



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 או 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 הטריוויואלית 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

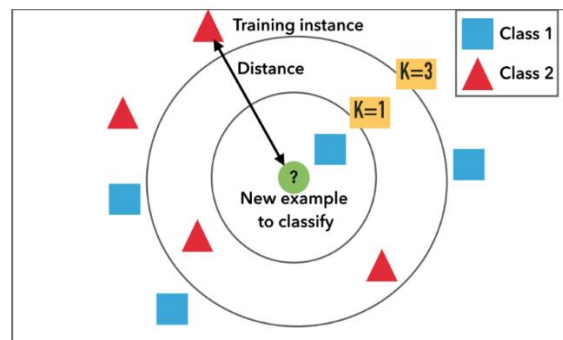
אנו ננסה לבחון את ההשפעה של בחירת אחוז מסוים מהתכונות על רמת הדיוק של המסווג, נשתמש בשמונה אלגוריתמים שונים על מנת לסנן חלק מהתכונות ולבדוק את הביצועים של כל אחד מהם.

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 ככל שיש לנו הרבה דוגמאות באימון.



נרמול dataset: ב ח-n k 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 וננרמל את התכונה לפי ערכים אלה.

Entropy

יהי X משתנה רנדומי עם ערכים x_1, \dots, x_n

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

האנטרופיה של X מוגדרת:

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.

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

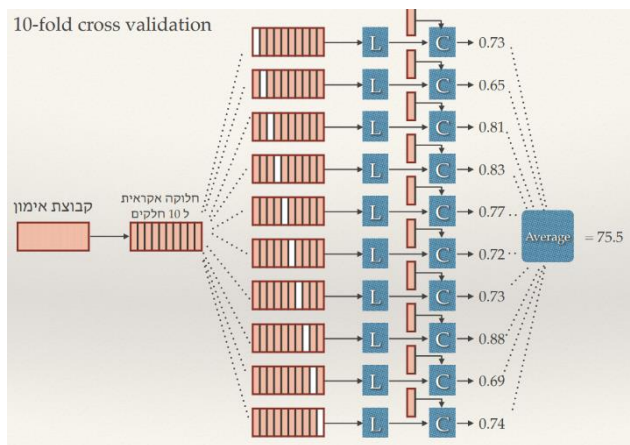
Cross-validation (CV): זו שיטה כדי לבדוק את רמת הדיוק של מסווג מסוים, אנחנו נחלק ה data שלנו ל K חלקים, K פרמטר שאפשר לבחון, בהינתן $data - set$ מסוים, נסמן את קבוצת כל הדוגמאות שבו ע"י Examples. כמו כן, עבור כל $data - set$ נחלק את הדוגמאות שבו ל- m folds שיישאר **קבועים** לאורך כל הניסויים ונסמנם: $Folds = \{fold_1, fold_2, \dots, fold_m\}$. 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

$train_i = Examples \setminus fold_i$, כלומר קבוצת הדוגמאות שאינן שייכות ל- $fold_i$.

ב. חשב את אחוזי הדיוק ($accuracy$) של המסווג על קבוצת הבדיקה שתהא ה- i fold, 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 הקשר בין שני משתנים כשאר כל אחד מהם נמדד בסולם אחר, 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 $-1 \leq r_s \leq 1$, 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:

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

בפריקט שלנו השתמשנו בפונקציה `spearmanr` של `scipy.stats`.

אלגוריתמים

בכל האלגוריתמים השתמשנו במסווג KNN.

$\text{spearman_min_distance}(\text{feature}, \text{window})$ – 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.

$\text{MaxIG}(\text{window}, k)$ - מחזירה את k התכונות מתוך החלון שיש להן IG הגדול ביותר. הקלט של כל אלגוריתם k dataset 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 $\text{spearman correlation}$, את הפתרון אנחנו נאתחל עם התכונה שיש לה IG מקסימלי ובכל פעם אנחנו נוסיף לפתרון את התכונה הכי רחוקה מהתכונות שנמצאות בפתרון, הרעיון מאחורי 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 ובבחר 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. Local Search

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 הסיוע בבעיה שלנו יחסית גדול $k(n - k)$ 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 num_features times:
    flag=0
    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
        flag=1
    break
    if flag==1:
        break
return res
```

8.RandomForest Search

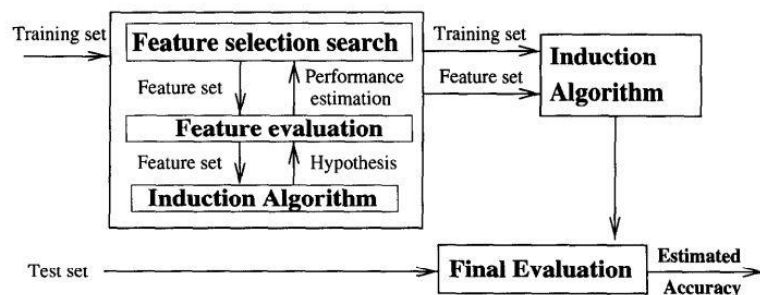
רקע: RandomForest הוא אלגוריתם אשר תחילה ילמד N (היפר-פרמטר) עצי החלטה ID3, כל אחד עם תת קבוצה שונה של דוגמאות מקבוצת האימון.
בחירת דוגמאות לכל עץ מתוך N העצים שנלמד: גודל קבוצת האימון יסומן ב- n . עבור כל עץ מ- N העצים נגריל (באופן רנדומלי) $n \cdot p$ דוגמאות מקבוצת האימון, כאשר p הוא פרמטר לאלגוריתם וערכו 0.3.
בחירת K תכונות כערך אלגוריתם בחירת התכונות: עבור כל עץ, נחשב לכל תכונה את ממוצע הדוגמאות 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 /  $n_{tree}$ 

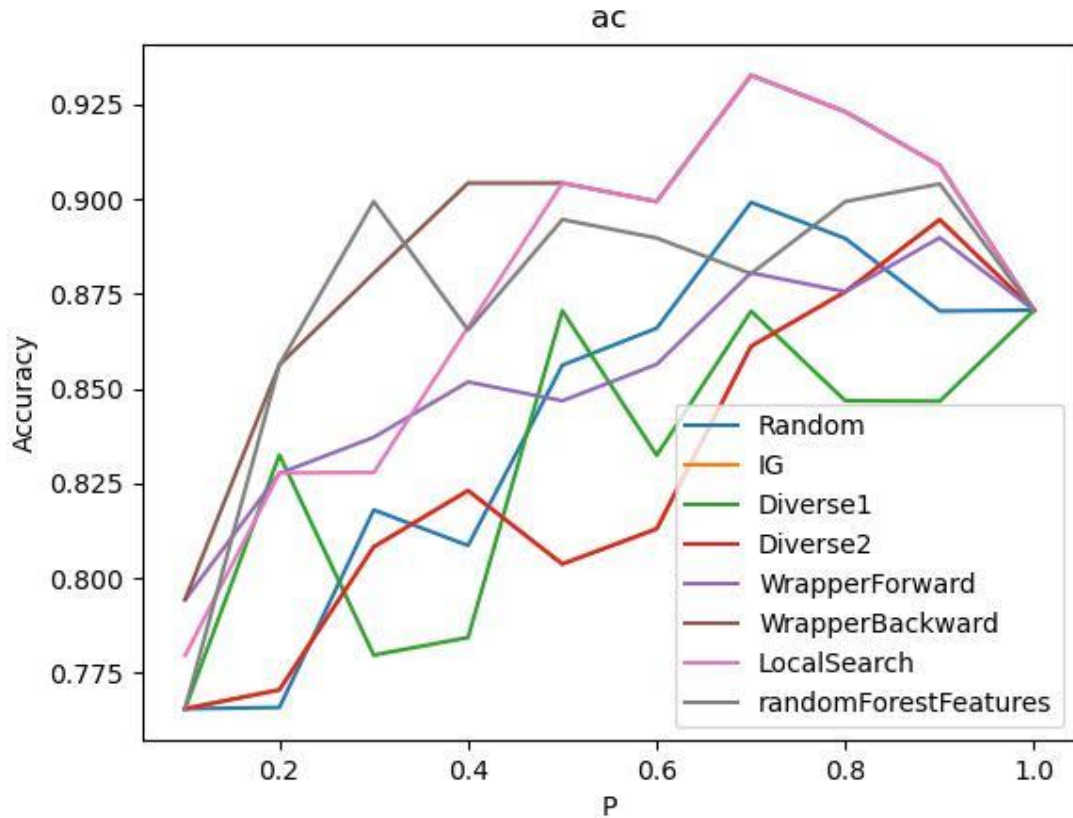
sort features by their separation count (based on feature_weight_map)
res = top k features
return res
```

ניסויים ותוצאות

ניסוי 1:

Dataset	Num of samples	Num of features
Accent recognition	210	12



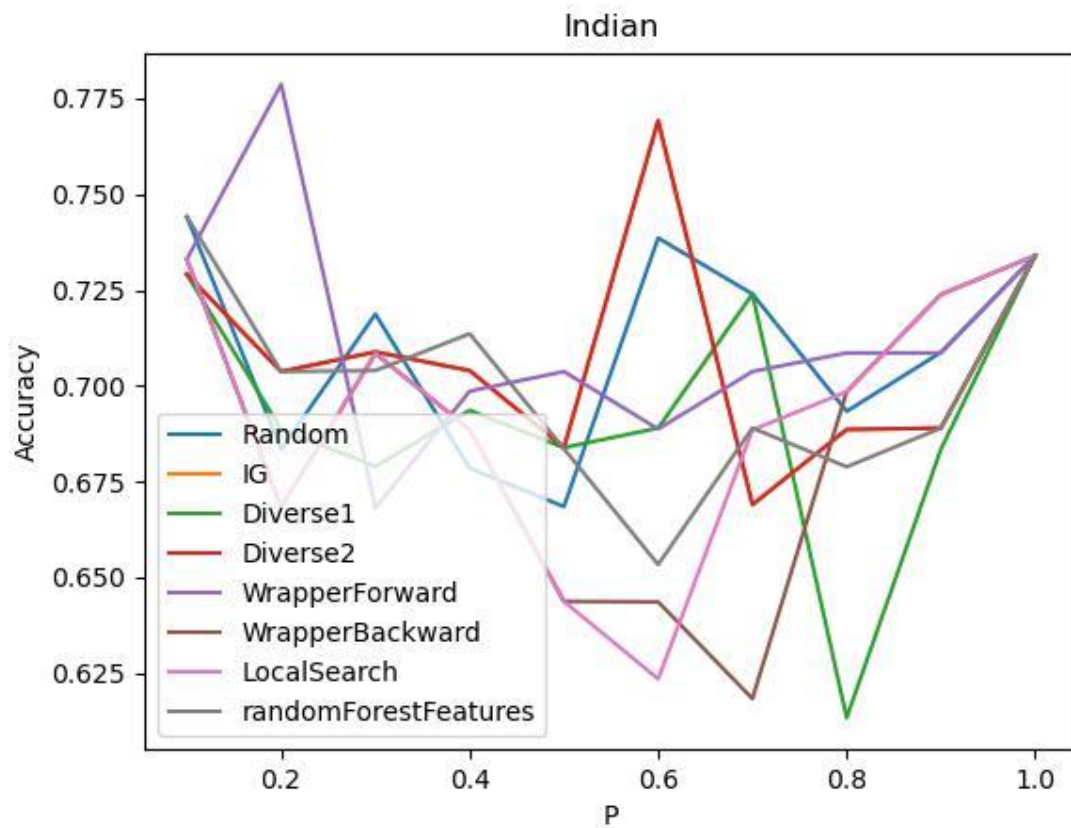
	0.9	0.8
Random	0.87	0.89
max ig	0.89	0.88
diverse1	0.85	0.85
diverse2	0.89	0.88
wrapperforward	0.89	0.88
wrapperbackward	0.91	0.92
local search	0.91	0.92
randomforest	0.9	0.9

Best accuracy:

	p	accuracy
Random	0.7	0.9
max ig	0.9	0.89
diverse1	0.7	0.87
diverse2	0.9	0.89
wrapperforward	0.9	0.89
wrapperbackward	0.7	0.93
local search	0.7	0.93
randomforest	0.9	0.9

ניסוי 2:

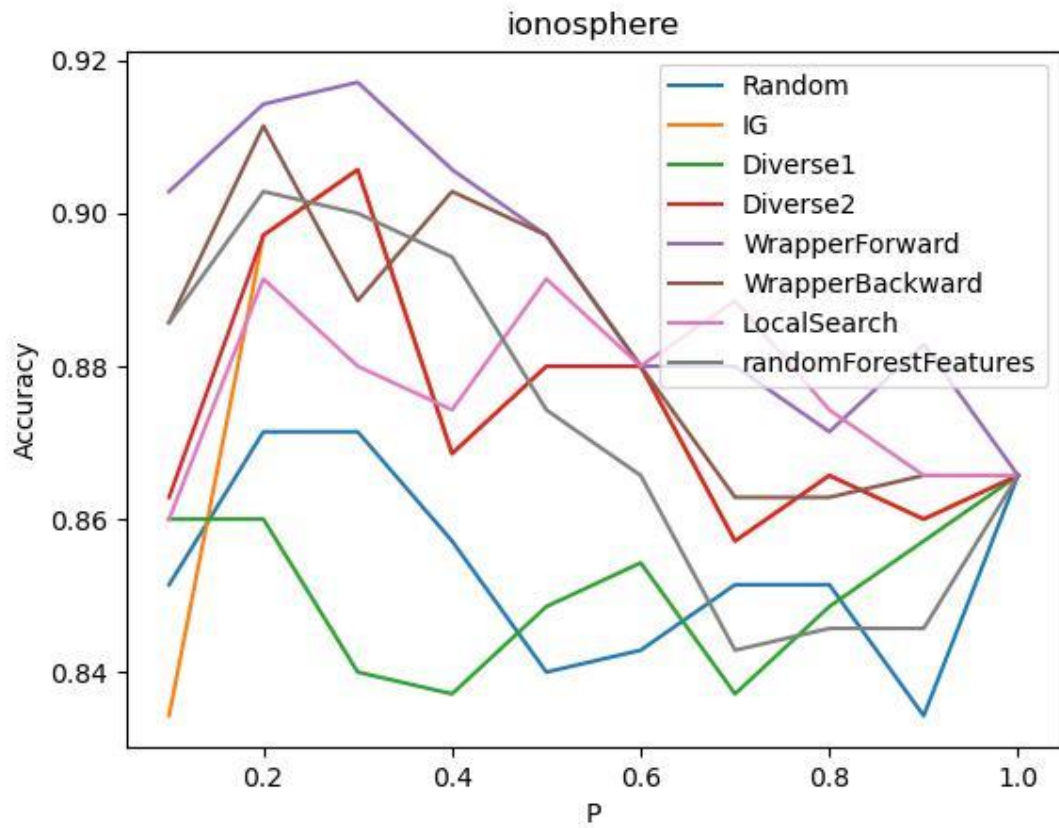
Dataset	Num of samples	Num of features
Indian	200	10



	0.9	0.8
Random	0.71	0.69
max ig	0.69	0.69
diverse1	0.68	0.61
diverse2	0.69	0.69
wrapperforward	0.71	0.71
wrapperbackward	0.72	0.7
local search	0.72	0.7
randomforest	0.69	0.68

Best accuracy:		
	p	accuracy
Random	0.6	0.74
max ig	0.6	0.77
diverse1	1	0.73
diverse2	0.6	0.77
wrapperforward	0.2	0.78
wrapperbackward	1	0.73
local search	1	0.73
randomforest	0.1	0.74

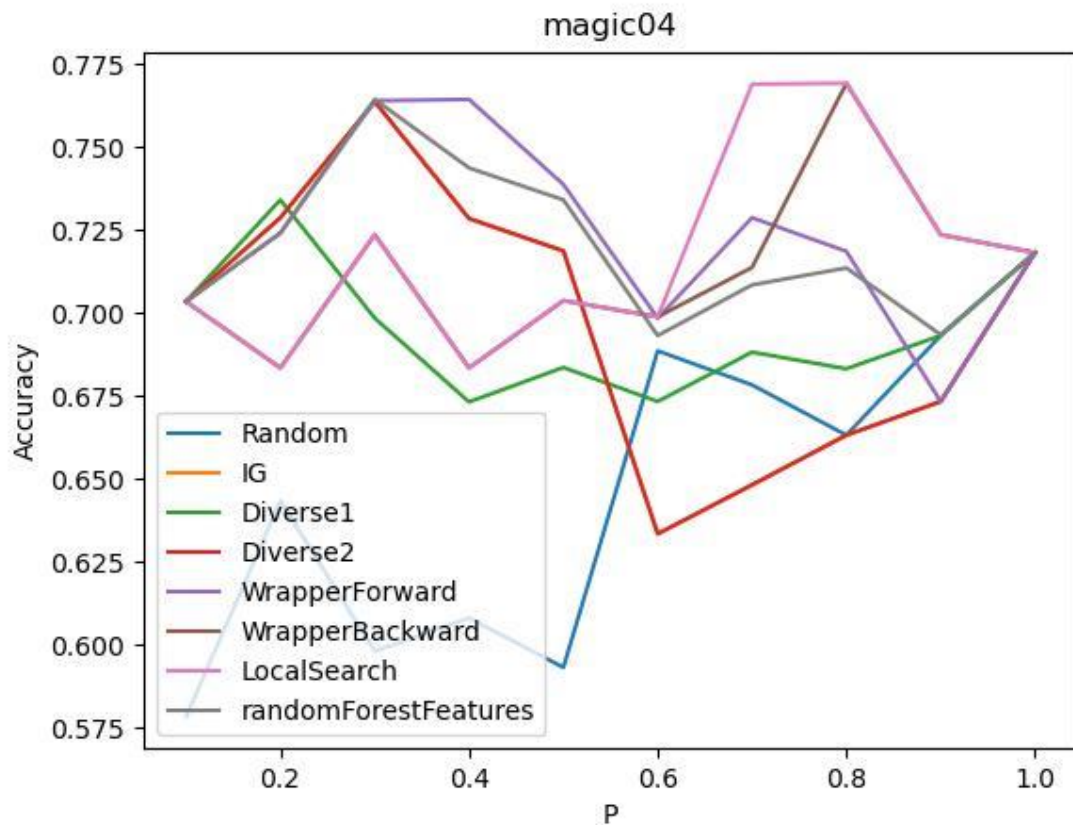
Dataset	Num of samples	Num of features
ionosphere	351	23



	0.9	0.8
Random	0.83	0.85
max ig	0.86	0.87
diverse1	0.86	0.85
diverse2	0.86	0.87
wrapperforward	0.88	0.87
wrapperbackward	0.87	0.86
local search	0.87	0.87
randomforest	0.85	0.85

Best accuracy:		
	p	accuracy
Random	1	0.87
max ig	0.3	0.91
diverse1	1	0.87
diverse2	0.3	0.91
wrapperforward	0.3	0.92
wrapperbackward	0.2	0.91
local search	0.7	0.89
randomforest	0.3	0.9

Dataset	Num of samples	Num of features
magic	200	10



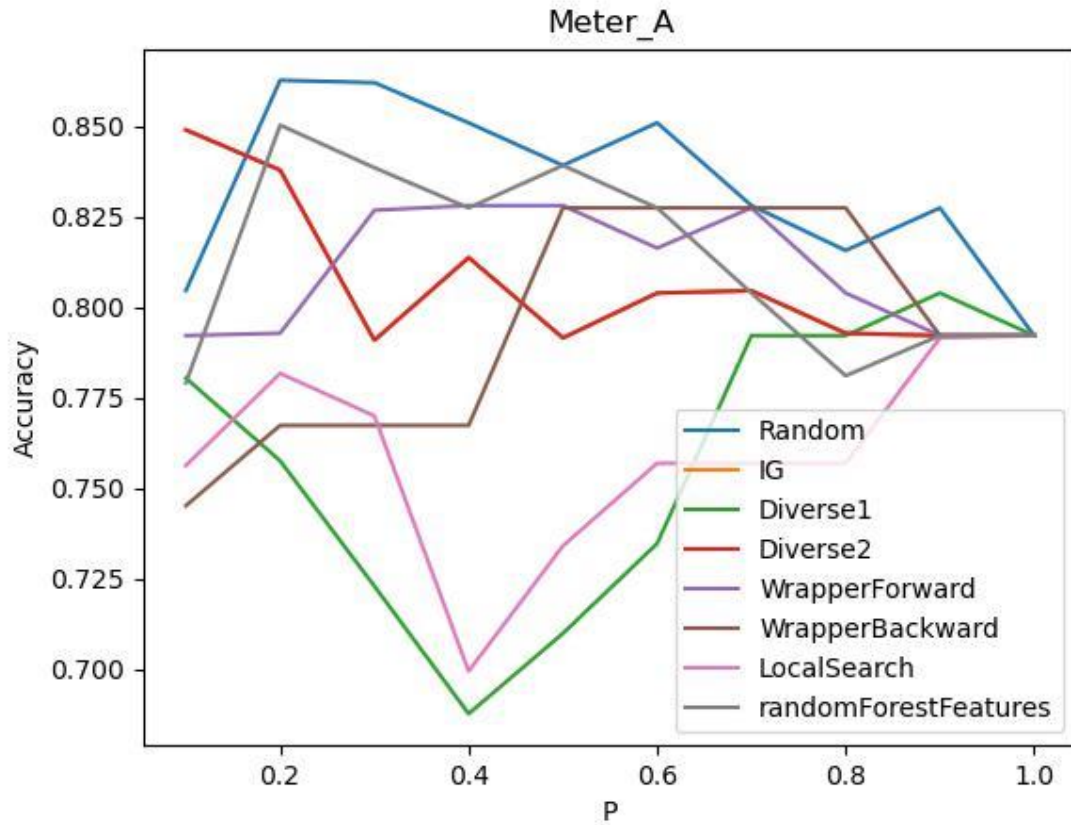
Best accuracy:

	0.9	0.8
Random	0.69	0.66
max ig	0.67	0.66
diverse1	0.69	0.68
diverse2	0.67	0.66
wrapperforward	0.67	0.72
wrapperbackward	0.72	0.77
local search	0.72	0.77
randomforest	0.69	0.71

	p	accuracy
Random	1	0.72
max ig	0.3	0.76
diverse1	0.2	0.73
diverse2	0.3	0.76
wrapperforward	0.4	0.76
wrapperbackward	0.8	0.77
local search	0.8	0.77
randomforest	0.4	0.74

ביסוי 5:

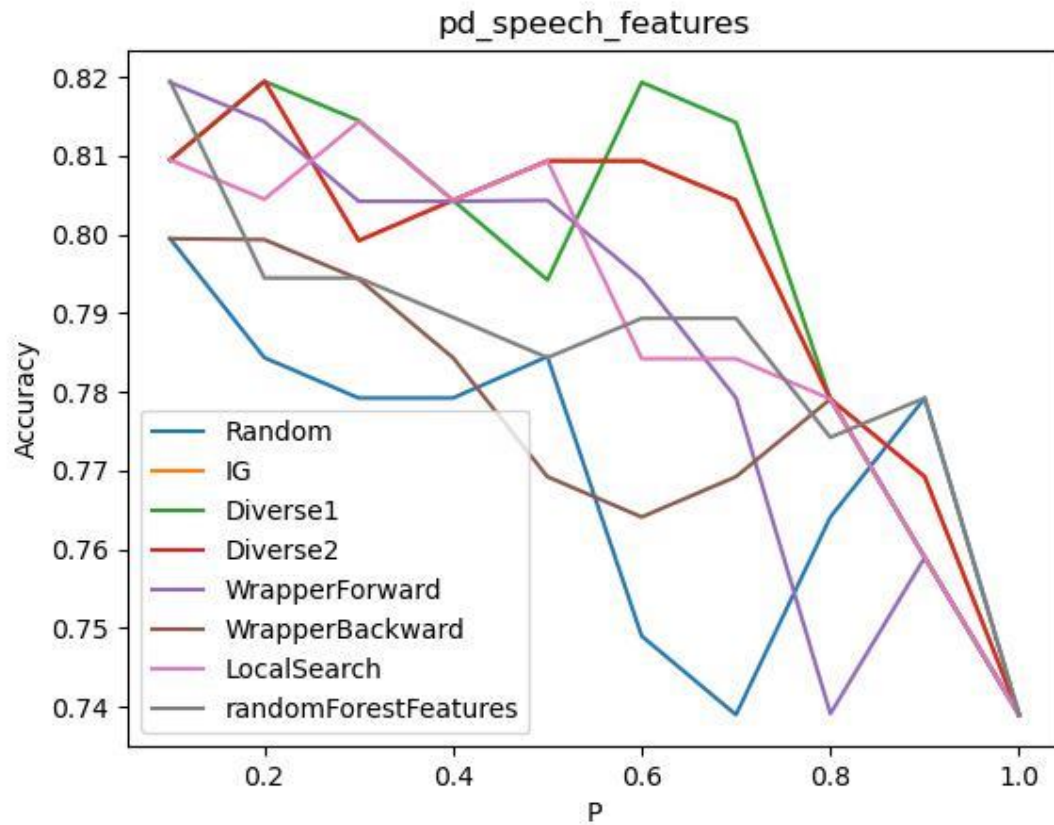
Dataset	Num of samples	Num of features
Meter_A	87	36



	0.9	0.8
Random	0.83	0.82
max ig	0.79	0.79
diverse1	0.8	0.79
diverse2	0.79	0.79
wrapperforward	0.79	0.8
wrapperbackward	0.79	0.83
local search	0.79	0.76
randomforest	0.79	0.78

Best accuracy:		
	p	accuracy
Random	0.3	0.86
max ig	0.1	0.85
diverse1	0.9	0.8
diverse2	0.1	0.85
wrapperforward	0.7	0.83
wrapperbackward	0.8	0.83
local search	1	0.79
randomforest	0.2	0.85

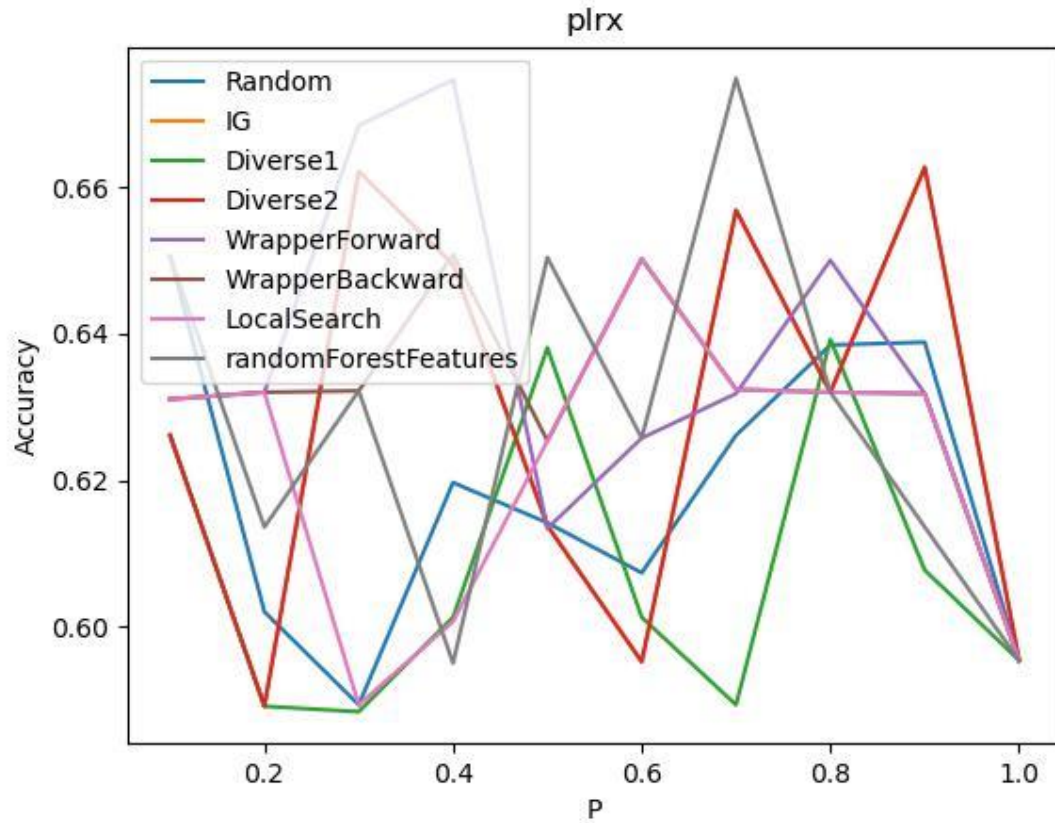
Dataset	Num of samples	Num of features
Pd speech	200	17



	0.9	0.8
random	0.78	0.76
max_ig	0.77	0.78
diverse1	0.76	0.78
diverse2	0.77	0.78
wrapperforward	0.76	0.74
wrapperbackward	0.76	0.78
localsearch	0.76	0.78
randomforest	0.78	0.77

Best accuracy:		
	p	accuracy
random	0.1	0.8
max_ig	0.2	0.82
diverse1	0.2	0.82
diverse2	0.2	0.82
wrapperforward	0.1	0.82
wrapperbackward	0.2	0.8
localsearch	0.5	0.81
randomforest	0.1	0.82

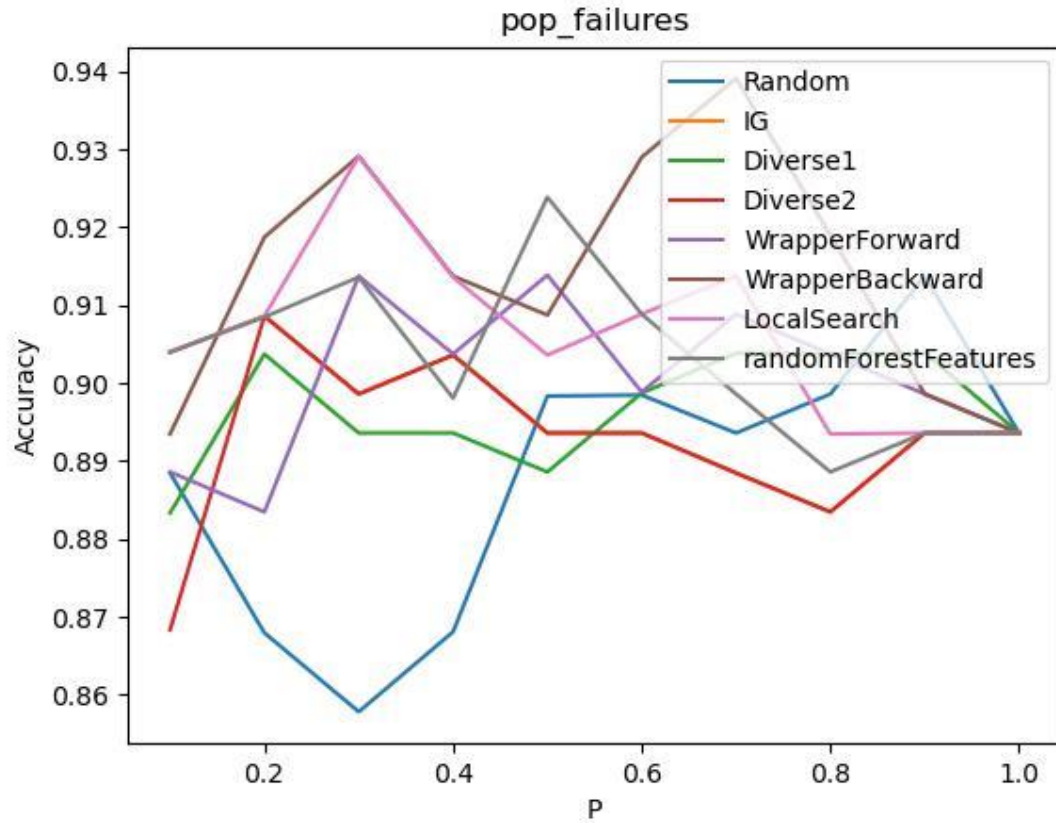
Dataset	Num of samples	Num of features
Parkinson	164	12



	0.9	0.8
Random	0.64	0.64
max ig	0.66	0.63
diverse1	0.61	0.64
diverse2	0.66	0.63
wrapperforward	0.63	0.65
wrapperbackward	0.63	0.63
local search	0.63	0.63
randomforest	0.61	0.63

Best accuracy:		
	p	accurcay
Random	0.1	0.65
max ig	0.9	0.66
diverse1	0.8	0.64
diverse2	0.9	0.66
wrapperforward	0.3	0.67
wrapperbackward	0.6	0.65
local search	0.6	0.65
randomforest	0.7	0.68

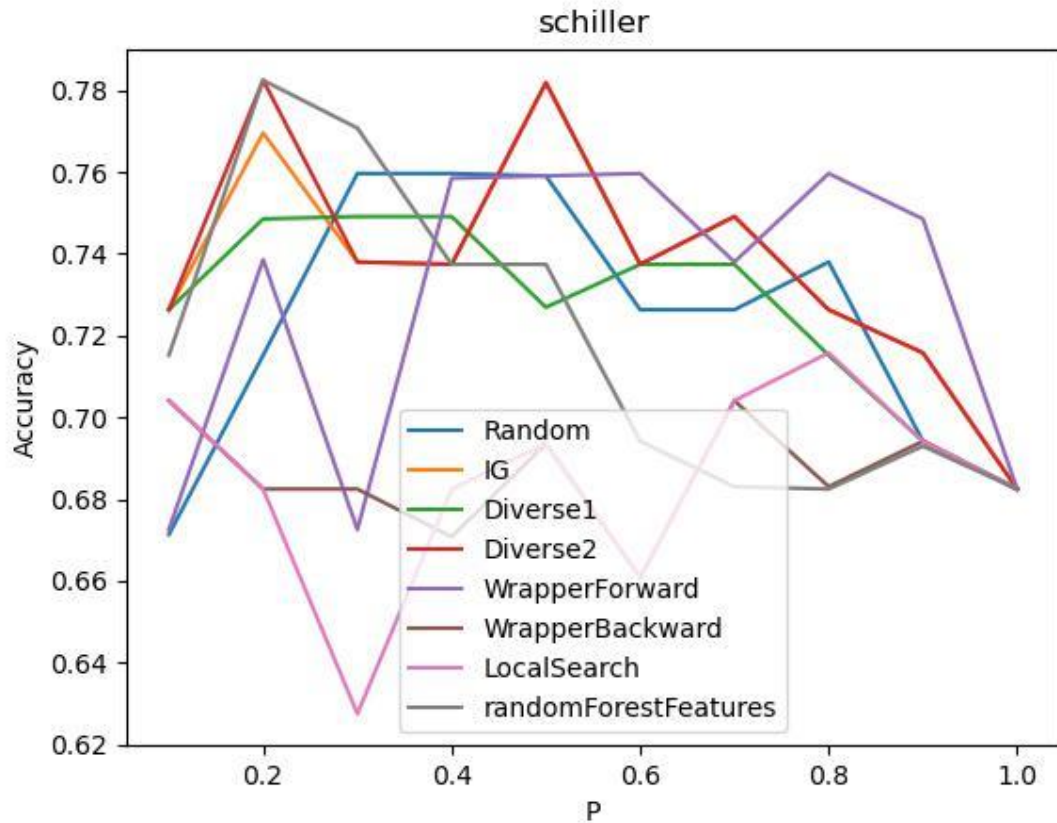
Dataset	Num of samples	Num of features
Pop_failures	199	20



	0.9	0.8
Random	0.91	0.9
max ig	0.89	0.88
diverse1	0.9	0.9
diverse2	0.89	0.88
wrapperforward	0.9	0.9
wrapperbackward	0.9	0.92
local search	0.89	0.89
randomforest	0.89	0.89

Best accuracy:		
	p	accuracy
Random	0.9	0.91
max ig	0.2	0.91
diverse1	0.9	0.9
diverse2	0.2	0.91
wrapperforward	0.7	0.91
wrapperbackward	0.7	0.94
local search	0.3	0.93
randomforest	0.5	0.92

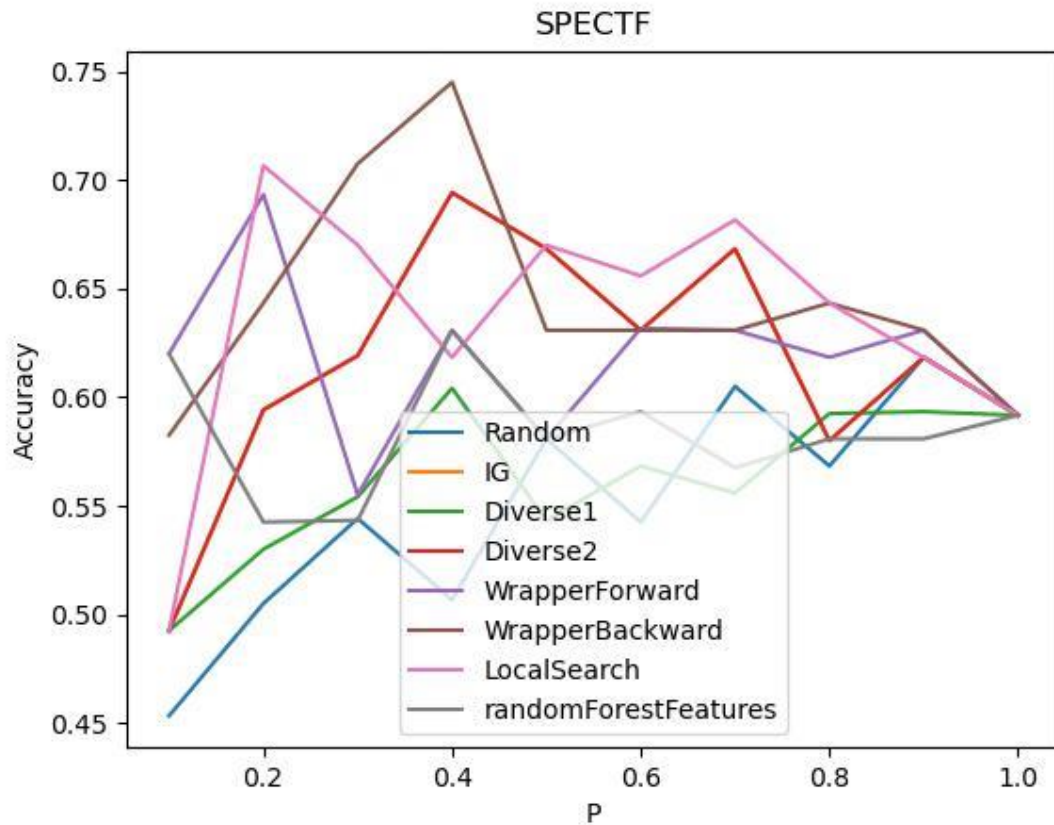
Dataset	Num of samples	Num of features
schiller	92	14



	0.9	0.8
Random	0.69	0.74
max ig	0.72	0.73
diverse1	0.69	0.72
diverse2	0.72	0.73
wrapperforward	0.75	0.76
wrapperbackward	0.69	0.68
local search	0.69	0.72
randomforest	0.69	0.68

Best accuracy:		
	p	accuracy
Random	0.5	0.76
max ig	0.5	0.78
diverse1	0.4	0.75
diverse2	0.5	0.78
wrapperforward	0.8	0.76
wrapperbackward	0.7	0.7
local search	0.8	0.72
randomforest	0.2	0.78

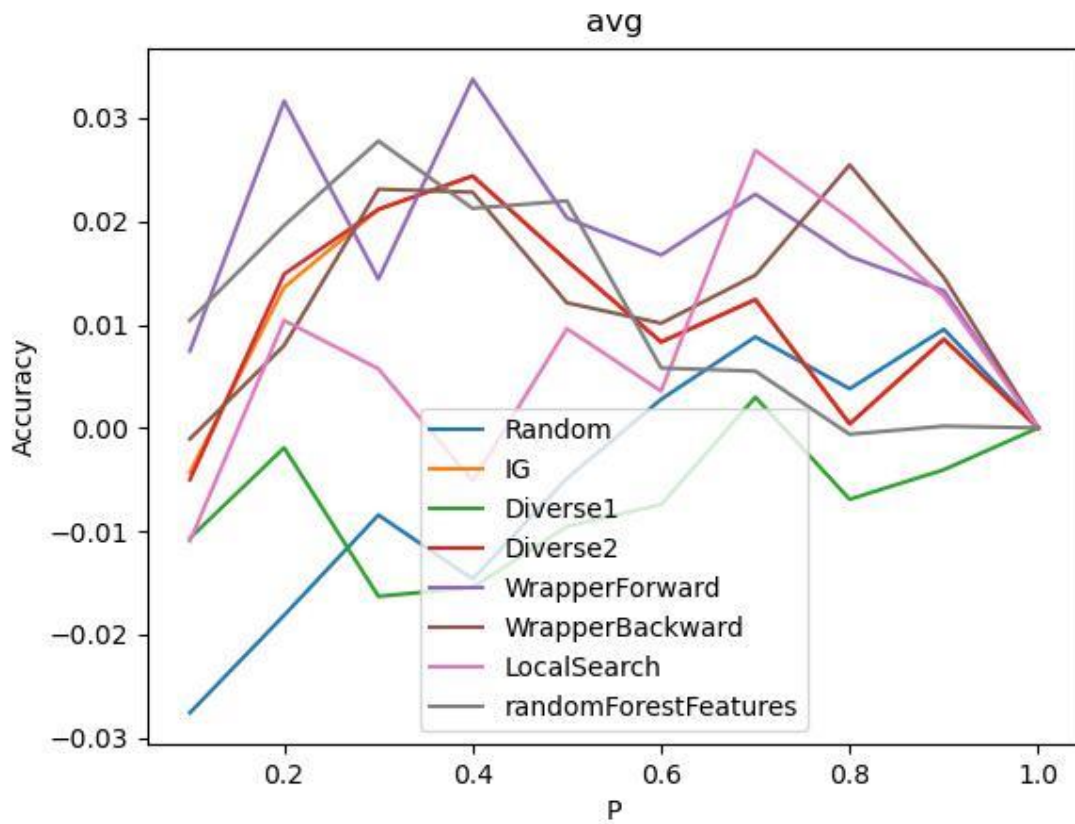
Dataset	Num of samples	Num of features
SPECTF	80	15



	0.9	0.8
Random	0.62	0.57
max ig	0.62	0.58
diverse1	0.59	0.59
diverse2	0.62	0.58
wrapperforward	0.63	0.62
wrapperbackward	0.63	0.64
local search	0.62	0.64
randomforest	0.58	0.58

Best accuracy:		
	p	accuracy
Random	1	0.59
max ig	0.4	0.69
diverse1	0.4	0.6
diverse2	0.7	0.67
wrapperforward	0.2	0.69
wrapperbackward	0.4	0.75
local search	0.2	0.71
randomforest	0.4	0.63

גרף מסכם



	1	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
Random	0	0.01	0	0.01	0	0	-0.01	-0.01	-0.02	-0.03
max ig	0	0.01	0	0.01	0.01	0.02	0.02	0.02	0.01	0
diverse1	0	0	-0.01	0	-0.01	-0.01	-0.02	-0.02	0	-0.01
diverse2	0	0.01	0	0.01	0.01	0.02	0.02	0.02	0.01	-0.01
wrapperforward	0	0.01	0.02	0.02	0.02	0.02	0.03	0.01	0.03	0.01
wrapperbackward	0	0.01	0.03	0.01	0.01	0.01	0.02	0.02	0.01	0
local search	0	0.01	0.02	0.03	0	0.01	-0.01	0.01	0.01	-0.01
randomforest	0	0	0	0.01	0.01	0.02	0.02	0.03	0.02	0.01

גרף מסכם: מסקנות

Random : אלגוריתם זה בוחר תכונות באופן רנדומלי, איך שאנחנו רואים מהגרף אם יש הרבה תכונות זה מקשה על המסווג ללמוד את הנתונים , אפשר לראות שאם נסנן עד 30% מהתכונות נקבל דיוק יותר טוב של המסווג , לאחר מכן הדיוק מתחיל לרדת כי אנחנו מאבדים חלק גדול מהתכונות.

Ig : אלגוריתם זה מסנן את התכונות בצורה לינארית ככל שאנחנו נותנים לו לסנן יותר אנו שמים לב שהוא מצליח להשיג דיוק יותר טוב , אבל מסנן בצורה יותר איטית מהאלגוריתמים האחרים, בנוסף לפעמים הוא יורד אחרי שכבר השיג דיוק טוב כי יכול להיות שהוא בוחר בתכונות שהן תלויות אחת בשנייה שי יש להן IG יותר גבוה וזה לא תורם למסווג ללמוד , עד שאנו מגיעים לאחוזים נמוכים שם אנחנו מאבדים מרבית התכונות ולכן הדיוק יורד.

Diverse 1: אלגוריתם לא רואים שיפור כאשר מסננים תכונות , אבל אפשר לראות שעבור אחוזים מסוימים אנחנו יכולים להשיג אותו דיוק כמו שהיינו בוחרים בכל התכונות נכון שלא השגנו שיפור בדיוק אבל אפשר לאמן את המסווג על פחות תכונות וכך לחסוך בזמן האימון והסיווג.

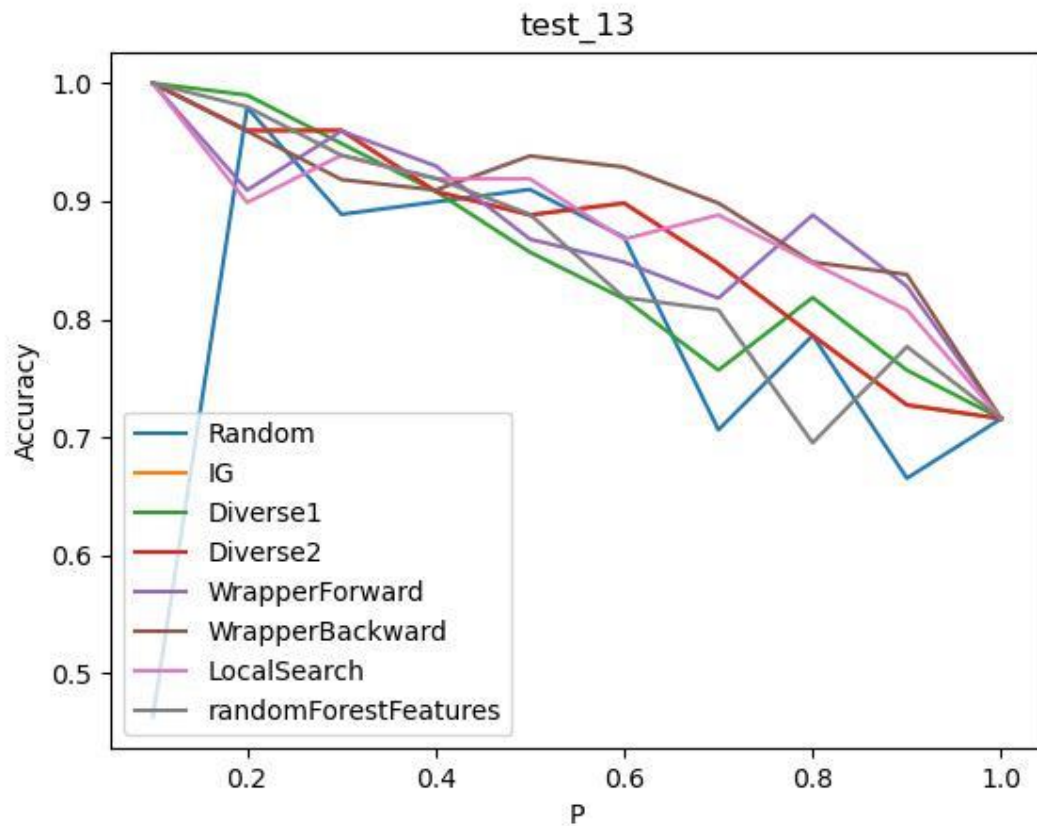
Diverse 2: אלגוריתם זה מתלכד עם אלגוריתם IG לאורך הרוב המוחלט של הניסיונות, כלומר הוא מתחיל לסנן מהתחילה אבל באופן איטי עד כשאנחנו מגיעים לאחוז מסוים, הוא מתלכד עם ה IG כי בכל האחוזים אנחנו בוחרים מהתכונות שיש לנו עד עכשיו והחלון שמכיל את התכונות הרחוקות אחת מהשנייה מתלכד עם החלון עם התכונות הטובות שבחרנו עד אחוז מסוים זה כי יוצא ש $2 \cdot k \geq \text{window size}$.

WrapperForward : אלגוריתם זה מצליח לסנן את מרבית התכונות הלא טובות כבר מההתחלה, הוא ממשיך לסנן תכונות לא טובות כל עוד אנחנו מקטנים את האחוזים, אבל רוב העבודה נעשה כבר בהתחלה הסינון , והוא מצליח לשמור על התכונות הטובות גם כאשר מקטנים את האחוזים .

WrapperBackward: אלגוריתם זה גם מצליח לסנן מרבית תכונות לא טובות כבר מההתחלה, והוא עושה את זה הכי מהיר מבין שאר האלגוריתמים , אבל ככל שאנחנו מקטנים את האחוז הדיוק יורד לעומת ה wrapperforward שמצליח לשמר את התכונות הטובות גם באחוזים הקטנים.

Local Search : אלגוריתם זה מתנהג בהתחלה בדומה ל backward הוא מצליח להשיג שיפור טוב כבר מההתחלה ולסנן את התכונות הלא רצויות שפוגעות לנו בדיוק, אבל הדיוק שלו מתחיל לרדת מוקדם יותר מה backward כי אנחנו מתחילים עם חלון אקראי מתוך תת הקבוצה של התכונות שנשארה לנו.

RandomForest : אלגוריתם זה לא מצליח להשיג שיפור משמעותי בהתחלה , אפשר לראות שאלגוריתם זה מביא לדיוק הטוב ביותר כאשר אנחנו בוחרים באחוזים נמוכים של התכונות יחסית לשאר האלגוריתמים, הוא גם מצליח לשמר על התכונות הטובות כאשר אנחנו עוברים לאחוזים יותר נמוכים.

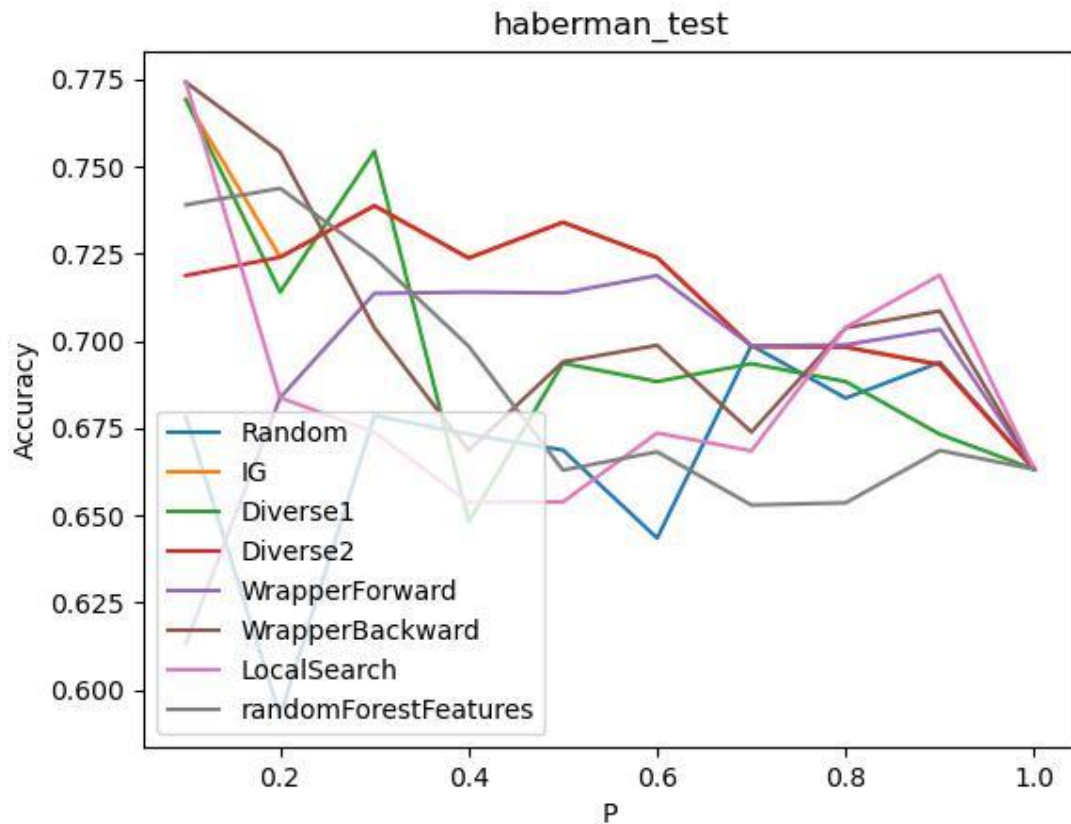


באן עשינו ניסוי לבדוק את התקינות של האלגוריתמים שלנו בנינו dataset מלאכותי, הוא מכיל 30 תכונות רועשות שנוצרו רנדומלית ותכונה אחת רלוונטית, כל התכונות מכילות הערך 1 או 0, והסיווג הוא M או B, אם הדוגמה מסווגת להיות M אז התכונה הרלוונטית היחידה שלנו מכילה את הערך 0, אחרת מכילה את הערך B.

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



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

סיכום

בפריקט זה אנחנו בחנו כמה אלגוריתמים לסינון תכונות 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.

אנו מבחינים כאשר אנחנו נרצה לסנן אחוז קטן מהתכונות כדאי להשתמש באלגוריתמים local search ו wrapperbackward כי הם מצליחים להשיג שיפור טוב כבר מההתחלה, כל האלגוריתמים לעומת diverse1 וה local search מתנהגים כמעט באותו אופן באחוזים נמוכים.

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/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