# Feature Extraction and Selection

*תקציר: במהלך ההכשרה, אסכם מספר נושאים "רוחביים" שאינם מתקשרים לפרק ספציפי. הנושא החמישי הוא יצירה ובחירה של פיצ'רים. במסמך זה אסביר את הקושי בנ"ל ואסקור שיטות לבחירה אוטומטית של פיצ'רים.*

## הבעיה

בחירת פיצ'רים זה תהליך של בחירת משתני הקלט אשר בהם נעשה שימוש בעת אימון המודל (תוך התעלמות מאלו שלא בחרנו). הצורך בבחירה של פיצ'רים נובע מכך שלעיתים יש לנו הרבה מאד פיצ'רים, שלא כולם רלוונטיים לבעיה. עודף בפיצ'רים עלול לפגוע בהליך האימון, כמו גם להכביד מבחינה חישובית (זמן וכו'). יצירת פיצ'רים זה תהליך יצירת אותן עמודות (לרוב על ידי שימוש בפיצ'רים אחרים, פחות אינפורמטיביים). תמונה שממחישה את הצורך בבחירת פיצ'רים טובים מוצגת באיור 1.

|  |
| --- |
| Image for post |

איור 1: איור של ביצועי מודל ביחס לכמות הפיצ'רים

## שיטות לבחירת פיצ'רים

ישנן כמה שיטות לבחירה של פיצרים

### ידנית

ניתן לחשב את הקורלציות שבין הפיצ'רים למשתנה המטרה ולבחור את אלו להם יש קורלציה גדולה. בנוסף, אם יש פיצ'רים אשר ביניהם יש קורלציה גבוהה, ניתן להשמיט אחד מהם כי כנראה שלא יהיה ערך בלשמור את שניהם. פונקציה שעושה את הבחירות הנ"ל ידנית (בעזרת מתן פונקציה על פיה נדרג את הפיצ'רים כמו chi2 היא GenericUnivariateSelect של sklearn).

### מודלים שעל הדרך מראים חשיבות פיצ'רים

חלק מהמודלים (למשל, xgboost) מאפשרים גישה לחשיבות של הפיצ'רים לפי מידת השימוש בהם במהלך בניית המודל. פיצ'רים אלו היו המשמעותיים ביותר במודל ולכן ניתן להניח כי הם חשובים. פונקציה נחמדה שמקלה על החיים: SelectFromModel של sklearn.

### אלגוריתמים לבחירה אוטומטית של פיצ'רים

#### Recursive Feature Elimination

בכל איטרציה נבנה מודל ונחשב את חשיבות הפיצ'רים (כמו שתואר לעיל), לאחר מכן נוריד את הפיצ'ר הכי פחות חשוב ונחזור על התהליך עד שנקבל את מספר הפיצ'רים שנרצה. היתרון שבאיטרציות הוא שבמידה ויש פיצ'רים שבהתחלה נראים לא חשובים אבל בהמשך (לאחר שנצמצם את מספר הפיצ'רים) יהיו יותר חשובים אז נשמור אותם (לעומת איטרציה יחידה בה מורידים את כל הפיצ'רים הלא חשובים ביחד). החיסרון הוא שהאלגוריתם איטי. פונקציה מ sklearn שמממשת את זה: RFE (או RFECV שעושה אותו דבר אבל עם cross validation בכל שלב).

#### Boruta

אלגוריתם שמטרתו למצוא את הפיצ'רים הרלוונטים (במקום אלו שיהיו הכי טובים למשימה). האמירה פה היא שלמצוא תת קבוצה של פיצ'רים זו משימה קשה מדי ויותר פשוט למצוא תת קבוצע של פיצ'רים אשר כולם בה תורמים למטרה בצורה כלשהי. האלגוריתם עובד בצורה הבאה: בכל איטרציה מוסיפים למידע מספר פיצ'רים רנדומלים (לפי המימוש, לפחות 5 בכל פעם) שיהיו רעש. כעת, מריצים random forest ומוציאים ממנו את חשיבות הפיצ'רים (כולל חשיבות הרעש שהוספנו). נגדיר את החשיבות של פיצ'ר הרעש הכי חשוב בתור חשיבות סף. לכל פיצ'ר שעוד לא נקבע אם הוא חשוב, נבצע מבחן מובהקות סטטיסטי על מנת לבדוק האם הפיצ'ר חשוב פחות או יותר מחשיבות הסף. אם פיצ'ר נמצא כחשוב פחות (בצורה מובהקת) אז נמחק אותו, אם נמצא חשוב יותר (בצורה מובהקת) אז נסמן אותו כחשוב ואחרת לא נעשה כלום ונחזור על התהליך. נפסיק כאשר נגיע למצב בו כל הפיצ'רים (שנשארו) הוכרזו כחשובים או שנתקענו. מודל boruta מממש את האלגוריתם עם הפונקציה BorutaPy.

## ניסוי

השתמשתי ב digits (בערך 175 דוגמאות לכל ספרה). שיערוך הביצועים בוצע בעזרת 10FCV. עבור כל אלגוריתם לקחתי 46 פיצ'רים מתוך ה 64 (כי boruta לקח 46 פיצ'רים ושאר השיטות מקבלות את מספר הפרמטרים כפרמטר. זמן הריצה של boruta היה משמעותית ארוך יותר (תלוי מספר איטרציות). בכל המקרים המודל שעשה שימוש בפיצ'רים הנבחרים היה עץ החלטה עם היפר-פרמטרים שנבחרו ב sklearn כברירת מחדל. המול שהורץ ב- boruta היה random forest התוצאות מצורפות באיור 2.

|  |  |
| --- | --- |
| אלגוריתם | דיוק |
| Chi2 | 0.817 |
| Random forest | 0.822 |
| Boruta | 0.827 |

איור 2: ביצועי האלגוריתמים

## חבילות רלוונטיות

### FeatureTools

תומך ביצירת פיצ'רים אוטומטית. כל dataframe הוא entity בתוך entityset. ניתן ליצור מentity פיצ'רים שמהווים שילובים של עמודות מאותו DF (מכפלה, חיבור וכו'). בנוסף, ניתן להגדיר יחס בין DF על מנת ליצור פיצ'רים אגרגטיביים.

### TPOT

משתמש באלגוריתם חיפוש גנטי על מנת למצוא את האלגוריתם (וההיפר-פרמטרים המתאימים) הכי טובים. מבצע 10FCV לכל pipeline שמנסה. מאמן את הכי טובים שמצא על כל הדאטה. מבחינת סינטקס הוא מאד נוח כי הוא בנוי על sklearn והשימוש מאד דומה. מצד שני, מזהירים שהרצה של TPOT יכולה להיות ארוכה במקרה של מאגר מידע גדול. מנסיון שלי עם wine dataset, עצרתי את הריצה לאחר 8 דקות והגעתי ל pipeline עם ביצועים טובים יחסית (0.96). נראה קלאסי להרצות סופ"ש.

### AutoSKlearn

נראה כמו TPOT מבחינת שימוש וייעוד. חיפוש pipeline מתבצע בעזרת אופטימיזציה באיסייאנית.

### AutoPytorch

מהמפתחים של AutoSklearn אבל בשביל Pytorch. לא מצאתי יותר מידי מדריכים באינטרנט או תיעוד מועיל. ב source כתוב שזו גרסת אלפה מוקדמת (נכון לכתיבת מסמך זה, הגיט שלהם נערך לפני 25 ימים), אולי בעתיד יהיה שווה להסתכל על זה.

### AutoKeras

כמו הקודמים (מפתחים אחרים) אבל ל keras. קשה לי לאמוד את זה אבל נראה שהמודל הזה יותר בשל ויותר מוכר מ AutoPytorch.