Decision Trees Features Subset Selection

Project in AI
236502

Email Address	Identity card	Student Name
altory@campus.technion.ac.il	315636829	Muhammad Altory
quosai@campus.technion.ac.il	208526814	Quosai Dahamshi
muhammad.dah@campus.technion.ac.il	318396926	Muhammad Dahamshi

Supervisor:

Prof. Shaul Markovitch

הקדמה:

הפרויקט שלנו עוסק בבחירת תכונות מתוך קבוצת אימון נתונה, בהינתן קבוצת אימון אנו נבחר את התכונות הכי משפיעות על קבלת ההחלטה או הסיווג ואז נאמן את המסווג שלנו.

ממשו את האלגוריתמים הבאים לבחירת תכונות. כל האלגוריתמים מקבלים קבוצת אימון T בגודל N וארגומנט $0 \leq p \leq 1$ כל האלגוריתמים מחזירים תת קבוצה של התכונות בגודל $K = \lfloor p \cdot T \rfloor$.

תהליך בחירת התכונות יכול לעזור כמעט לכל סוג של מסווגים , התהליך נעשה לפני תחילת האימון.

השיטה הטריוויאלית לבחירת תכונות היא לקחת כל תת קבוצה אפשרית של התכונות ולאמן את המסווג על תת קבוצה זו ולבדוק את הדיוק של המסווג , אבל שיטה זו לוקחת הרבה מאוד זמן , לכן אנו נעדיף להשתמש באלגוריתמים מתקדמים על מנת לייעל את תהליך בחירת התכונות.

הסיבות שעבורן אנו נרצה לסנן תכונות:

1. מחיקת תכונות מיותרות , כלומר אלה שלא עוזרות לנו להפריד בין המחלקות השונות יכול להביא את המסווג לדיוק יותר טוב.
2. מאפשרת לנו לחסוך זמן בעת אימון המסווג וגם זמן החיזוי יהיה יותר קטן כי אנו נשתמש רק בחלק מן התכונות

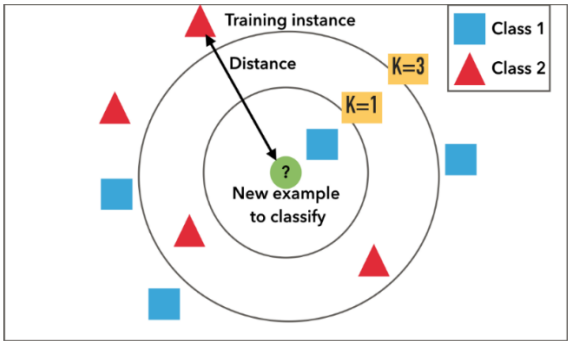
אנו ננסה לבחון את ההשפעה של בחירת אחוז מסוים מהתכונות על רמת הדיוק של המסווג, נשתמש בשמונה אלגוריתמים שונים על מנת לסנן חלק מהתכונות ולבדוק את הביצועים של כל אחד מהם.

בפרויקט שלנו אנו נבחן את התהליך של בחירת תכונות על רמת הדיוק של המסווג KNN.

רקע

בכל פעם אנחנו בוחרים תכונות אנחנו נרצה לבדוק עד כמה הם טובים, אפשר להשתמש בכל אלגוריתם למידה כלשהו כדי לאמן אותו עם תכונות אלה ולבדוק את הדיוק שלו אנחנו השתמשנו ב KNN מסווג השכן הקרוב ביותר.

K-NN: זהו אלגוריתם למידה אשר בהינתן קלט של דוגמה חדשה הוא מחשב את המרחק שלה משאר הדוגמאות שמהוות קבוצת, את המרחק ניתן למדוד בדרכים שונות השיטה שאנחנו נשתמש בה נקראת המרחק האיקלידי, הוא מסווג את הדוגמה לפי הסיווג של k הדוגמאות הקרובות ביותר, K הוא היפר פרמטר אשר נקבע לפני תחילת האימון, זמן האימון של אלגוריתם זה אינו גדול אבל הסיווג יקר בכל שיש לנו הרבה דוגמאות באימון.



נרמול dataset: ב k-nn כאשר רוצים לסווג דוגמה אנחנו מחשבים את המרחק שלה משאר הדוגמאות וניקח את k הדוגמאות הקרובות ביותר והסיווג יהיה לפי הסיווג של רוב הדוגמאות k , אם יש שוני בין סדרי הגודל של התכונות יכול להיות שיהיו יותר משפיעות על קבלת החלטות לעומת תכונות עם ערכים קטנים , הפתרון הוא לנרמל את ה dataset על מנת שכל התכונות יהיה להם אותו סדר גודל , לכל תכונה אנחנו נחשב את הממוצע של הערכים שלה וסטיית התקן ונרמל את התכונה לפי ערכים אלה .

Entropy:

יהי X משתנה רנדומי עם ערכים $x_1, \ldots, x_n$

האנטרופיה X של מוגדרת: $H(X) = -\sum_{i=1}^{n} p(x_i) \cdot \log_2(p(x_i))$

כאשר מגדירים $0 \cdot log_2(0) \triangleq 0$

IG (information-gain) - תוספת האינפורמציה של תכונה: אחת הדרכים לבדוק את יכולת תכונה מסוימת להפריד בין מחלקות שונות היא להשתמש ב IG, חלק מהאלגוריתמים שמימשנו מסתמכים באופן חזק במדד זה כדי לבחור תכונות, ה IG מוגדר להיות כירידה המשוקללת באנטרופיה כתוצאה מפיצול הדוגמאות לקבוצות שנוצרו עבור ערכי התכונה.

$$\mathrm{IG}(f, E) = H(E) - \sum_{i=1}^{k} \frac{|E_i|}{|E|} \cdot H(E_i)$$

Cross-validation (CV): זו שיטה כדי לבדוק את רמת הדיוק של מסווג מסוים, אנחנו נחלק ה data שלנו ל K חלקים, K פרמטר שאפשר לכוונן, בהינתן $data - set$ מסוים, נסמן את קבוצת כל הדוגמאות שבו ע"י $Examples$. כמו כן, עבור כל $data - set$ נחלק את הדוגמאות שבו ל-m $folds$ שיישארו **קבועים** לאורך כל הניסויים ונסמנם: $Folds = \{fold_1, fold_2, \dots, fold_m\}$. כאשר נרצה לחשב את ערך הדיוק של אלגוריתם למידה מסוים או ועדה הומוגנית מסוג נפעל באופן הבא, לכל $i \in \{1, 2, \dots, m\}$:

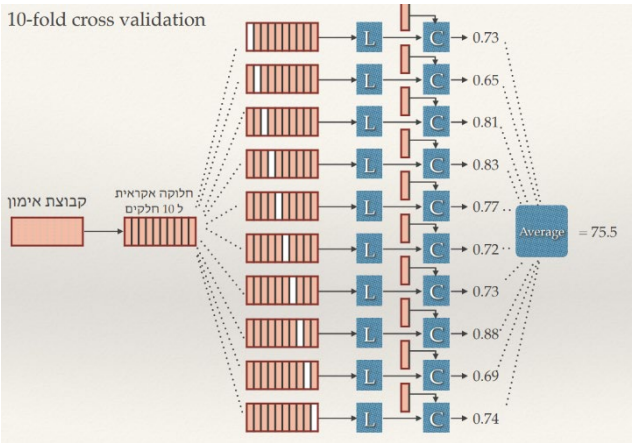
א. הפעל את אלגוריתם הלמידה כאשר קבוצת הדוגמאות המשמשת ללמידה (קבוצת האימון) היא הקבוצה $\text{train}_i = \text{Examples} \backslash \text{fold}_i$, כלומר קבוצת הדוגמאות שאינן שייכות ל-$fold_i$.

ב. חשב את אחוזי הדיוק ($accuracy$) של המסווג על קבוצת הבדיקה שתהא ה-$fold$ ה-i, כלומר $test_i = fold_i$, נסמן אחוזי דיוק אלה ב- acc_i.

כעת, הגדר את ערך הדיוק של אלגוריתם הלמידה להיות הממוצע בין ערכי הדיוק שנמצאו, כלומר:

$$accuracy = \frac{1}{m}\sum_{i=1}^{m} acc_i$$

הלמידה וכן החישוב של אחוזי הדיוק מתבצעים לאחר שהחלוקה ל-$folds$ כבר בוצעה.



מתאם ספירמן- Spearman's Rank Coefficient of Correlation (r_s) מקדם המתאם של ספירמן הוא מדד קשר אשר בודק את <u>הקשר בין שני משתנים כשאר כל אחד מהם נמדד בסולם אחר</u>, אנחנו נשתמש במקדם זה בחלק מהאלגוריתמים על מנת לבדוק את הקשר בין שתי תכונות כאשר כל אחת מהן היא בסדר גודל אחר, ערכי מתאם ספירמן נעים בין $-1 \le r_s \le 1$, אשר מחושבים על סמך הקשר בין הדירוגים של ערכי המשתנים, ולא בין הערכים עצמם.

הערך הנמוך ביותר במשתנה X יקבל את הדירוג 1, הערך השני את הדירוג 2, וכן הלאה. כך גם עבור הערכים במשתנה Y. עבור מתאם חיובי, נצפה שהדירוג של משתנה אחד יתאים לדירוג של המשתנה האחר, ואילו עבור מתאם שלילי, נצפה שעבור דירוגים גבוהים במשתנה האחד, נקבל דירוגים נמוכים במשתנה השני, ולהיפך.

לאחר שהתקבלו התוצאות, יש להמיר את הערכים שהתקבלו בכל אחד מהמשתנים לדירוגים. לבסוף, על מנת לחשב את מדד ספירמן עצמו, יש לחשב את ההפרשים בין הדירוגים d_i ולאחר מכן את ריבוע ההפרשים d_i^2.
לבסוף, יש להציב בנוסחה של מתאם ספירמן:

$$r_s = 1 - \frac{6 \cdot \Sigma d_i^2}{n \cdot (n-1)}$$

בפרויקט שלנו השתמשנו בפונקציה spearmanr של הספרייה scipy.stats.

אלגוריתמים

בכל האלגוריתמים השתמשנו במסווג KNN.

spearman min_distance (feature, window) – פונקציה זו מוגדרת כך בהינתן משתנה feature וקבוצת משתנים res אנחנו נחשב מקדם ספירמן בין המשתנה feature לכל אחד מהתכונות ב window והתוצאה תהיה מקדם ספירמן המינימלי מבין כל הערכים שקיבלנו, משתנה יכול להיות ויקטור במקרה שלנו feature וקטור עמודה ו window קבוצה של ויקטורי עמודה.

MaxIG (window, k) - מחזירה את k התכונות מתוך החלון שיש להן IG הגדול ביותר. הקלט של כל אלגוריתם dataset ו k מספר התכונות שעל האלגוריתם לבחור והפלט k התכונות הטובות ביותר לפי החישוב של האלגוריתם.

מימוש האלגוריתמים:

1.Random

אלגוריתם טריוויאלי, באלגוריתם זה בוחרים באקראי קבוצה של תכונות:

```
res= random(features)
return res
```

2.MaxIG

באלגוריתם זה אנחנו נבחר קבוצה של תכונות אשר יכולות לתת הפרדה מקסימלית, הרעיון מאחורי אלגוריתם זה לסנן את התכונות עם IG נמוך, כלומר אלה שלא עוזרות לנו להפריד בין המחלקות השונות:

```
Sort features by IG
res = top k features
return res
```

3.Diverse1

באלגוריתם זה אנחנו נבחר חלון התחלתי עם קבוצה בגודל 2K שיש להם IG מקסימלי והמטרה היא לבחור תכונות מתוך חלון זה שלא תלויות אחת בשנייה תוך שימוש ב spearman correlation, את הפתרון אנחנו נאתחל עם התכונה שיש לה IG מקסימלי ובכל פעם אנחנו נוסיף לפתרון את התכונה הכי רחוקה מהתכונות שנמצאות בפתרון, הרעיון מאחרי אלגוריתם זה לקחת תכונות שעוזרות לנו להפריד שלא תלויות אחת בשנייה:

```
Window = MaxIG(features,2*k) // size = 2k
res = [MaxIG(Window,1)]
Repeat K-1:
        For feature in Window:
                calculate spearman min_distance(feature,res)
        add feature with maximum distance to res
return res
```

4.Diverse2:

באלגוריתם זה אנחנו נאתחל חלון עם התכונות הכי רחוקות אחת מהשנייה ומתוכם אנחנו נבחר את אלה עם ה IG המקסימלי, באלגוריתם זה לעומת האלגוריתם הקודם אנחנו ניקח בהתחלה את התכונות שהכי רחוקות אחת משנייה כי יכול להיות שהתכונות עם IG מקסימלי יהיו תלויות אחת בשנייה וזה לא יעזור לנו, ואז בוחרים אלה עם IG מקסימלי :

```
res = sum spearman min_distance between each feature pair is maximum // size= 2k
Sort features in res by IG
select top k features
return res
```

5.WrapperForward

באלגוריתם זה אנחנו נשתמש בשיטת אלגוריתם חמדני לבחירת תכונות, אנחנו נתחיל עם קבוצה ריקה בכל שלב אנחנו נוסיף תכונה ונאמן את המסווג שלנו ונבחר התכונה שמעלה את הדיוק הכי הרבה:

```
res = []
Features = all features
repeat K:
        For feature in Features\res:
                classifier = train knn with res+feature
                Test classifier with cross-validation
        add to res feature with maximum accuracy
return res
```

6.WrapperBackward

אלגוריתם זה פועל כמו אלגוריתם קודם, רק במקום להוסיף תכונה עם דיוק מקסימלי אנחנו מורידים תכונה שנותנת לנו דיוק מנמלי, כלומר מנסים להוריד תכונות הפוגעות בביצועים של המסווג שלנו

```
result = all features
Repeat N-k:
        best_accuracy = -inf
        worst_feature = {}
        For feature in result:
                classifier = train with result\{feature}
                accuracy = Test classifier with cross-validation
                if(accuracy >= best_accuracy):
                        best_accuracy = accuracy
                        worst_feature = {feature}
        result = result \ worst_feature
return result
```

7.LocalSearch

באלגוריתם זה אנחנו מתחילים עם קבוצת תכונות רנדומלית ובכל פעם ננסה להחליף תכונה בתכונה אחרת שלא נמצאת בקבוצה, אנחנו נבחר את ההחלפה הראשונה שמביאה לשיפור (first choice hill climbing), בחרנו להשתמש בשיטה זו כי המרחב שלנו מאוד גדול ושיטה זו יעיל מבחינת זמן, מקדם הסיעוף בבעיה שלנו יחסית גדול $\boldsymbol{k(n-k)}$ לכן שימוש באלגוריתמי חיפוש אחרים בכלל לא יעיל.

```
res = Random(features) // size k
classifier = train knn with res features
current_accuracy = test classifier with cross validation
Repeat 10 times:
        For current in res:
                For candidate_feature in Features\res:
                        classifier = train(U (res, candidate_feature)\ current)) # try to swap
                        candidate_accuracy = test classifier with cross validation
                if candidate_ accuracy > current_accuracy: # swapping features improve accuracy
                        res = U (res, candidate_feature)\ current)
                        current_accuracy = candidate_accuracy
return res
```

8.RandomForest Search

<u>רקע:</u> RandomForest הוא אלגוריתם אשר תחילה ילמד N (היפר-פרמטר) עצי החלטה ID3, כל אחד עם תת קבוצה שונה של דוגמאות מקבוצת האימון.
<u>בחירת דוגמאות לכל עץ מתוך N העצים שנלמד:</u> גודל קבוצת האימון יסומן ב-n. עבור כל עץ מ-N העצים נגריל (באופן רנדומלי) $p \cdot n$ דוגמאות מקבוצת האימון, כאשר p הוא פרמטר לאלגוריתם וערכו 0.3.
<u>בחירת K תכונות כערך אלגוריתם בחירת התכונות:</u> עבור כל עץ, נחשב לכל תכונה את ממוצע הדוגאמות המשוקלל שעבורו התכונה הזו השתמשנו בה לפצל בעץ (הסבר בהמשך), ואז נחזיר את K התכונות הטובות ביותר לפי משקל זה.

הרעיון באלגוריתם זה להשתמש בתכונות שעוזרות לנו הכי הרבה ללמד את עצי החלטה (התכונות שהעצים משתמשים בהם הכי הרבה כדי להפריד) בנינו יער החלטה ולכל תכונה נתנו משקל כאשר המשקל הוא סכום הפעמים שתכונה זו שמשה להפריד בכל אחת מהעצים ,בכל פעם שתכונה מפרידה בצומת מסוים אנחנו נוסיף למשקל שלה מספר הדוגמאות בצומת הנוכחי חלקי מספר הדוגמאות בשורש העץ שבו הופיעה:

Build 10 decision trees, each tree trains on random indices with 30% of the total data
While building the decision trees,

```
feature_weight_map = [feature: 0 for all feature in dataset.features]
let # n_tree number of samples in root (= dataset size * 0.3)
for each tree_k in forest:
        for each node in tree:
                current_feature = node.feature
                feature_weight_map[current_feature] += #node.samples /n_tree

sort features by their separation count (based on feature_weight_map)
res = top k features
return res
```

מתודולוגיה ניסויית

הורדנו 10 datasets מתוך האתר UCI datasets, בכל ה datasets התכונות הן רציפות והמטרה היא בינארית.

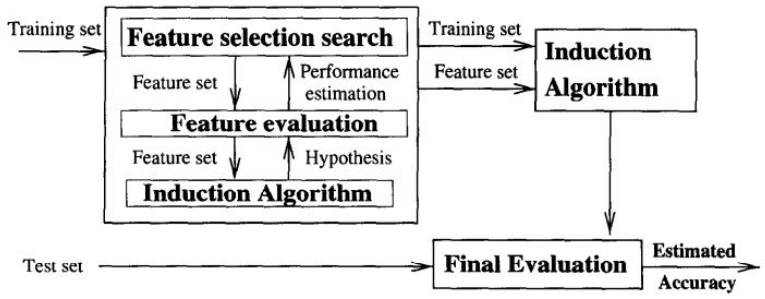
כדי לבדוק את התקינות של האלגוריתמים יצרנו שלושה datasets מלאכותיים, הוספנו תכונות רנדומליות ל datasets אמיתיים כדי לבדוק את היכולת של האלגוריתמים השונים לסנן את התכונות הרועשות אלו.

לכל dataset נתנו לכל אלגוריתם לבחור את התכונות הרלוונטיות עבור ה dataset הנוכחי, כל אלגוריתם הרצנו עשרה פעמים על כל dataset כל פעם אנחנו נבקש מהאלגוריתם לבחור אחוז מסוים מהתכונות אנחנו נתחיל עם קבוצת כל התכונות ואחר כך 90% עד 10% מהתכונות.

בכל פעם אנחנו נותנים לאלגוריתם לבחור קבוצת תכונות מתוך אלה שכבר חישבנו, כלומר שרוצים לבחור 80% מהתכונות אנחנו נעביר לאלגוריתם את 90% התכונות הטובות שחישבנו מקודם.

כל dataset פיצלנו לחמישה פולדים, כדי להשתמש בשיטת CV שהסברנו מקודם: בכל הרצה ארבעה פולדים ישמשו כקבוצת אימון והפולד החמישי ישמש כקבוצת מבחן. עבור כל אלגוריתם ועבור כל אחוז של תכונות הדיוק הוא הממוצע של חמשת הדיוקים על 5 הפולדים השונים.

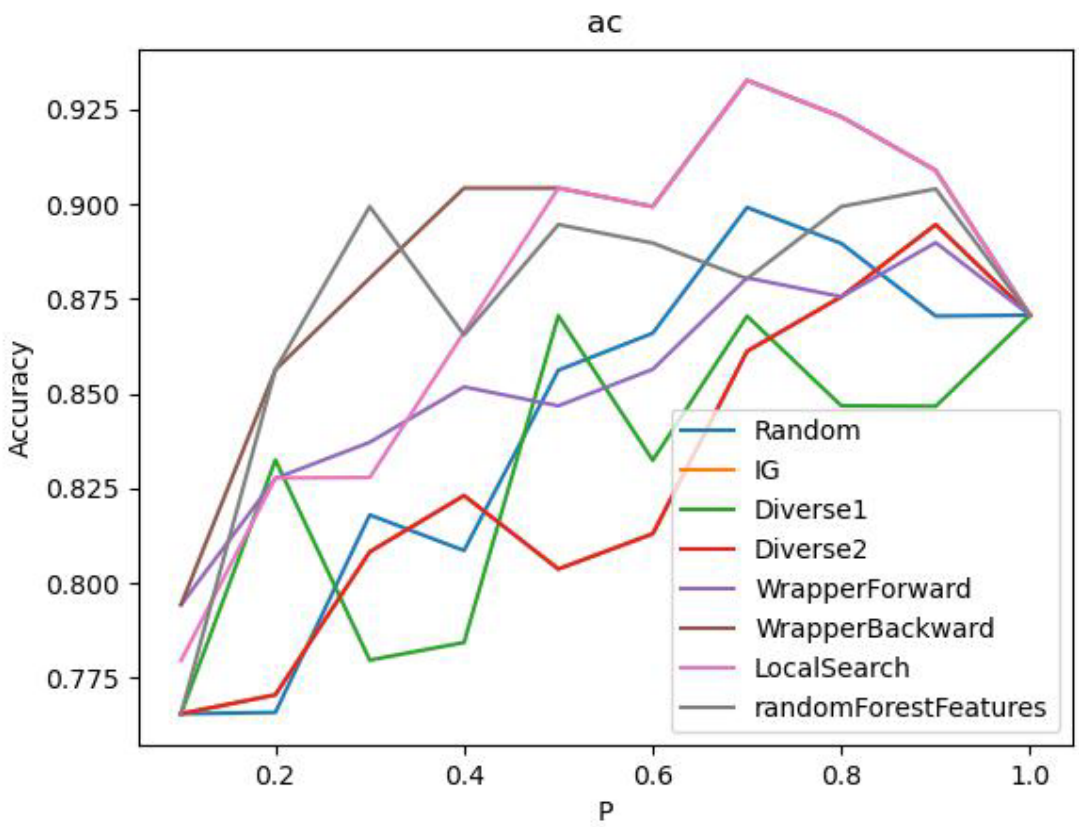
השתמשנו במסווג KNN של sklearn בכל הניסויים מס' השכנים היה 5 והשתמשנו במרחק אוקלידי.



ניסויים ותוצאות

ניסוי 1:

Dataset	Num of samples	Num of features
Accent recognition	210	12

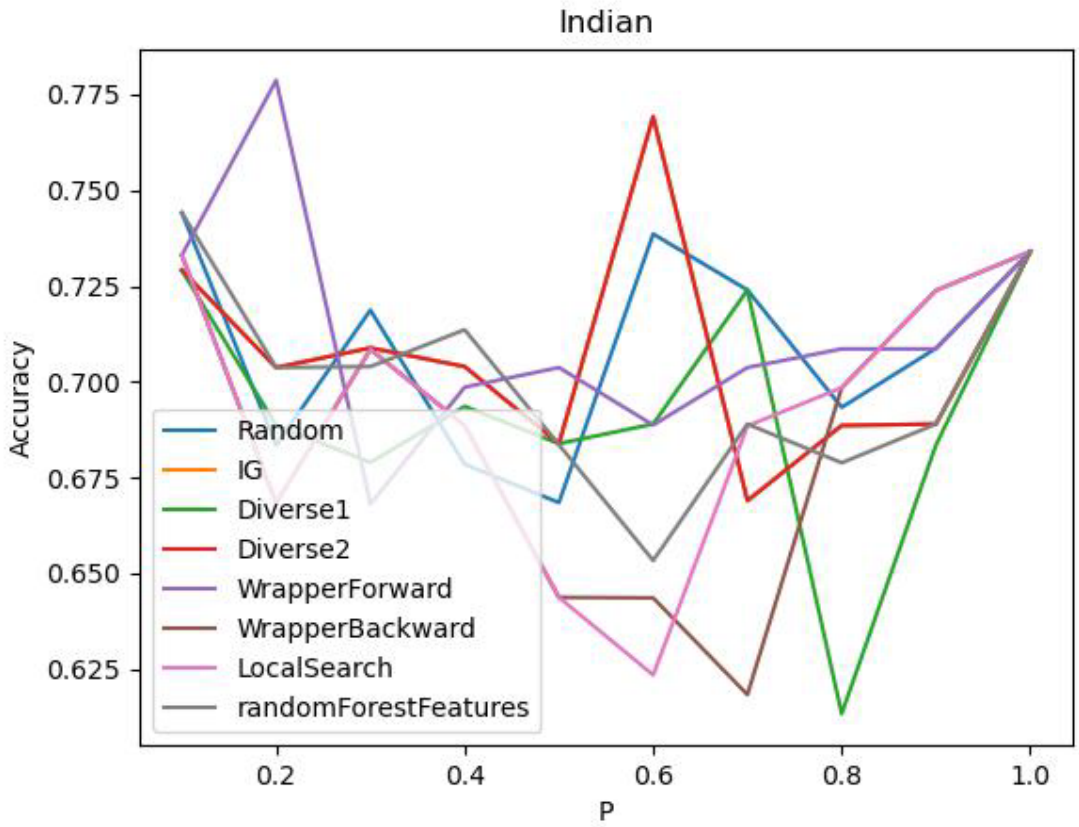


	1	0.9	0.8
random	0.87	0.87	0.89
max_ig	0.87	0.89	0.88
diverse1	0.87	0.85	0.85
diverse2	0.87	0.89	0.88
wrapperforward	0.87	0.89	0.88
wrapperbackward	0.87	0.91	0.92
localsearch	0.87	0.91	0.92
randomforest	0.87	0.9	0.9

כאן אפשר לראות כשאר בחרנו 90% מהתכונות האלגוריתמים localsearch ו wraperbackward הצליחו להשיג שיפור 4% בדיוק של המסווג ושבחרנו 80% הדיוק עלה ב 5% וזה ביחס לדיוק של המסווג אם היינו בוחרים בכל התכונות

ניסוי 2:

Dataset	Num of samples	Num of features
Indian	200	10

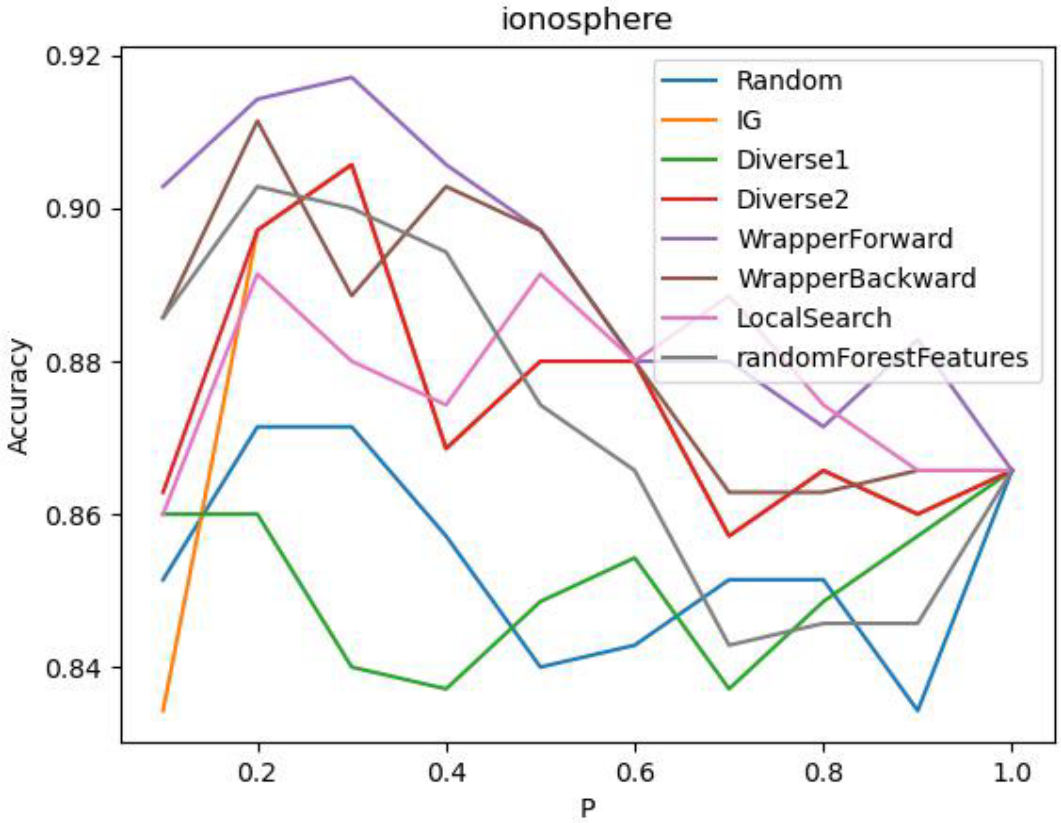


	1	0.6	0.2
random	0.73	0.74	0.68
max_ig	0.73	0.77	0.7
diverse1	0.73	0.69	0.69
diverse2	0.73	0.77	0.7
wrapperforward	0.73	0.69	0.78
wrapperbackward	0.73	0.64	0.67
localsearch	0.73	0.62	0.67
randomforest	0.73	0.65	0.7

בשאר השתמשנו בכל התכונות הדיוק של המסווג היה 73% וכאשר בחרנו ב 0.6 מהתכונות אלגוריתם diverse 2 ו max_ig הצליחו לשפר ב 5% וכשאר השתמשנו ב 0.2 מהתכונות האלגוריתם wrapperforward שיפר את הדיוק ב 5% גם כן

ניסוי 3:

Dataset	Num of samples	Num of features
ionosphere	351	23

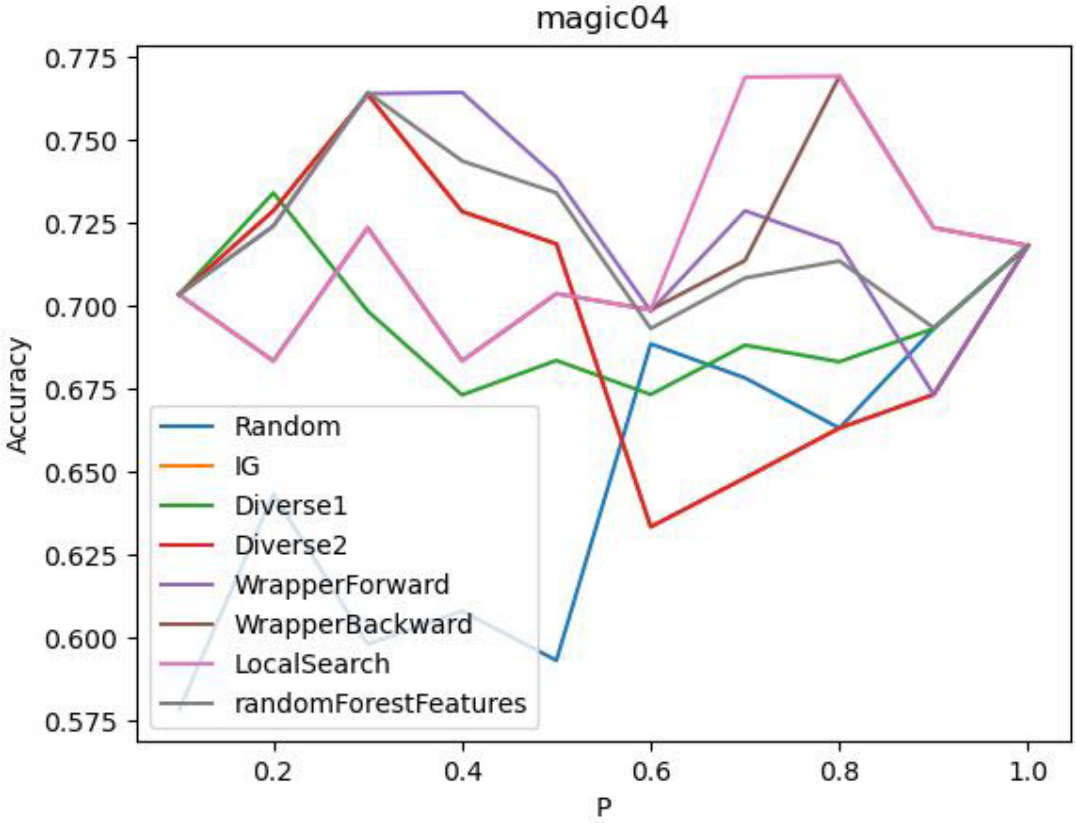


	1	0.4	0.3
random	0.87	0.86	0.87
max_ig	0.87	0.87	0.91
diverse1	0.87	0.84	0.84
diverse2	0.87	0.87	0.91
wrapperforward	0.87	0.91	0.92
wrapperbackward	0.87	0.9	0.89
localsearch	0.87	0.87	0.88
randomforest	0.87	0.89	0.9

אפשר לראות שהאלגוריתמים diverse2 ו wrapperbackward הצליחו להשיג שיפר ב 5% ו 4% בדיוק של המסווג

ניסוי 4:

Dataset	Num of samples	Num of features
magic	200	10

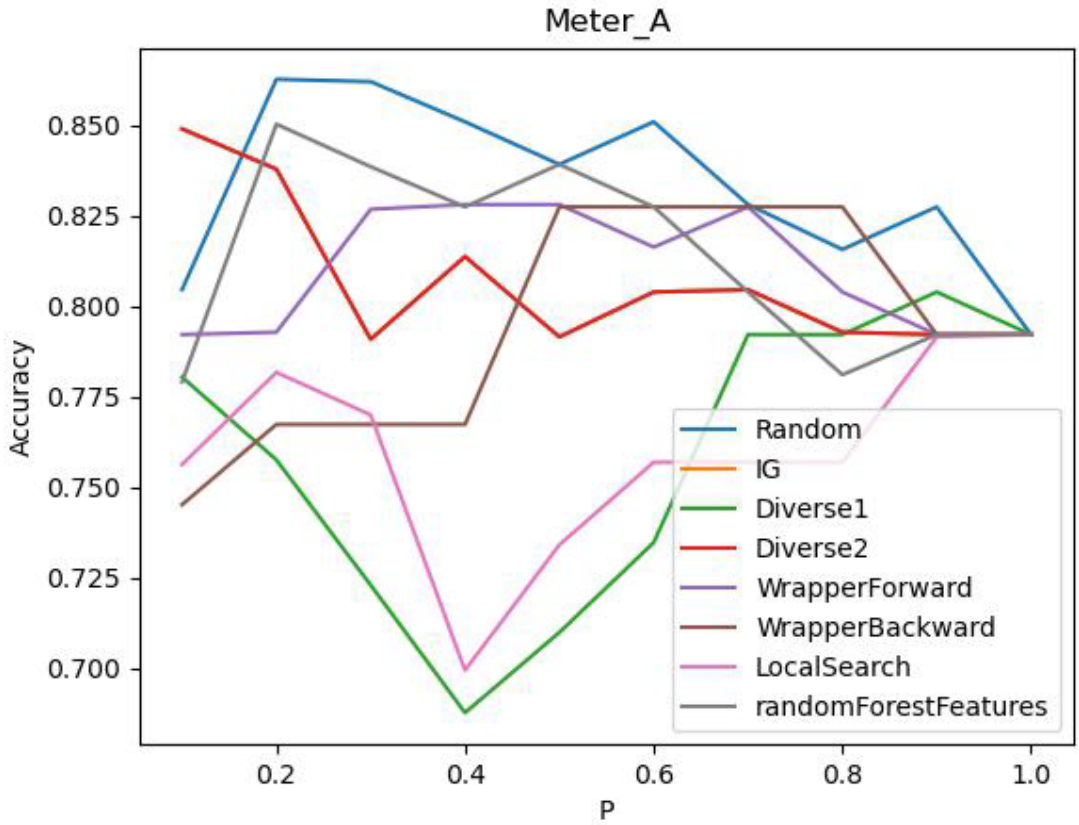


	1	0.8	0.7
random	0.72	0.66	0.68
max_ig	0.72	0.66	0.65
diverse1	0.72	0.68	0.69
diverse2	0.72	0.66	0.65
wrapperforward	0.72	0.72	0.73
wrapperbackward	0.72	0.77	0.71
localsearch	0.72	0.77	0.77
randomforest	0.72	0.71	0.71

כאן אפשר לראות שהאלגוריתמים ב
local search ו wrapperbackward
השיגו שיפור של 5% בדיוק של המסווג
אנחנו שמים לב שבכל פעם אנחנו
משיגים שיפור בממוצע של 4%

ניסוי 5:

Dataset	Num of samples	Num of features
Meter_A	87	36

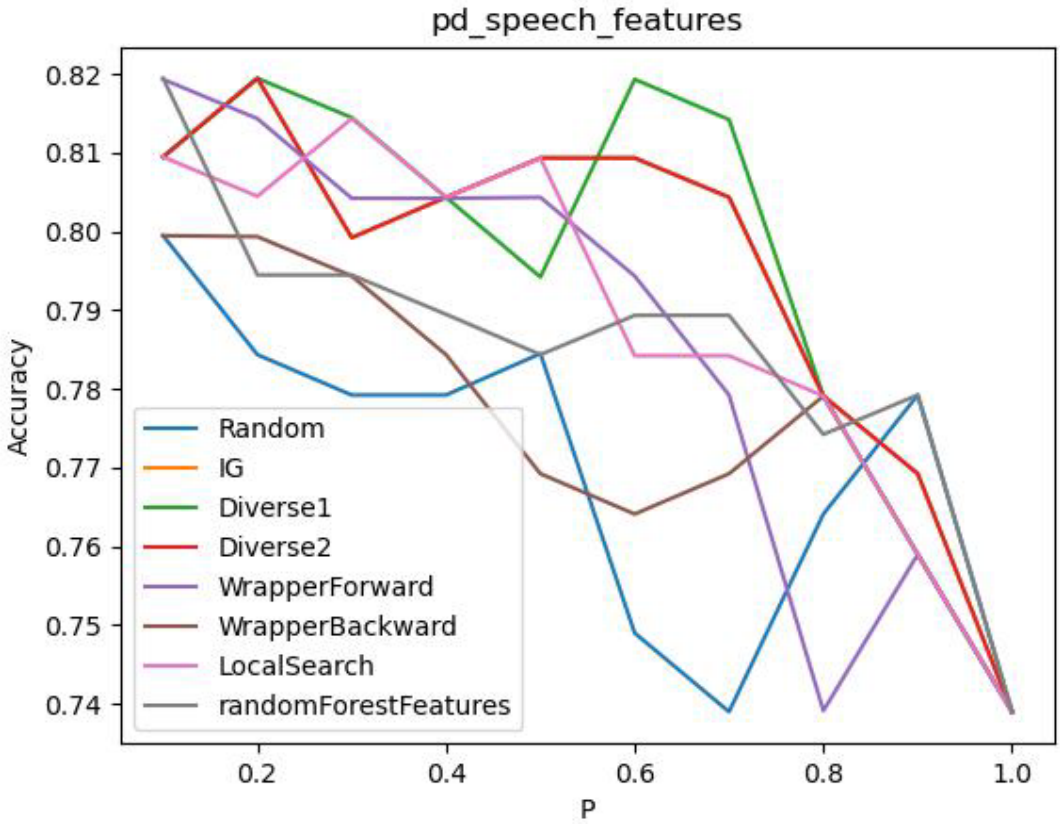


	1	0.8	0.7
random	0.79	0.82	0.83
max_ig	0.79	0.79	0.8
diverse1	0.79	0.79	0.79
diverse2	0.79	0.79	0.8
wrapperforward	0.79	0.8	0.83
wrapperbackward	0.79	0.83	0.83
localsearch	0.79	0.76	0.76
randomforest	0.79	0.78	0.8

ב dataset זה אנו רואים שה wrappers הצליחו להשיג שיפר של 5% בדיוק של המסווג כאשר בוחרים ב 0.8 ו 0.7 מהתכונות

ניסוי 6:

Dataset	Num of samples	Num of features
Pd speech	200	17

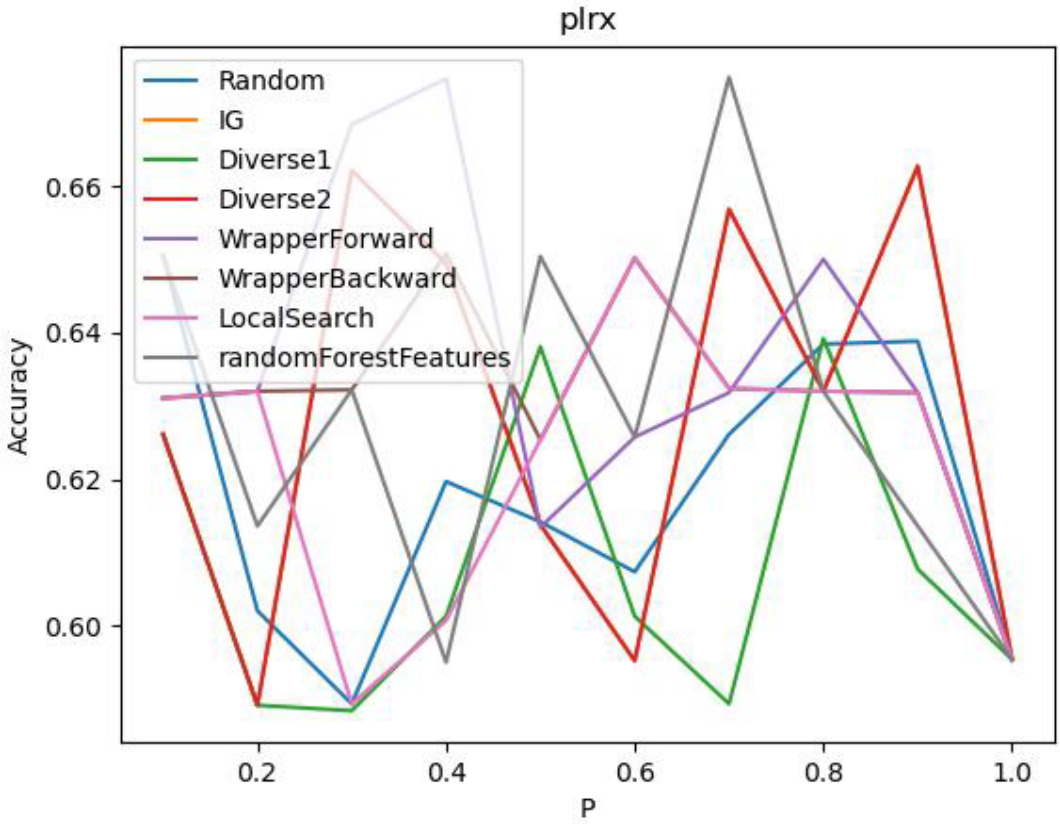


	1	0.2	0.1
random	0.74	0.78	0.8
max_ig	0.74	0.82	0.81
diverse1	0.74	0.82	0.81
diverse2	0.74	0.82	0.81
wrapperforward	0.74	0.81	0.82
wrapperbackward	0.74	0.8	0.8
localsearch	0.74	0.8	0.81
randomforest	0.74	0.79	0.82

ב dataset זה אנחנו רואים שכל האלגוריתמים באופן גורף מסננים את התכונות הרועשות בהדרגה זה מעיד על כל שיש מס' גדול של תכונות לא רלוונטיות האלגוריתמים local search ו randomforest הצליחו להשיג שיפור של 8% ו 9%

ניסוי 7:

Dataset	Num of samples	Num of features
Parkinson	164	12

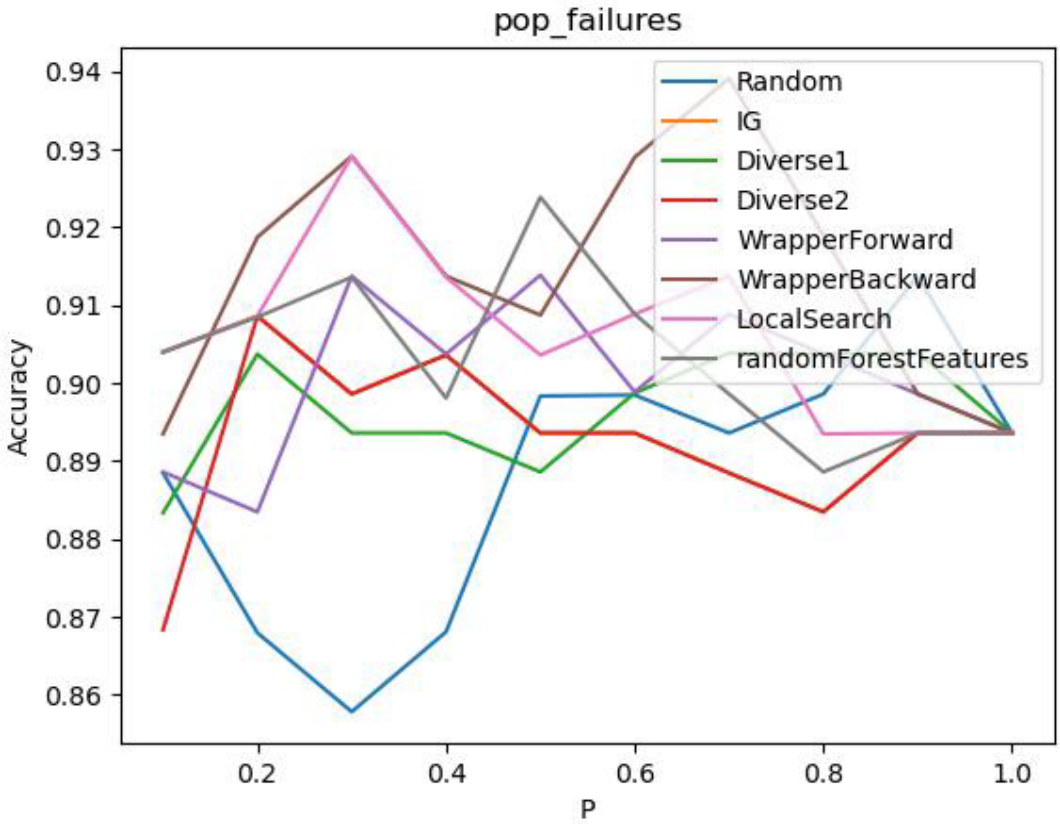


	1	0.7	0.4
random	0.6	0.63	0.62
max_ig	0.6	0.66	0.65
diverse1	0.6	0.59	0.6
diverse2	0.6	0.66	0.65
wrapperforward	0.6	0.63	0.67
wrapperbackward	0.6	0.63	0.65
localsearch	0.6	0.63	0.6
randomforest	0.6	0.67	0.6

אפשר לראות כאן שחלק גדול
מהאלגוריתמים הצליחו להשיג שיפר של
5% ו 7% כאשר השתמשנו ב 0.7 וב 0.4
מהתכונות

ניסוי 8:

Dataset	Num of samples	Num of features
Pop_failures	199	20

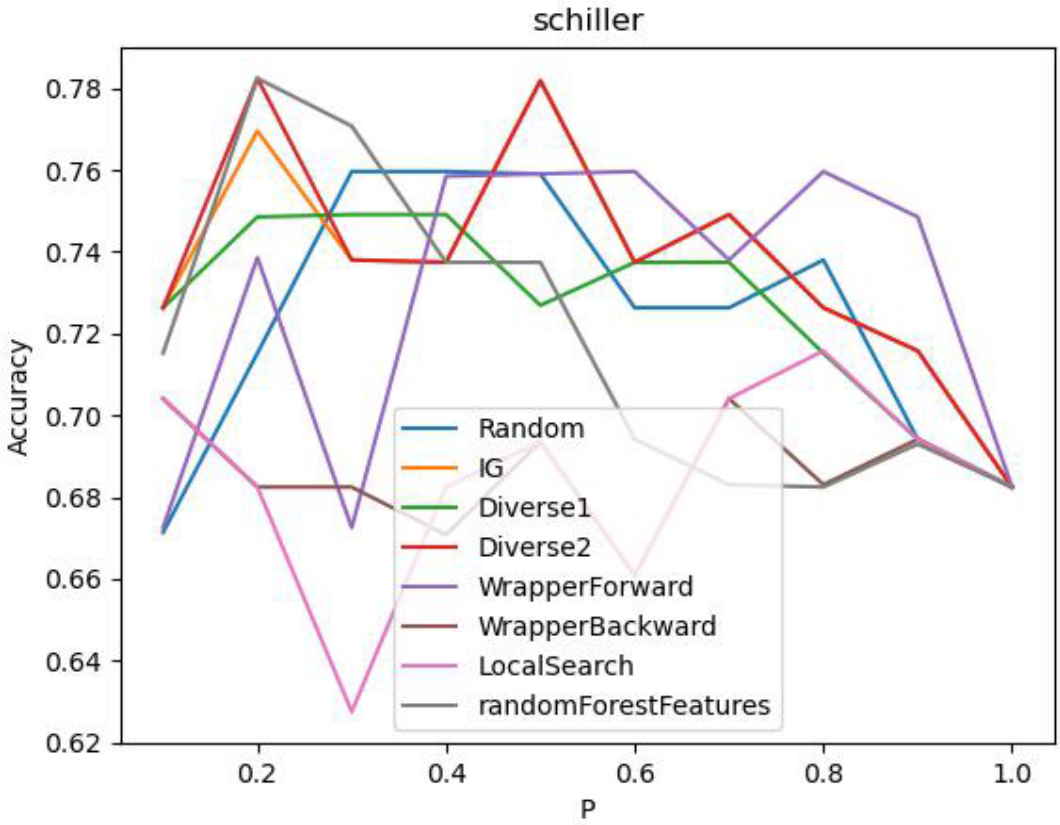


	1	0.7	0.3
random	0.89	0.89	0.86
max_ig	0.89	0.89	0.9
diverse1	0.89	0.9	0.89
diverse2	0.89	0.89	0.9
wrapperforward	0.89	0.91	0.91
wrapperbackward	0.89	0.94	0.93
localsearch	0.89	0.91	0.93
randomforest	0.89	0.9	0.91

כאן אפשר לראות ש wrapperbackward
וה localsearch הצליחו להשיג שיפור של
5% ו 4% לעומת המסווג שהשתמש בכל
התכונות כשאר בחרנו 0.7 ו 0.3 מהתכונות

ניסוי 9:

Dataset	Num of samples	Num of features
schiller	92	14

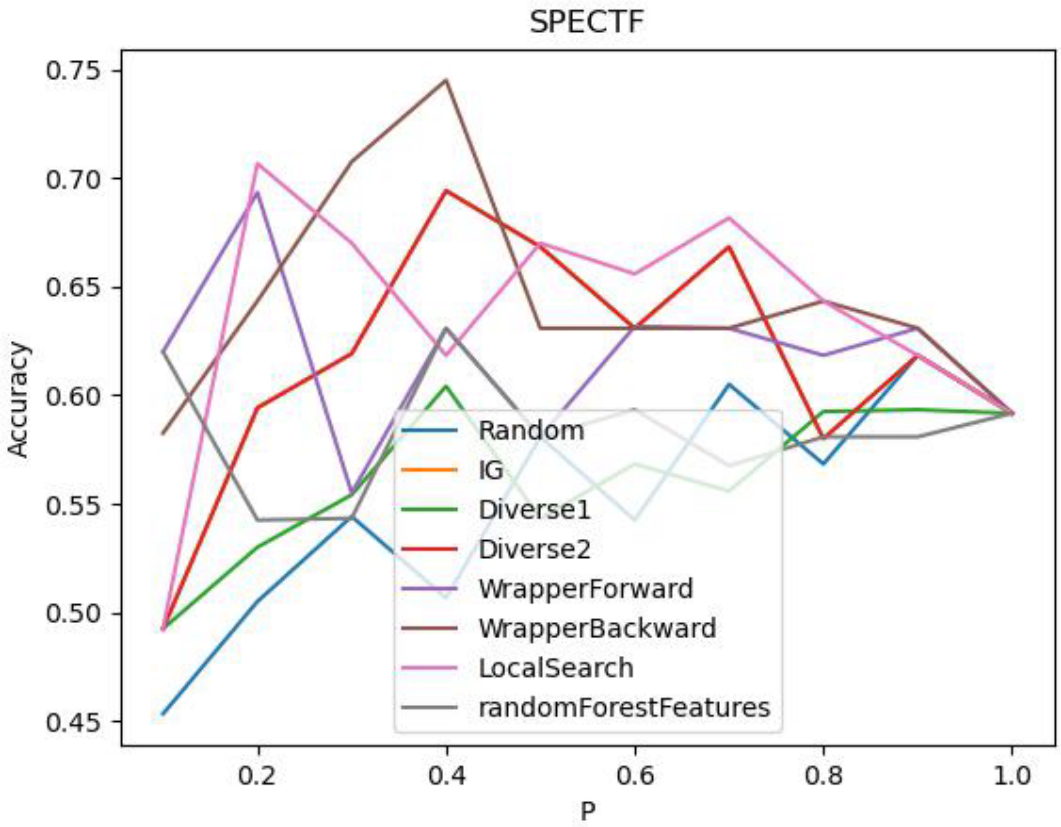


	1	0.9	0.5
random	0.68	0.69	0.76
max_ig	0.68	0.72	0.78
diverse1	0.68	0.69	0.73
diverse2	0.68	0.72	0.78
wrapperforward	0.68	0.75	0.76
wrapperbackward	0.68	0.69	0.69
localsearch	0.68	0.69	0.69
randomforest	0.68	0.69	0.74

אנחנו יכולים לראות שהאלגוריתם max_ig וה wrappers הצליחו להשיג שיפור של 10% כמעט כאשר בחרנו ב 0.5 מהתכונות ושיפור של 8% כאשר בחרנו ב 0.9 מהתכונות

ניסוי 10:

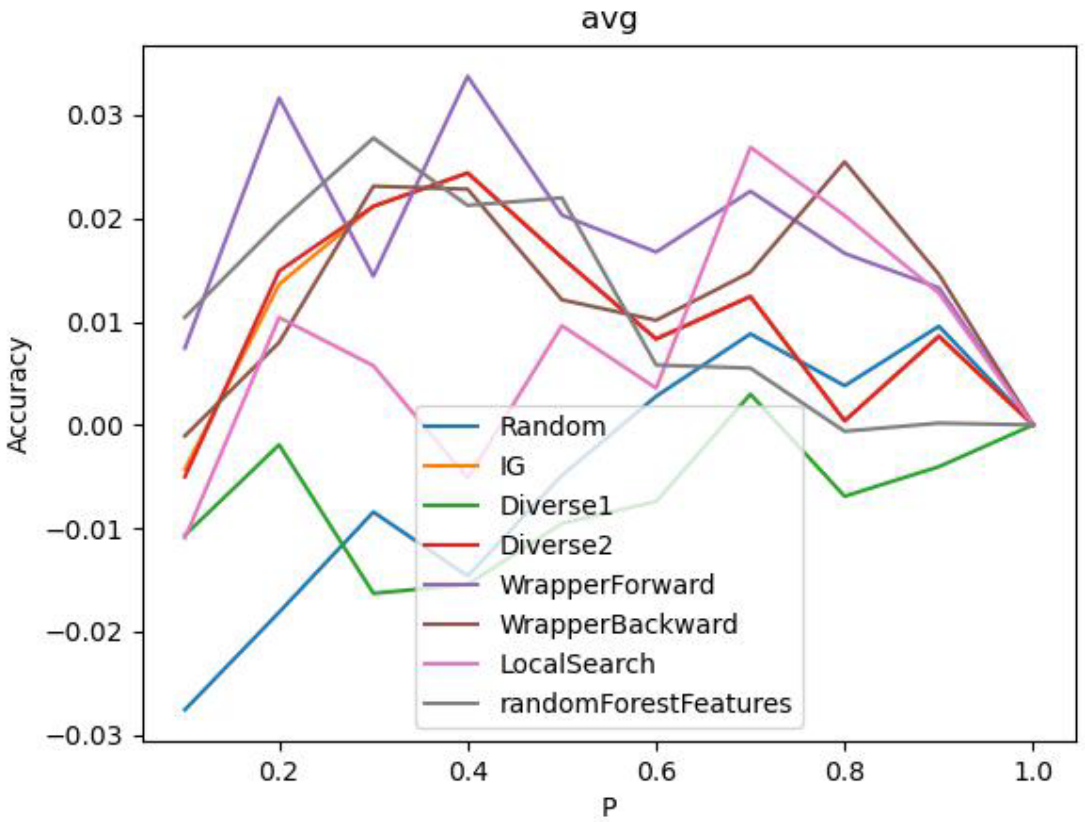
Dataset	Num of samples	Num of features
SPECTF	80	15



	1	0.9	0.4
random	0.59	0.62	0.51
max_ig	0.59	0.62	0.69
diverse1	0.59	0.59	0.6
diverse2	0.59	0.62	0.69
wrapperforward	0.59	0.63	0.63
wrapperbackward	0.59	0.63	0.74
localsearch	0.59	0.62	0.62
randomforest	0.59	0.58	0.63

כאן אפשר לראות שהאלגוריתמים max_ig ו diverse2 ו wrapperbackward מצליחים להשיג שיפור ביותר מ 10% כאשר בוחרים ב 0.4 מהתכונות

גרף מסכם

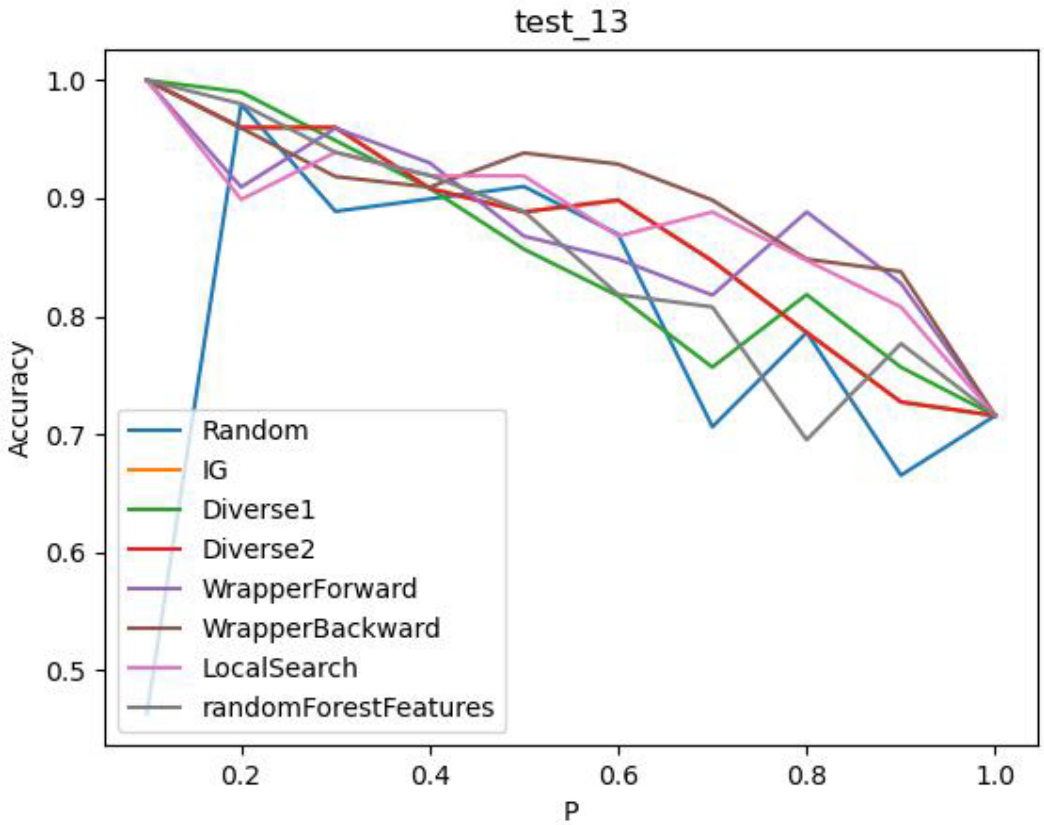


כאן אנחנו מתארים גרף שמראה את ממוצע השיפורים של כל אלגוריתם על כל datasets , בציר האופקי יש לנו אחוז התכונות שהשתמשנו בהם ובציר האנכי הדיוק.

אפשר לראות שהאלגוריתמים local search ושני ה wrappers כאשר מסננים 20% מהתכונות אפשר להשיג שיפור של כמעט 2% ביחס למסווג שהתאמן על כל התכונות.

גם האלגוריתמים wrapperforward ו diverse2 מצליחים להשיג שיפור 2-3% כאשר בוחרים ב 40% מהתכונות.

ניסויים נוספים שביצענו:



כאן עשינו ניסוי לבדוק את התקינות של האלגוריתמים שלנו בנינו dataset מלאכותי , הוא מכיל 30 תכונות רועשות שנוצרו רנדומלית ותכונה אחת רלוונטית, כל התכונות מכילות הערך 1 או 0, והסיווג הוא M או B, אם הדוגמה מסווגת להיות M אז התכונה הרלוונטית היחידה שלנו מכילה את הערך 0 , אחרת מכילה את הערך B.

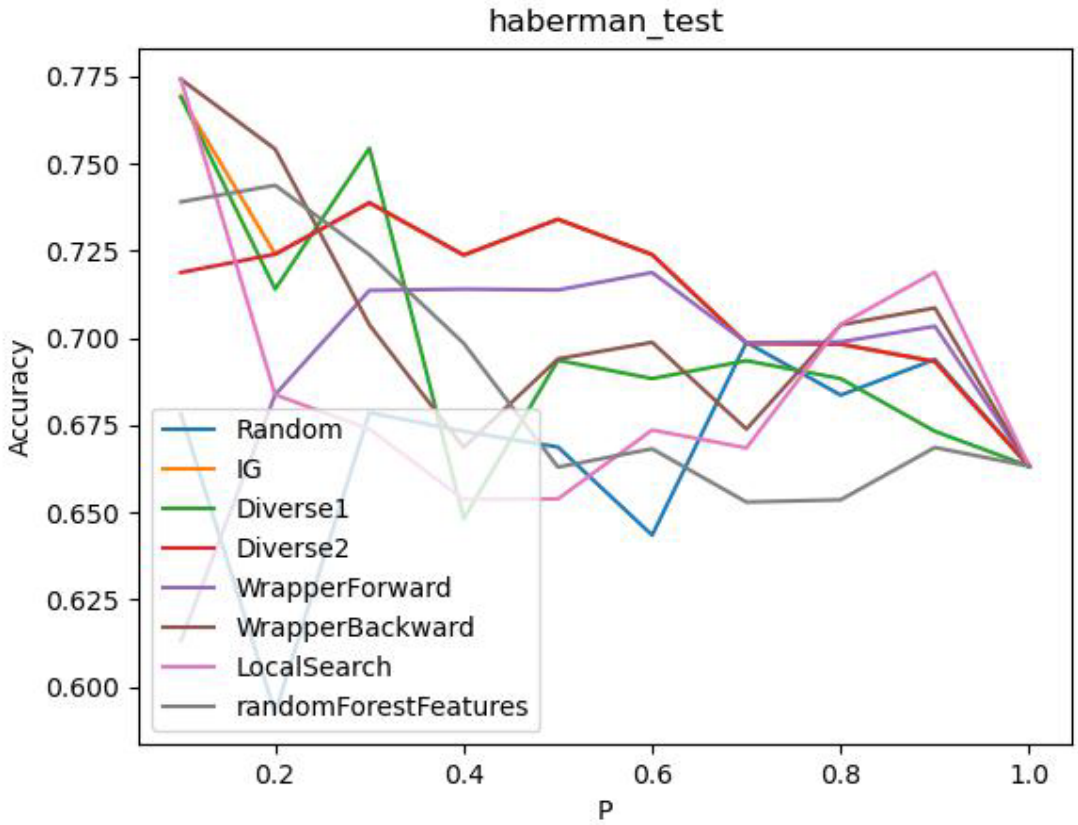
אפשר לראות שכל האלגוריתמים מצליחים בסופו של דבר לסנן את התכונות הרועשות ולהישאר עם התכונה הרלוונטית , מלבד האלגוריתם Random כי הוא פועל באופן אקראי.

M	0
B	1
M	0
B	1
M	0
B	1
M	0
B	1
M	0
B	1
M	0
B	1
M	0

כאן מציגים חלק מה dataset שיצרנו, ומראים רק את התכונה הרלוונטית , שאר התכונות יצרנו אותם באקראי

ניסוי נוסף:

כאן



השתמשנו ב dataset מקורי שמכיל 3 תכונות ו 100 דוגמאות והוספנו לו 22 תכונות רועשות כדי לבדוק את היכולת של האלגוריתמים שלנו לסנן תכונות אלה, כאשר הוספנו תכונות רועשות הדיוק של המסווג ירד ב 10% אחוזים מספר הדוגמאות והתכונות היה קטן ולכן הוספנו הרבה תכונות רועשות כדי לפגוע בדיוק של המסווג , אפשר לראות שרוב האלגוריתמים הצליחו לסנן את התכונות הרועשות ולהישאר עם 10% מהתכונות שהן הטובות ביותר והדיוק עלה ל ב 10%.

סיכום

בפרויקט זה אנחנו בחנו כמה אלגוריתמים לסינון רעשים וראינו שיש מקרים שבהם כן יש תכונות שמטעות את המסווג , תכונות אלה יכולות להוריד את הדיוק את של המסווג ובנוסף גם להאריך את זמן האימון.

אנחנו בחנו את האלגוריתמים שלנו מול datasets יחסית קטנים מבחינת מספר הדוגמאות ומספר התכונות , יהיה מעניין מאוד לבדוק אותם מול datasets עם מספר תכונות גדול יחסית.

המסווג שהשתמשנו בו היה KNN , מסווג זה מאוד רגיש לתכונות לא רלוונטיות וזה היה יותר קל עבורנו לראות את היכולת של תכונות רועשות להשפיע על תהליך הלמידה, למשל עצי החלטה לא כל כך רגישים למספר קטן של תכונות מטעות.

אלגוריתם החיפוש שהשתמשנו בו היה first choice – hill climbing, אלגוריתם זה מאוד יעיל שמקדם הסיעוף גדול מאוד אבל הוא לא האלגוריתם הטוב ביותר מבין אלגוריתמי החיפוש , עבודה עתידית אפשרית היא למצוא שיטות איך להשתמש באלגוריתמי חיפוש אחרים תוך כדי התחשבות בזמן הריצה שלהם.

ביבליוגרפיה:

[1] “Introduction to Artificial Intelligence” course: https://webcourse.cs.technion.ac.il/236501

[2] datasets:

https://archive.ics.uci.edu/ml/datasets/Speaker+Accent+Recognition
https://archive.ics.uci.edu/ml/datasets/Speaker+Accent+Recognition
https://archive.ics.uci.edu/ml/datasets/ILPD+(Indian+Liver+Patient+Dataset)
https://archive.ics.uci.edu/ml/datasets/ionosphere
https://archive.ics.uci.edu/ml/datasets/magic+gamma+telescope
https://archive.ics.uci.edu/ml/datasets/Ultrasonic+flowmeter+diagnostics
https://archive.ics.uci.edu/ml/datasets/Parkinson+Dataset+with+replicated+acoustic+features+
https://archive.ics.uci.edu/ml/machine-learning-databases/00230/
https://archive.ics.uci.edu/ml/datasets/SPECTF+Heart
https://archive.ics.uci.edu/ml/datasets/Quality+Assessment+of+Digital+Colposcopies
https://archive.ics.uci.edu/ml/datasets/climate+model+simulation+crashes

[3] Spearman rank correlation coefficient:

https://mathworld.wolfram.com/SpearmanRankCorrelationCoefficient.html

[4] sklearn feature selection: https://scikit-learn.org/stable/modules/feature_selection.html