

Ensemble Methods

שרון מלטר, אתגר 17

10 באוגוסט 2024

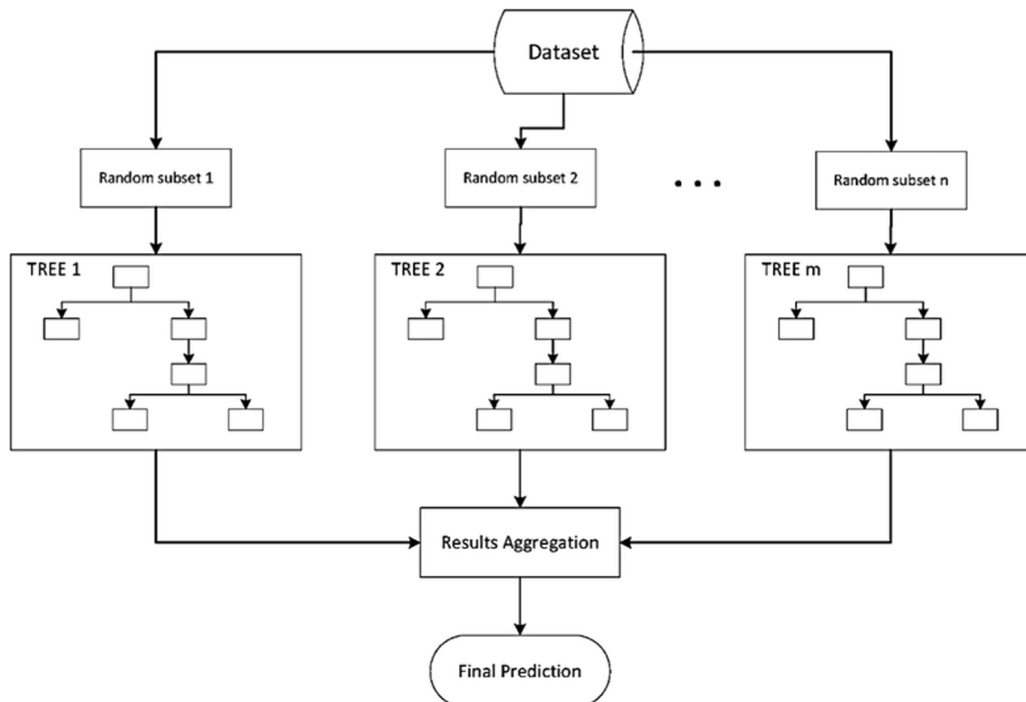
1 רקע

נניח שיש לנו מספר מודלים לקלסיפיקציה או רגרסיה. אנחנו יודעים איך להשתמש בכל אחד כשלעצמו, אך מה אם ניתן להשיג תוצאות טובות יותר אם נשלב אותם? *aka*, כפי שלמדנו בשיטת *Multiclass SVM*, ניתן להשתמש במספר מודלים ולסווג את הדוגמיות לפי המחלקה בה הן קיבלו את הניקוד הגבוה ביותר. יש לכך מספר יתרונות:

- ניתן לשפר דיוק.
- משתמשים בסוגים שונים של אלגוריתמים.
- מימוש פשוט.
- נדרש מספר מועט של פרמטרים.
- ניתן להקטין קורלציה ו- *overfitting*.

2 BAGging (Bootstrap Aggregating)

בהינתן N דוגמיות מסווגות, מתוכן נבחר מספר *bootstrapped subsamples*. כלומר, נבחר n תתי קבוצות ממסד הנתונים, עם חזרה, ולכל תת-קבוצה $i \in \{0, \dots, n\}$ נאמן מודל סיווג C_i . בשביל הסיווג הכללי עבור דוגמית חדשה, נספור את כמות ההצבעות מכל C_i ונבחר במחלקה שקיבלה את מספר הקולות הגבוה ביותר. ומה עם רגרסיה? - לוקחים את ממוצע הערכים של הרגרסורים :) כל התהליך היפה הזה מכונה *BAGging*. להלן שרטוט המציג אותו ואת חתיכותו;



הסבר: בהינתן מסד נתונים, בוחרים ממנו n קבוצות של דוגמיות. כל אחת מועבר למודל אחר (עץ אחר) ולבסוף משתמשים בכל הפלטים של אותם מודלים על מנת לסווג דוגמיות. מעבר ליופי, יש ב- *BAGging* מספר יתרונות;

- ניתן לייעל את ביצועי המודלים שמשמשים בהם כאשר אינם יציבים. כלומר, כאשר שינויים קטנים ב- *training data* משנים באופן משמעותי את המודל המתקבל.
- הגישה מפחיתה *variance* (שונות) כלומר ניתן להפחית *over fitting*.
- פשוט לממש (:

אך יש גם חסרונות:

- ייתכן *bias* (הטיה) משמעותי אם לא ממדלים את אופן השימוש במודלים האחרים, למשל הענקת מספר קטן מדי של דוגמיות למודלים, כך שרבים מהם לא מבינים את מורכבות הנתונים. במקרה זה נקבל *underfitting*.
 - ייתכן שזמן החישוב יהיה יקר (מאחר שמשמשים במספר מודלים)
- אבל *BAGging* הינו לא המודל היחיד עם אותה הגישה. אולי ניתן קצת לשנות את המימוש?

3 Random Forest

זהו מודל שניתן לחשוב עליו כאל *BAGging* עם טוויסט (!) ב- *BAGging*, לכל תת מסווג / רגרסור תלוי בכל הפיט'רים של הדאטה. לעומת זאת, ב- *Random Forest*, המודלים מקבלים דוגמיות ביחס לסלקציה רדומלית של פיט'רים. כלומר, כל עץ מקבל דוגמיות עם m פיט'רים ורק הם ניתנים למסווג. לרוב מגדירים $m = \sqrt{d}$ או $m = \log_2 d$ כאשר d הוא מספר הפיט'רים (המימד של מרחב הקלט) שימו לב שבגישה זו אין קיצוץ, כלומר אנו בוחרים את הפיט'רים לכל מודל שרירותית ולא זורקים אותם.

ועכשיו לשיפוט. יתרונות;

- שיטה זו עובדת טוב אם לכל עץ יש אחוז שגיאה נמוך באופן יחסי.
- השיטה מקטינה קורלציה בין עצים.
- השיטה מקטינה *over fitting*.
- ניתן להשתמש בה כדי לזהות את הפיט'ר החשוב ביותר מבין הקיימים.
- השיטה גמישה (ניתן לבצע שינויים רבים ובאופן פשוט)

חסרונות;

- השיטה יכולה להיות איטית, מאחר שמשמשים במספר עצים. באופן כללי, ככל שמספר העצים גדל, האלגוריתם מאט והופך לפחות אפקטיבי.
- השיטה רגישה לבחירת ה- *hyperparameter m*, כלומר תלויה מאוד ב- m המתאים לדאטה.

עד כאן דיבורים. בואו נראה מה הבנו עם שאלה!

4 חידה

הוכיחו שבשיטת ה- *BAGging*, כאשר גודל הדוגמית (דוגמית יכולה להיות נתון אחד או יותר) הוא $n = N$, כלומר כאשר מספר הדוגמיות שמקבל כל תת-מודל שווה למספר הדוגמיות הכולל, לפחות 63% מהדוגמאות המקוריות מופיע בדאטה של תת מודל כלשהו. (ולא 100% מכיוון שהבחירה הינה עם חזרה) הוכחה בדף הבא.

הוכחה:

נניח שנתוני האימון שלנו הם $S = \{(x_i, y_i) | i = 1, \dots, n\}$ ואנו בוחרים n מתוך n דוגמיות, ללא סדר ועם חזרה. נניח ש- S_i היא התוצאה. ההסתברות שדוגמה (x_i, y_i) לא מופיעה ב- S_i היא $(1 - \frac{1}{n})^n$, מכיוון שבמשך n פעמים נבחרת ל- S_i דוגמית אחת באופן רנדומלי מבין n אפשרויות. לאחר מכן נקבל כי-

$$(1 - \frac{1}{n})^n \approx e^{-1} \approx 0.37$$

כלומר בערך 37% מהדאטה לא נמצאת ב- S_i .

עד כאן שיח תאורטי, בואו נראה מימוש בקוד (יאי!)

5 מימוש Random Forest

```
from sklearn.ensemble import RandomForestClassifier

RFclassifier = RandomForestClassifier(n_estimators=3, max_depth=4, random_state=42)
RFclassifier.fit(X_trainset, y_trainset)
```

```
RandomForestClassifier
RandomForestClassifier(max_depth=4, n_estimators=3, random_state=42)
```

כאן אנו משתמשים במודל *Random Forest* מ-*sklearn* ומתאימים אליו את דוגמיות האימון. נמצא את דיוק המודל:

```
y_pred = RFclassifier.predict(X_testset)
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_testset, y_pred))
```

הפלט: *DecistionTree's Accuracy* : 0.9666666666666667

נחמד. אבל עבור אותה דאטה, ניתן להשיג גם דיוק של 98.33% מעצי החלטה. ננסה לשפר את המימוש בעזרת *cross validation*. כלומר, נלמד ממספר אופטימלי של מודלים.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

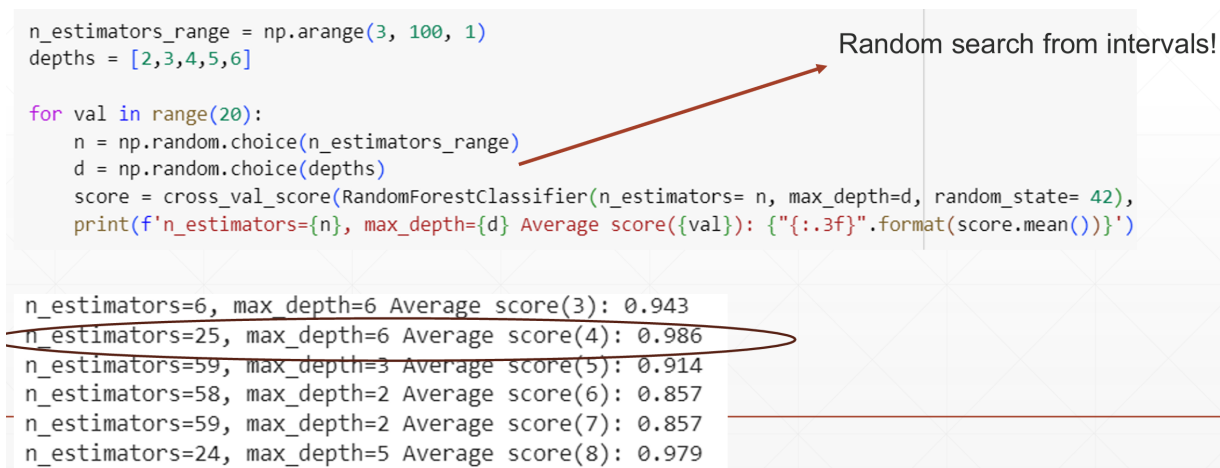
kf = KFold(n_splits=5, shuffle=True, random_state=42)

n_estimators = np.arange(1, 100, 10)

for val in n_estimators:
    score = cross_val_score(RandomForestClassifier(n_estimators= val, random_state=
    print(f'Average score({val}): {"{:.3f}".format(score.mean())}')
```

```
Average score(1): 0.914
Average score(11): 0.950
Average score(21): 0.993
Average score(31): 0.986
Average score(41): 0.986
```

הסבר: נעלה מ- *sklearn* מתת הספרייה *model_selection* (אשר מספקת כלים לבחירת מודלים) את *KFold* ו- *cross_val_score*. הפעולה *KFold(n_splits = 5, *, shuffle = False, random_state = None)* של *train/test* ל- *k* תתי-קבוצות רציפות (ולא הגרלה באופן דיפולטיבי) נגריל את הדאטה ל- 5 כדי למנוע הטיה הנובעת ממיקום הדוגמיות. לאחר מכן בודקים את דיוק ההערכה של *Random Forest* עבור שימוש ב- $n \in \{1, \dots, 100\}$ תתי-מערכים בכל פעם. אז הגענו למטרה הקודמת. אך האם ניתן לשפר גם מקרים עם פחות מערכים, באמצעות אופטימיזציה של עומק העצים?



```

n_estimators_range = np.arange(3, 100, 1)
depths = [2,3,4,5,6]

for val in range(20):
    n = np.random.choice(n_estimators_range)
    d = np.random.choice(depths)
    score = cross_val_score(RandomForestClassifier(n_estimators=n, max_depth=d, random_state=42),
    print(f'n_estimators={n}, max_depth={d} Average score({val}): "{:.3f}".format(score.mean())')

n_estimators=6, max_depth=6 Average score(3): 0.943
n_estimators=25, max_depth=6 Average score(4): 0.986
n_estimators=59, max_depth=3 Average score(5): 0.914
n_estimators=58, max_depth=2 Average score(6): 0.857
n_estimators=59, max_depth=2 Average score(7): 0.857
n_estimators=24, max_depth=5 Average score(8): 0.979

```

Random search from intervals!

הסבר: בוחרים באופן רנדומלי עומקים ומספר תתי-מערכים ובודקים מהו הדיוק המתקבל עבור אותו זיווג.