחור 4 או-5.

**** ולכן אנחנו ניעזר בשיטה נוספת, שבה אנחנו ממזערים את ההפרש בין המדדים DB ל-DUNN.

**Dunn:** מבטא את המרחק בין המחלקות, המדד הזה נרצה למקסם אותו. המדד השני הוא מדד DB: מבטא את המרחק בתוך כל אשכול, המטרה הינה למזער את המדד הזה.

ניתן לראות מהגרף שהמרחק בין שני המדדים יהיה מקסימלי עבור 5=K.

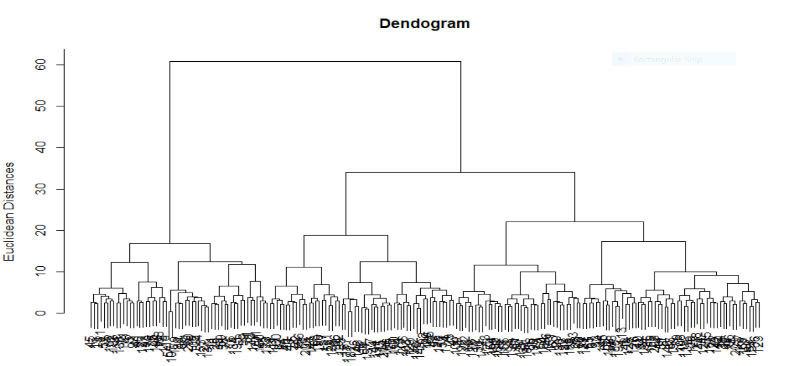
ולכן, נוכל לומר שה-K המיטבי הוא 5, אך אנו יודעים מהסיווג האמיתי שאין זה מתקשר לסיפור המקרה.

## שיטת אשכול נוספת

בסעיף זה נציג שיטת אשכול שנקראת: **Hierarchical clustering**. בשיטה זו, האלגוריתם מתחיל כך שכל sample הינה קבוצה (אשכול). בשיטה של מרחק איקלידי, הוא מנסה למצוא את שתי הקבוצות הקרובות ביותר, ומאחד אותם לקבוצה אחת. האלגוריתם ממשיך לאחד את הקבוצות, עד שמגיעים לקבוצה אחת שנכלל בתוכה את כל sampleים.

נציג את התהליך שתיארנו בעזרת dendrogram, שהוא ייצוג חזותי של האשכולות, המציג את ההיררכיה בצורה מאוד ברורה:

ככל שששתי הקבוצות שרוצים לאחד יותר קרובות, המרחק ביניהם יותר קטן (תחילת הדינדרוגרם), וככל שיש לנו פחות קבוצות, המרחק האיקלידי ביניהם לרוב יהיה יותר גדול.

נבחר את מספר הקבוצות על ידי הסתכלות על המרחק האיקלידי שמקשר בין שתי קבוצות (נעשה קו אופקי). לפי הגרף, ניתן לראות שהמרחק מתחיל לגדול בצורה משמעותית אחרי מרחק 20, ולכן נעדיף לבחור 4=K או 5=K.

ההבדל בין שיטת האשכול הזו לבין K-Means, היא שבשיטה הזו אנחנו יכולים לראות בצורה יותר טובה איך הקבוצות מתאחדות.

היתרונות של המודל הזה לעומד ה-K-Means, הינה שהדינדרוגראם מציג איך הוא מאחד את הקבוצות, מה שעוזר לנו מאות בהבנת הנתונים שיש לנו, וכן, אין צורך להגדיר את מספר הקבוצות (K), אלא רק לייצר את הדינדרוגראם ולהסיק מסקנות ממנו.

ל-Hierarchical clustering יש גם חסרונות (לעומת-K-Means). למשל, אם נדמיין גרף שבו מוצג דינדוגראם שיש בו 10000 sampleים, הגרף לא יהיה מובן ויהיה קשה לבחור ממנו את מספר הקבוצות האופטימלי. בנוסף לכך, ל-Hierarchical clustering זמן ריצה של , ול-K-means זמן ריצה של .

השוואה בין מודלים

מהתוצאות שקיבלנו, ל-NN וגם ל-DT היה את אותו אחוז דיוק עבור סט הבחינה במודל הסופי. לעומת זאת, אנו נעדיף את רשת הנוירונים על עץ ההחלטה, מכיוון שלעץ ההחלטה סיכוי יותר גדול לטעויות חישוב. למשל, תצפיות חריגות במשתנה רציף יכולות להשפיע על המדדים ולבנות עץ לא הכי מדויק.

כמו כן נעדיף את רשת הנוירונים על K-means, מכיוון שאלגוריתם זה בוחר נקודות מרכז אקראיות - דבר שיכול להשפיע על החלוקה. בנוסף, מצאנו ש-K האופטימלי הוא 3, כלומר עכשיו יותר קשה להשוות בין המודלים. אם היינו משתמשים ב- K-means++, כנראה שהתוצאות המתקבלות היו יותר מדוייקות.

בנוסף לכך, אם יש לנו את משתנה המטרה, מדוע לא להשתמש בו? המודלים המונחים יכולים להיות עדיפים יותר ומדוייקים יותר מכיוון שיש לנו את משתנה המטרה.

|  |  |  |  |
| --- | --- | --- | --- |
|  | K-Means | DT | NN |
| ביצועים | 23.3%  הביצועים שלו תלויים מאוד בבחירת מרכזי האשכול הראשוניים. בהיותו אלגוריתם חמדן, הוא מתכנס לעיתים למינימום מקומיים ועשוי לייצר אשכולות ריקים. | 77.05% | 78.7% |
| מורכבות | מורכבות ריצה של O() מורכבות זמן גבוהה זו הופכת אמצעי k לבלתי ראויים ליישומים גדולים. | סיבוכיות זמן הריצה גדלה באופן מעריכי עם מספר המאפיינים. | מודל מורכב בעל סיבוכיות זמן ריצה גבוהה ומקום נרחב בזיכרון ביחס למודלים אחרים. |
| יתרונות | 1.פשוט יחסית ליישום.  2.משתנה למערכות נתונים גדולות.  3.מבטיח התכנסות.  4.מסתגל בקלות לדוגמאות חדשות.  5.הכללה לאשכולות בצורות ובגדלים שונים, כמו אשכולות אליפטיים. | 1.בהשוואה לאלגוריתמים אחרים עצי ההחלטה דורשים פחות מאמץ להכנת נתונים במהלך העיבוד המקדים.  2.עץ החלטה אינו דורש נורמליזציה של נתונים.  3.עץ החלטה אינו דורש גם קנה מידה של נתונים.  4.ערכים חסרים בנתונים גם אינם משפיעים על תהליך בניית עץ ההחלטות במידה ניכרת.  5.מודל עצי החלטה הוא מאוד אינטואיטיבי וקל להסביר | 1.אחסון מידע על כל הרשת - היעלמותם של כמה פיסות מידע במקום אחד אינה מגבילה את תפקוד הרשת.  2.לאחר אימון, הנתונים עשויים להפיק פלט אפילו עם מידע לא שלם.  3.השחתה של תאים אחד או יותר של הרשת אינה מונעת ממנו לייצר תפוקה.  4.בעל זיכרון מבוזר  5.לומד מאירועים ומקבל החלטות על ידי התייחסות לאירועים דומים.  6.יכולת עיבוד מקבילה |
| חסרונות | 1.בחירה ידנית.  2.תלות בערכים ראשוניים.  3.מתקשה בנתוני אשכולות שבהם האשכולות הם בגדלים וצפיפות משתנים.  4.ככל שמספר ממדים גדל, מידת דמיון מבוססת מרחק מתכנסת לערך קבוע בין כל דוגמאות נתונות. | 1.שינוי קטן בנתונים יכול לגרום לשינוי גדול במבנה עץ ההחלטה ולגרום לחוסר יציבות.  2.לעץ החלטה לפעמים החישוב יכול להיות מורכב בהרבה בהשוואה לאלגוריתמים אחרים.  3.עץ ההחלטה כרוך לרוב בזמן גבוה יותר לאימון הדגם.  4.אימון עץ ההחלטה הוא יקר יחסית מכיוון שמורכבות והזמן שנמשך הוא רב יותר. | 1.תלות בחומרה - דורשים מעבדים בעלי עיבוד מקביל, לפי המבנה שלהם.  2.תפקוד לא מוסבר של הרשת  3.אין כלל ספציפי לקביעת מבנה הרשתות. מבנה הרשת המתאים מושג באמצעות ניסיון וניסוי וטעיה.  4.הקושי להציג את הבעיה לרשת: ANNs יכולים לעבוד עם מידע מספרי. יש לתרגם בעיות לערכים מספריים לפני שהוצגו ל- ANN.  5.משך הרשת אינו ידוע |

# המודל הנבחר

כמו שהזכרנו קודם, נעדיף את רשת הנוירונים על המודלים האחרים.

זאת מכיוון שמודל זה מספק אחוזי דיוק גבוהים יותר משאל המודלים עבור סט הבחינה. אמנם ההפרש בדיוק מול מודל עץ ההחלטה הוא 1.5%~ אך מודל מודל האישכול הפער משמעותי מאוד (דיוק גבוה מעל פי 3 ממודל K-Means). על כן נבחר במודל זה - על אף חסרון סיבוכיות הריצה והמורכבות שלו. זאת מכיוון שגם שינוי קטן בנתונים של מודל עץ החלטה יכול להביא בשינוי משמעותי במבנה שלו ולהריע את הדיוק עוד יותר (חוסר יציבות) ועל כן קיימת סבירות שהפרש באחוז הדיוק יגדל. מודל האישכול K-Means סיפק דיוק נמוך פי 3 ממודל רשת הנוירונים ועל אף שיתרונותיו הם בפשטותו וקלות היישום שלו אין הצדקה לפער כה גדול בדיוק בהשוואה בין מודלים אלו.

נבדוק גם שלושה מודלים נוספים שהם: svm, random forest, gbm.

## Random Forest

מודל נוסף שאנחנו רוצים להציג הוא RF, שבנוי מעצים רבים (ברירת המחדל היא 500).

נציג ישירות תהליך של CV עבור 10=K. וגם ננסה לכוון את ההיפר-פרמטרים הבאים:

**ntree:** מספר העצים שהמודל בנוי מהם. צריכים לכוון את הפרמטר הזה, כך שעבור ערכים גדולים יכול להיות שתתקיים התאמת יתר.

**mtry:** מספר המשתנים שנדגמו באופן אקראי כמועמדים בכל פיצול.

נבדוק גם את ערכי ברירת המחדל(ntree=500,mtry=3)

תמונה שמכילה טקסט, מפה

התיאור נוצר באופן אוטומטינציג את התוצאות בגרף לעיל:

Red=test validation

2:6 מציגים את mtry

תמונה שמכילה מפה, טקסט

התיאור נוצר באופן אוטומטי

אותו גרף אבל בלי ה-train:

**mtry=2,ntree=550:**

test Accuracy = 0.8688525

נקבל את אחוזי הדיוק הגבוהים ביותר בעזרת המודל של Random Forest.

## Gradient Boosting

המודל הזה הינו אחד המודלים המפורסמים ביותר, אשר ניצח פעמים רבות בתחרויות ב-Kaggle.

חלק מיתרונותיו: GBM יכול לבצע אופטימיזציה לפונקציות אובדן שונות ומספק כמה אפשרויות לכוונון היפר-פרמטרים. כמו כן, בדרך כלל הוא עובד טוב מאוד עם נתונים שאינם מעובדים מראש (למשל, משתנים קטגוריים ומספריים יחד) ונתונים חסרים.

גם במודל הזה נשתמש ב-CV, 10=K. אבל כעת נעשה tuning בעזרת החבילה caret:

interaction.depth = c(3, 5, 7, 9)

n.trees = (1:10)\*10

shrinkage = 0.1

) n.minobsinnode = c(10,15,20,25

נציג את התוצאות שקיבלנו:

תמונה שמכילה טקסט, מפה

התיאור נוצר באופן אוטומטי

**n.trees=50, n.minobsinnode=20, tree depth=5**, הקומבינציה האופטימלית עבור מודל GBM.

לעומת זאת שהמודל הינו מודל חזק מאוד, קיבלנו אחוזי דיוק יותר טובים ב-RF.

# נספחים

#######################################################

########## Machine Learning Project - Part 2 ##########

#######################################################

library(ggplot2)

library(dplyr)

## Please put all the datasets in the same folder(Set As Working Directory) ##

#Uploading the dataset:

getSet <- function(Heart){

##################################

##### Working with NA Values #####

##################################

#make missing values as NA:

Heart$age <- ifelse(Heart$age > 100,NA,Heart$age) #age column:

Heart$ca <- ifelse(Heart$ca == 4,NA,Heart$ca) #ca column:

Heart$thal <- ifelse(Heart$thal == 0,NA,Heart$thal) #thal column:

#Duplicated rows:

which(duplicated(Heart) | duplicated(Heart[nrow(Heart):1, ])[nrow(Heart):1])

#Missing data in specific row:

row.missing <- function(Column){

MissingValues <- c()

for(i in 1:length(Column)){

if(is.na(Column[i])){

MissingValues <- c(MissingValues,i)

}

}

print(MissingValues)

}

Heart$gender = as.factor(Heart$gender)

Heart$fbs = as.factor(Heart$fbs)

Heart$exang = as.factor(Heart$exang)

Heart$cp = as.factor(Heart$cp)

Heart$slope = as.factor(Heart$slope)

Heart$ca = as.factor(Heart$ca)

Heart$thal = as.factor(Heart$thal)

Heart$restecg = as.factor(Heart$restecg)

Heart <- Heart %>% rename(Y = y) #change target name(To make it easy:

#y is y axis,Y is the Target variable)

Heart$Y <- as.factor(Heart$Y) #change Y to factor

#####################################

##### Working with missing data #####

#####################################

#Dropping id column:

Heart <- select(Heart,-id)

#Dropping rows with NA columns:

if(length(Heart) > 13){

Heart <- Heart %>% filter(!is.na(ca))

Heart <- Heart %>% filter(!is.na(thal))

}

# Fill NA values in age column:

ageMed <- as.data.frame(Heart %>% group\_by(ca) %>% summarise(MED = median(age,na.rm = T)))

for(i in 1:length(Heart$age)){

if(is.na(Heart[i,1])){#just for missing values

Heart[i,1] <- ageMed[as.integer(Heart[i,12])+1,2] #change missing value by age-ca median.

}

}

##########################

##### Discretization #####

##########################

# BloodPressureRange

BloodPressure <- function(tres){

if(tres<120){

tres <- 1

}

else if(tres<129){

tres <- 2

}

else if(tres<139){

tres <- 3

}

else{

tres <- 4

}

}

Heart$BloodPressureRange <- as.factor(sapply(Heart$trestbps,BloodPressure))

###############################

##### Feature engineering #####

###############################

# GoodMaxHeart

Heart$new <- rep(220,length(Heart$age)) - Heart$age

Heart$GoodMaxHeart <- as.factor(ifelse((Heart$new\*0.76 > Heart$thalach) &

(Heart$new\*0.64 < Heart$thalach),1,0))

Heart <- select(Heart,-new)

return(Heart)

}

Heart <- getSet(read.csv('Xy\_train.csv'))

#############################################################

# Heart\_test\_final <- getSet(read.csv('X\_test.csv'))

###########################################################################

###########################################################################

## Split the Dataset to training & test sets:

library(caTools)

set.seed(29)

split <- sample.split(Heart$Y ,SplitRatio = 0.7)

Heart\_train <- subset(Heart,split == T)

Heart\_test <- subset(Heart,split == F)

#########################

##### Decision Tree #####

#########################

library(rpart)

library(rpart.plot)

### 1

tree <- rpart(Y ~ . ,method = 'class',data = Heart\_train)

#prp(tree)

rpart.plot(tree)

#Predict on training set:

preds.tree.train <- predict(tree, newdata = select(Heart\_train,-Y), type = 'class')

(CM\_train <- table(preds.tree.train,Heart\_train$Y))

cat('train Accuracy =',sum(diag(CM\_train)) / sum(CM\_train))

#Predict on test set:

preds.tree.test <- predict(tree, newdata = select(Heart\_test,-Y), type = 'class')

(CM\_test <- table(preds.tree.test,Heart\_test$Y))

cat('test Accuracy =',sum(diag(CM\_test)) / sum(CM\_test))

### 2

library(caret)

set.seed(29)

folds <- createFolds(Heart\_train$Y, k = 10)

final\_pred\_test <- c()

final\_pred\_train <- c()

Minsplit <- seq(1,101,10)

for(i in Minsplit){

test\_pred <- c()

train\_pred <- c()

for(x in folds){

training\_fold <- Heart\_train[-x, ]

test\_fold <- Heart\_train[x, ]

classifier <- rpart(formula = Y ~ .,

data = training\_fold,

method = 'class', minsplit = i)

y\_pred\_test <- predict(classifier, newdata = test\_fold[,-14], type = 'class')

cm\_test <- table(test\_fold[,14], y\_pred\_test)

accuracy\_test <- sum(diag(cm\_test)) / sum(cm\_test)

y\_pred\_train <- predict(classifier, newdata = training\_fold[,-14], type = 'class')

cm\_train <- table(training\_fold[,14], y\_pred\_train)

accuracy\_train <- sum(diag(cm\_train)) / sum(cm\_train)

test\_pred <- c(test\_pred,accuracy\_test)

train\_pred <- c(train\_pred,accuracy\_train)

}

tst <- mean(test\_pred)

trn <- mean(train\_pred)

final\_pred\_train <- c(final\_pred\_train,trn)

final\_pred\_test <- c(final\_pred\_test,tst)

}

(df\_minsplit <- data.frame(Minsplit,final\_pred\_train,final\_pred\_test))

ggplot(df\_minsplit, aes(Minsplit)) +

geom\_line(aes(y=final\_pred\_test, color = "test")) +

geom\_line(aes(y=final\_pred\_train, color = "train")) +

scale\_color\_manual(values = c( "steelblue","darkred"))+

scale\_y\_continuous(name="Accuracy", limits=c(0, 1)) +

xlab('min split') + theme(legend.position = c(0.23, 0.1),

legend.direction = "horizontal")

###

final\_pred\_test <- c()

final\_pred\_train <- c()

cp <- c(0,0.002,0.005,0.01,0.015,0.02,0.03,0.5,1,2)

for(i in cp){

test\_pred <- c()

train\_pred <- c()

for(x in folds){

training\_fold <- Heart\_train[-x, ]

test\_fold <- Heart\_train[x, ]

classifier <- rpart(formula = Y ~ .,

data = training\_fold,

method = 'class', cp = i)

y\_pred\_test <- predict(classifier, newdata = test\_fold[,-14], type = 'class')

cm\_test <- table(test\_fold[,14], y\_pred\_test)

accuracy\_test <- sum(diag(cm\_test)) / sum(cm\_test)

y\_pred\_train <- predict(classifier, newdata = training\_fold[,-14], type = 'class')

cm\_train <- table(training\_fold[,14], y\_pred\_train)

accuracy\_train <- sum(diag(cm\_train)) / sum(cm\_train)

test\_pred <- c(test\_pred,accuracy\_test)

train\_pred <- c(train\_pred,accuracy\_train)

}

tst <- mean(test\_pred)

trn <- mean(train\_pred)

final\_pred\_train <- c(final\_pred\_train,trn)

final\_pred\_test <- c(final\_pred\_test,tst)

}

(df\_cp <- data.frame(cp,final\_pred\_train,final\_pred\_test))

ggplot(df\_cp, aes(cp)) +

geom\_line(aes(y=final\_pred\_test, color = "test")) +

geom\_line(aes(y=final\_pred\_train, color = "train")) +

scale\_color\_manual(values = c( "steelblue","darkred"))+

scale\_y\_continuous(name="Accuracy", limits=c(0, 1)) +

xlab('cp') + theme(legend.position = c(0.23, 0.1),

legend.direction = "horizontal")

###

final\_pred\_test <- c()

final\_pred\_train <- c()

maxdepth <- c(1:10)

for(i in maxdepth){

test\_pred <- c()

train\_pred <- c()

for(x in folds){

training\_fold <- Heart\_train[-x, ]

test\_fold <- Heart\_train[x, ]

classifier <- rpart(formula = Y ~ .,

data = training\_fold,

method = 'class', maxdepth = i)

y\_pred\_test <- predict(classifier, newdata = test\_fold[,-14], type = 'class')

cm\_test <- table(test\_fold[,14], y\_pred\_test)

accuracy\_test <- sum(diag(cm\_test)) / sum(cm\_test)

y\_pred\_train <- predict(classifier, newdata = training\_fold[,-14], type = 'class')

cm\_train <- table(training\_fold[,14], y\_pred\_train)

accuracy\_train <- sum(diag(cm\_train)) / sum(cm\_train)

test\_pred <- c(test\_pred,accuracy\_test)

train\_pred <- c(train\_pred,accuracy\_train)

}

tst <- mean(test\_pred)

trn <- mean(train\_pred)

final\_pred\_train <- c(final\_pred\_train,trn)

final\_pred\_test <- c(final\_pred\_test,tst)

}

(df\_maxdepth <- data.frame(maxdepth,final\_pred\_train,final\_pred\_test))

ggplot(df\_maxdepth,aes(x=maxdepth)) +

geom\_line(aes(y = final\_pred\_train), color = "darkred") +

geom\_line(aes(y = final\_pred\_test), color="steelblue", linetype="twodash")+

ggtitle('blue: validation, brown: train')+

ylab('accuracy') + xlab('max depth') #+

#geom\_vline(xintercept = max(df\_maxdepth$final\_pred\_test), color = "yellow")

ggplot(df\_maxdepth, aes(maxdepth)) +

geom\_line(aes(y=final\_pred\_test, color = "test")) +

geom\_line(aes(y=final\_pred\_train, color = "train")) +

scale\_color\_manual(values = c( "steelblue","darkred"))+

scale\_y\_continuous(name="Accuracy", limits=c(0, 1)) +

xlab('max depth') + theme(legend.position = c(0.23, 0.1),

legend.direction = "horizontal")

####

library(e1071)

audit.rpart <- tune.rpart(Y ~ ., data=Heart\_train, minsplit=seq(1,16,2),

cp = c(0,0.0005,0.001,0.002,0.005,0.01,0.02,0.03),

maxdepth = 1:6)

audit.rpart

### 3

##

final\_classifier <- rpart(formula = Y ~ .,data = Heart\_train,method = 'class',

minsplit = 9, cp = 0.02, maxdepth = 4)

rpart.plot(final\_classifier)

y\_pred <- predict(final\_classifier, newdata = Heart\_test[,-14], type = 'class')

cm <- table(Heart\_test[,14], y\_pred)

(accuracy <- (sum(diag(cm)) / sum(cm)))

cat('tuning test Accuracy =',sum(diag(cm)) / sum(cm))

y\_pred <- predict(final\_classifier, newdata = Heart\_train[,-14], type = 'class')

cm <- table(Heart\_train[,14], y\_pred)

(accuracy <- (sum(diag(cm)) / sum(cm)))

cat('tuning train Accuracy =',sum(diag(cm)) / sum(cm))

##

rpart.plot(final\_classifier)

##

var.important <- final\_classifier$variable.importance

df\_var\_imp <- data.frame('var.name' = names(var.important),var.important)

ggplot(df\_var\_imp,aes(y=reorder(as.factor(var.name),var.important),x=var.important)) +

geom\_bar(stat = "identity",aes(fill = var.important)) + xlab('variable') +

ylab('important') +theme\_classic()

###############################################################

###############################################################

###############################################################

###Normalize the data

normalize <- function(x) {

return ((x - mean(x)) / (sd(x)))

}

HearT <- Heart

HearT$age <- normalize(HearT$age)

HearT$trestbps <- normalize(HearT$trestbps)

HearT$chol <- normalize(HearT$chol)

HearT$thalach <- normalize(HearT$thalach)

HearT$oldpeak <- normalize(HearT$oldpeak)

#####################

## Factors########

################

library(nnet)

HearT$gender <- class.ind(HearT$gender)

HearT$cp <- class.ind(HearT$cp)

HearT$fbs <- class.ind(HearT$fbs)

HearT$restecg <- class.ind(HearT$restecg)

HearT$exang <- class.ind(HearT$exang)

HearT$slope <- class.ind(HearT$slope)

HearT$ca <- class.ind(HearT$ca)

HearT$thal <- class.ind(HearT$thal)

HearT$BloodPressureRange <- class.ind(HearT$BloodPressureRange)

HearT$GoodMaxHeart <- class.ind(HearT$GoodMaxHeart)

#Splitting:

library(caTools)

set.seed(29)

split <- sample.split(HearT$Y ,SplitRatio = 0.7)

Heart\_train <- subset(HearT,split == T)

Heart\_test <- subset(HearT,split == F)

###############################################################

###############################################################

###############################################################

#######################

##### Neural Nets #####

#######################

library(nnet)

library(NeuralNetTools)

library(dplyr)

##

nn <- nnet(Y~.,data = Heart\_train, size = 1)

plotnet(nn)

#plot(nn)

# Train confusion matrix & accuracy

pred\_train <- predict(nn, type='class')

(cm\_train <- table(prediction=pred\_train, truth=Heart\_train$Y))

(accuracy\_train <- (sum(diag(cm\_train)) / sum(cm\_train)))

cat('train accuracy:',accuracy\_train)

# Test confusion matrix & accuracy

pred\_test <- predict(nn, newdata = select(Heart\_test,-Y), type='class')

(cm\_test <- table(prediction=pred\_test, truth=Heart\_test$Y))

(accuracy\_test <- (sum(diag(cm\_test)) / sum(cm\_test)))

cat('test accuracy:',accuracy\_test)

#########

### 2 ###

#########

library(caret)

set.seed(29)

folds <- createFolds(Heart\_train$Y, k = 10)

# size

final\_pred\_test <- c()

final\_pred\_train <- c()

Size <- c(1:27)

for(i in Size){

test\_pred <- c()

train\_pred <- c()

for(x in folds){

training\_fold <- Heart\_train[-x, ]

test\_fold <- Heart\_train[x, ]

classifier <- nnet(Y~.,data = Heart\_train, size = i)

y\_pred\_test <- predict(classifier, newdata = test\_fold[,-14], type = 'class')

cm\_test <- table(test\_fold[,14], y\_pred\_test)

accuracy\_test <- sum(diag(cm\_test)) / sum(cm\_test)

y\_pred\_train <- predict(classifier, newdata = training\_fold[,-14], type = 'class')

cm\_train <- table(training\_fold[,14], y\_pred\_train)

accuracy\_train <- sum(diag(cm\_train)) / sum(cm\_train)

test\_pred <- c(test\_pred,accuracy\_test)

train\_pred <- c(train\_pred,accuracy\_train)

}

tst <- mean(test\_pred)

trn <- mean(train\_pred)

final\_pred\_train <- c(final\_pred\_train,trn)

final\_pred\_test <- c(final\_pred\_test,tst)

}

(df\_nn\_size <- data.frame(Size,final\_pred\_train,final\_pred\_test))

ggplot(df\_nn\_size, aes(Size)) +

geom\_line(aes(y=final\_pred\_test, color = "test")) +

geom\_line(aes(y=final\_pred\_train, color = "train")) +

scale\_color\_manual(values = c( "steelblue","darkred"))+

scale\_y\_continuous(name="Accuracy", limits=c(0.925, 1)) +

xlab('size') + theme(legend.position = c(0.23, 0.1),

legend.direction = "horizontal")

#maxit

final\_pred\_test <- c()

final\_pred\_train <- c()

Maxit <- seq(50,500,10)

for(i in Maxit){

test\_pred <- c()

train\_pred <- c()

for(x in folds){

training\_fold <- Heart\_train[-x, ]

test\_fold <- Heart\_train[x, ]

classifier <- nnet(Y~.,data = Heart\_train, Maxit = i,size = 1)

y\_pred\_test <- predict(classifier, newdata = test\_fold[,-14], type = 'class')

cm\_test <- table(test\_fold[,14], y\_pred\_test)

accuracy\_test <- sum(diag(cm\_test)) / sum(cm\_test)

y\_pred\_train <- predict(classifier, newdata = training\_fold[,-14], type = 'class')

cm\_train <- table(training\_fold[,14], y\_pred\_train)

accuracy\_train <- sum(diag(cm\_train)) / sum(cm\_train)

test\_pred <- c(test\_pred,accuracy\_test)

train\_pred <- c(train\_pred,accuracy\_train)

}

tst <- mean(test\_pred)

trn <- mean(train\_pred)

final\_pred\_train <- c(final\_pred\_train,trn)

final\_pred\_test <- c(final\_pred\_test,tst)

}

(df\_nn\_maxit <- data.frame(Maxit,final\_pred\_train,final\_pred\_test))

ggplot(df\_nn\_maxit, aes(Maxit)) +

geom\_line(aes(y=final\_pred\_test, color = "test")) +

geom\_line(aes(y=final\_pred\_train, color = "train")) +

scale\_color\_manual(values = c( "steelblue","darkred"))+

scale\_y\_continuous(name="Accuracy", limits=c(0.9, 1)) +

xlab('maxit') + theme(legend.position = c(0.23, 0.1),

legend.direction = "horizontal")

#dacay

final\_pred\_test <- c()

final\_pred\_train <- c()

Decay <- c(0,0.000001,0.00001,0.0001,0.001,0.01,0.2,0.3,0.4,0.5,1,2,3)

for(i in Decay){

test\_pred <- c()

train\_pred <- c()

for(x in folds){

training\_fold <- Heart\_train[-x, ]

test\_fold <- Heart\_train[x, ]

classifier <- nnet(Y~.,data = Heart\_train, Decay = i,size = 1)

y\_pred\_test <- predict(classifier, newdata = test\_fold[,-14], type = 'class')

cm\_test <- table(test\_fold[,14], y\_pred\_test)

accuracy\_test <- sum(diag(cm\_test)) / sum(cm\_test)

y\_pred\_train <- predict(classifier, newdata = training\_fold[,-14], type = 'class')

cm\_train <- table(training\_fold[,14], y\_pred\_train)

accuracy\_train <- sum(diag(cm\_train)) / sum(cm\_train)

test\_pred <- c(test\_pred,accuracy\_test)

train\_pred <- c(train\_pred,accuracy\_train)

}

tst <- mean(test\_pred)

trn <- mean(train\_pred)

final\_pred\_train <- c(final\_pred\_train,trn)

final\_pred\_test <- c(final\_pred\_test,tst)

}

(df\_nn\_decay <- data.frame(Decay,final\_pred\_train,final\_pred\_test))

ggplot(df\_nn\_decay, aes(Decay)) +

geom\_line(aes(y=final\_pred\_test, color = "test")) +

geom\_line(aes(y=final\_pred\_train, color = "train")) +

scale\_color\_manual(values = c( "steelblue","darkred"))+

scale\_y\_continuous(name="Accuracy", limits=c(0.91, 1)) +

xlab('decay') + theme(legend.position = c(0.23, 0.1),

legend.direction = "horizontal")

#rang

final\_pred\_test <- c()

final\_pred\_train <- c()

Rang <- c(0,0.000001,0.00001,0.0001,0.001)

for(i in Rang){

test\_pred <- c()

train\_pred <- c()

for(x in folds){

training\_fold <- Heart\_train[-x, ]

test\_fold <- Heart\_train[x, ]

classifier <- nnet(Y~.,data = Heart\_train, Rang = i,size = 1,trace=F)

y\_pred\_test <- predict(classifier, newdata = test\_fold[,-14], type = 'class')

cm\_test <- table(test\_fold[,14], y\_pred\_test)

accuracy\_test <- sum(diag(cm\_test)) / sum(cm\_test)

print(accuracy\_test)

y\_pred\_train <- predict(classifier, newdata = training\_fold[,-14], type = 'class')

cm\_train <- table(training\_fold[,14], y\_pred\_train)

accuracy\_train <- sum(diag(cm\_train)) / sum(cm\_train)

#print(accuracy\_train)

test\_pred <- c(test\_pred,accuracy\_test)

train\_pred <- c(train\_pred,accuracy\_train)

}

y\_pred\_tst <- predict(classifier, newdata = Heart\_test, type = 'class')

cm\_tst <- table(Heart\_test$Y, y\_pred\_tst)

accuracy\_tst <- sum(diag(cm\_tst)) / sum(cm\_tst)

print(accuracy\_tst)

tst <- mean(test\_pred)

trn <- mean(train\_pred)

final\_pred\_train <- c(final\_pred\_train,trn)

final\_pred\_test <- c(final\_pred\_test,tst)

}

(df\_nn\_rang <- data.frame(Rang,final\_pred\_train,final\_pred\_test))

ggplot(df\_nn\_rang, aes(Rang)) +

geom\_line(aes(y=final\_pred\_test, color = "test")) +

geom\_line(aes(y=final\_pred\_train, color = "train")) +

scale\_color\_manual(values = c( "steelblue","darkred"))+

scale\_y\_continuous(name="Accuracy", limits=c(0.93, 1)) +

xlab('rang') + theme(legend.position = c(0.3, 0.1),

legend.direction = "horizontal")

#############################################################

library(caret)

nnGrid <- expand.grid(size = c(21:27),

maxit = seq(100,300,20),

decay = c(0,0.00001,0.0001,0.001,0.01))

#rang =c(0,0.000001,0.00001)

final\_pred\_test <- c()

final\_pred\_train <- c()

for(i in 1:nrow(nnGrid)){

for(x in folds){

training\_fold <- Heart\_train[-x, ]

test\_fold <- Heart\_train[x, ]

classifier <- nnet(Y~.,data = Heart\_train, size = nnGrid[i,1],maxit = nnGrid[i,2],

dacay = nnGrid[i,3])#,rang = nnGrid[i,4])

y\_pred\_test <- predict(classifier, newdata = test\_fold[,-14], type = 'class')

cm\_test <- table(test\_fold[,14], y\_pred\_test)

accuracy\_test <- sum(diag(cm\_test)) / sum(cm\_test)

y\_pred\_train <- predict(classifier, newdata = training\_fold[,-14], type = 'class')

cm\_train <- table(training\_fold[,14], y\_pred\_train)

accuracy\_train <- sum(diag(cm\_train)) / sum(cm\_train)

test\_pred <- c(test\_pred,accuracy\_test)

train\_pred <- c(train\_pred,accuracy\_train)

}

tst <- mean(test\_pred)

trn <- mean(train\_pred)

final\_pred\_train <- c(final\_pred\_train,trn)

final\_pred\_test <- c(final\_pred\_test,tst)

}

df\_tune\_nn <- nnGrid

df\_tune\_nn$train <- final\_pred\_train

df\_tune\_nn$test <- final\_pred\_test

head(arrange(df\_tune\_nn,desc(test)))

ggplot(df\_tune\_nn)

###Finally

tune\_nn <- nnet(Y~.,data = Heart\_train, size = 24, maxit = 100,

decay = 0)

plotnet(tune\_nn)

#plot(nn)

# Train confusion matrix & accuracy

pred\_train <- predict(tune\_nn, type='class')

(cm\_train <- table(prediction=pred\_train, truth=Heart\_train$Y))

(accuracy\_train <- (sum(diag(cm\_train)) / sum(cm\_train)))

cat('tune train accuracy:',accuracy\_train)

# Test confusion matrix & accuracy

pred\_test <- predict(tune\_nn, newdata = select(Heart\_test,-Y), type='class')

(cm\_test <- table(prediction=pred\_test, truth=Heart\_test$Y))

(accuracy\_test <- (sum(diag(cm\_test)) / sum(cm\_test)))

cat('tune test accuracy:',accuracy\_test)

#############################################################

set.seed(29)

fitControl <- trainControl(## 10-fold CV

method = "repeatedcv",

number = 10,

## repeated ten times

repeats = 10)

nnGrid <- expand.grid(size=c(21:27),

decay=c(0,0.00001,0.0001,0.001,0.01),

bag=c('TRUE','FALSE'))

nrow(nnGrid)

nnFit2 <- train(Y ~ ., data = Heart\_train,

method = "avNNet",

trControl = fitControl,

verbose = FALSE,

tuneGrid = nnGrid)

gbmFit2

gbmFit2$bestTune

beep(3)

### The optimal hyper parameter combination:

classifier <- train(form = Y ~ ., data = Heart\_train, method = 'avNNet')

classifier

classifier$bestTune

# Train confusion matrix & accuracy

pred\_train <- predict(classifier, type='raw')

(cm\_train <- table(prediction=pred\_train, truth=Heart\_train$Y))

(accuracy\_train <- (sum(diag(cm\_train)) / sum(cm\_train)))

cat('train accuracy:',accuracy\_train)

# Test confusion matrix & accuracy

pred\_test <- predict(nn, newdata = select(Heart\_test,-Y), type='class')

(cm\_test <- table(prediction=pred\_test, truth=Heart\_test$Y))

(accuracy\_test <- (sum(diag(cm\_test)) / sum(cm\_test)))

cat('test accuracy:',accuracy\_test)

#####################

##### K - Means #####

#####################

normalize <- function(x) {

return ((x - mean(x)) / (sd(x)))

}

HearT <- Heart

HearT$age <- normalize(HearT$age)

HearT$trestbps <- normalize(HearT$trestbps)

HearT$chol <- normalize(HearT$chol)

HearT$thalach <- normalize(HearT$thalach)

HearT$oldpeak <- normalize(HearT$oldpeak)

#####################

## Factors########

################

library(nnet)

HearT$gender <- class.ind(HearT$gender)

HearT$cp <- class.ind(HearT$cp)

HearT$fbs <- class.ind(HearT$fbs)

HearT$restecg <- class.ind(HearT$restecg)

HearT$exang <- class.ind(HearT$exang)

HearT$slope <- class.ind(HearT$slope)

HearT$ca <- class.ind(HearT$ca)

HearT$thal <- class.ind(HearT$thal)

HearT$BloodPressureRange <- class.ind(HearT$BloodPressureRange)

HearT$GoodMaxHeart <- class.ind(HearT$GoodMaxHeart)

##1

k.means.model <- kmeans(as.matrix(select(HearT,-Y)),centers = 2)

cm\_kmeans <- table(k.means.model$cluster-1,HearT$Y)

acc\_kmeans <- (sum(diag(cm\_kmeans)) / sum(cm\_kmeans))

cat('Accuracy = ',acc\_kmeans)

library(cluster)

clusplot(as.matrix(select(HearT,-Y)),k.means.model$cluster,color = T,shade = T,

labels = 0, lines = 0, main="K-Means")

## 2,3

wcss <- vector()

for(k in 2:10) wcss[k-1] <- sum(kmeans(select(HearT,-Y),k, nstart = 25)$withinss)

plot(2:10,wcss,type = 'b', main = paste('Cluster clients'),

xlab = 'Number of Clusters', ylab = 'WCSS')

library(clv)

dunn <- c(); DB <- c(); K <- 8

trn.kmeans <- select(HearT,-Y)

for(k in 2:K){

clust\_data <- kmeans(trn.kmeans, centers=k)

scatt\_data <- cls.scatt.data(trn.kmeans, clust=clust\_data$cluster,

dist='euclidean')

dunn <- c(dunn, clv.Dunn(scatt\_data, 'centroid', 'centroid'))

DB <- c(DB, clv.Davies.Bouldin(scatt\_data, 'centroid', 'centroid'))

}

clust\_metrics <- data.frame(K = rep(seq(2,K,1),2), value = c(dunn, DB),

metric = c(rep('Dunn',K-1), rep('DB',K-1)))

ggplot(clust\_metrics, aes(x=K, y=value, color=factor(metric))) +

geom\_point() + geom\_line()+

theme\_gray() + theme(legend.position = c(0.2, 0.4))

## 4

#############################

## Hierarchical Clustering ##

#############################

#Finding Optimal K Clusters:

dandrogram <- hclust(dist(select(HearT,-Y),method = 'euclidean'),

method = 'ward.D')

#minimise the variance for each cluster(ward.D)

plot(dandrogram,

main = paste('Dendogram'),

xlab = 'Pations',

ylab = 'Euclidean Distances')

######################################################

######################################################

######################################################

#######################

##### More Models #####

#######################

#########

## XGB ##

#########

Heart <- getSet(read.csv('Xy\_train.csv'))

library(caret)

inTraining <- createDataPartition(Heart$Y, p = .7, list = FALSE)

# create a stratified random sample of the data into training and test sets

training <- Heart[ inTraining,]

testing <- Heart[-inTraining,]

fitControl <- trainControl(## 10-fold CV

method = "repeatedcv",

number = 10,

## repeated ten times

repeats = 10)

library(gbm)

gbmFit1 <- train(Y ~ ., data = training,

method = "gbm",

trControl = fitControl,

## This last option is actually one

## for gbm() that passes through

verbose = FALSE)

gbmFit1

## Tuning Grids.

gbmGrid <- expand.grid(interaction.depth = c(3, 5, 7, 9),

n.trees = (1:10)\*10,

shrinkage = 0.1,

n.minobsinnode = c(10,15,20,25))

nrow(gbmGrid)

set.seed(29)

gbmFit2 <- train(Y ~ ., data = training,

method = "gbm",

trControl = fitControl,

verbose = FALSE,

## Now specify the exact models

## to evaluate:

tuneGrid = gbmGrid)

gbmFit2

gbmFit2$bestTune

## Plotting the Resampling Profile:

#The plot function can be used to examine the relationship between

#the estimates of performance and the tuning parameters.

trellis.par.set(caretTheme())

plot(gbmFit2)

# a heatmap of the results:

trellis.par.set(caretTheme())

plot(gbmFit2, metric = "Accuracy", plotType = "level",

scales = list(x = list(rot = 90)))

# try this:

ggplot(gbmFit2)

############

#Predicting#

############

predict(gbmFit2, newdata = head(testing))

predY <- predict(gbmFit2, newdata = testing, type = "raw")

cm <- table(predY,testing$Y)

(Accuracy\_gbm <- sum(diag(cm))/sum(cm))

###################

## Random Forest ##

###################

Heart <- getSet(read.csv('Xy\_train.csv'))

## Split the Dataset to training & test sets:

library(caTools)

set.seed(29)

split <- sample.split(Heart$Y ,SplitRatio = 0.7)

Heart\_train <- subset(Heart,split == T)

Heart\_test <- subset(Heart,split == F)

library(randomForest)

rf <- randomForest(Y~.,Heart\_train)

rf\_pr <- predict(rf,newdata = Heart\_test,type = 'class')

(cm <- table(rf\_pr,Heart\_test$Y))

cat('Accuracy =',sum(diag(cm)) / sum(cm))

rf\_pr <- predict(rf,newdata = Heart\_train,type = 'class')

(cm <- table(rf\_pr,Heart\_train$Y))

cat('Accuracy =',sum(diag(cm)) / sum(cm))

library(caret)

set.seed(29)

folds <- createFolds(Heart\_train$Y, k = 30)

## Tune mtry, ntree:

Mtry <- c(1:4)

Ntree <- c(350,400,450,500,525,550,600)

ntree\_col <- c()

mtry\_col <- c()

final\_pred\_train <- c()

final\_pred\_test <- c()

for(i in Mtry){

for(j in Ntree){

test\_pred <- c()

train\_pred <- c()

for(x in folds){

training\_fold <- Heart\_train[-x, ]

test\_fold <- Heart\_train[x, ]

classifier <- randomForest(Y~.,data = training\_fold, mtry=i,ntree=j)

y\_pred\_test <- predict(classifier, newdata = test\_fold[,-14], type = 'class')

cm\_test <- table(test\_fold[,14], y\_pred\_test)

accuracy\_test <- sum(diag(cm\_test)) / sum(cm\_test)

#print(accuracy\_test)

y\_pred\_train <- predict(classifier, newdata = training\_fold[,-14], type = 'class')

cm\_train <- table(training\_fold[,14], y\_pred\_train)

accuracy\_train <- sum(diag(cm\_train)) / sum(cm\_train)

#print(accuracy\_train)

test\_pred <- c(test\_pred,accuracy\_test)

train\_pred <- c(train\_pred,accuracy\_train)

}

tst <- mean(test\_pred)

trn <- mean(train\_pred)

final\_pred\_train <- c(final\_pred\_train,trn)

final\_pred\_test <- c(final\_pred\_test,tst)

ntree\_col <- c(ntree\_col,j)

mtry\_col <- c(mtry\_col,i)

}

}

(df <- data.frame(ntree\_col,mtry\_col,final\_pred\_train,final\_pred\_test))

ggplot(df, aes(ntree\_col)) +

geom\_line(aes(y=final\_pred\_test,color = 'red'))+

#geom\_line(aes(y=final\_pred\_train,color='blue'))+

facet\_wrap(~mtry\_col)+

#theme(legend.position = c(0.9, 0.1))+

ylab('Accuracy') + xlab('ntree')

rf <- randomForest(Y~.,Heart\_train,ntree = 550,mtry=2)

rf\_pr <- predict(rf,newdata = Heart\_test,type = 'class')

(cm <- table(rf\_pr,Heart\_test$Y))

cat('test Accuracy =',sum(diag(cm)) / sum(cm))

#########

## SVM ##

#########

svm\_model\_train <- svm(Y ~. ,data = Heart\_train)#, kernel = 'linear')

pred\_svm\_model <- predict(svm\_model\_train, newdata = Heart\_train)

CM <- table(pred\_svm\_model,Heart\_train$Y)

cat(' Accuracy <-' ,sum(diag(CM)) / sum(CM))

pred\_svm\_model <- predict(svm\_model\_train, newdata = Heart\_test)

CM <- table(pred\_svm\_model,Heart\_test$Y)

cat(' Accuracy <-' ,sum(diag(CM)) / sum(CM))

#####tune

obj <- tune.svm(Species~., data = iris, gamma = 2^(-4:1), cost = seq(2,20,1),

kernel=c('linear','polynomial','radial','sigmoid'))

obj$best.parameters

svm\_model\_train <- svm(Y ~. ,data = Heart\_train,

gamma=0.125,cost=2,kernel='radial')#, kernel = 'linear')

pred\_svm\_model <- predict(svm\_model\_train, newdata = Heart\_train,

type='class')

CM <- table(pred\_svm\_model,Heart\_train$Y)

cat(' Accuracy <-' ,sum(diag(CM)) / sum(CM))

pred\_svm\_model <- predict(svm\_model\_train, newdata = Heart\_test,type='class')

CM <- table(pred\_svm\_model,Heart\_test$Y)

cat(' Accuracy <-' ,sum(diag(CM)) / sum(CM))