Робота з деревами ухвалення рішень в Haskell

НАУКОВИЙ КЕРІВНИК ПРОЦЕНКО В.С

Мета роботи: реалізувати алгоритм побудови дерев ухвалення рішень за допомогою Haskell, застосувати його та оцінити результати

Об’єкт дослідження: дерева ухвалення рішень

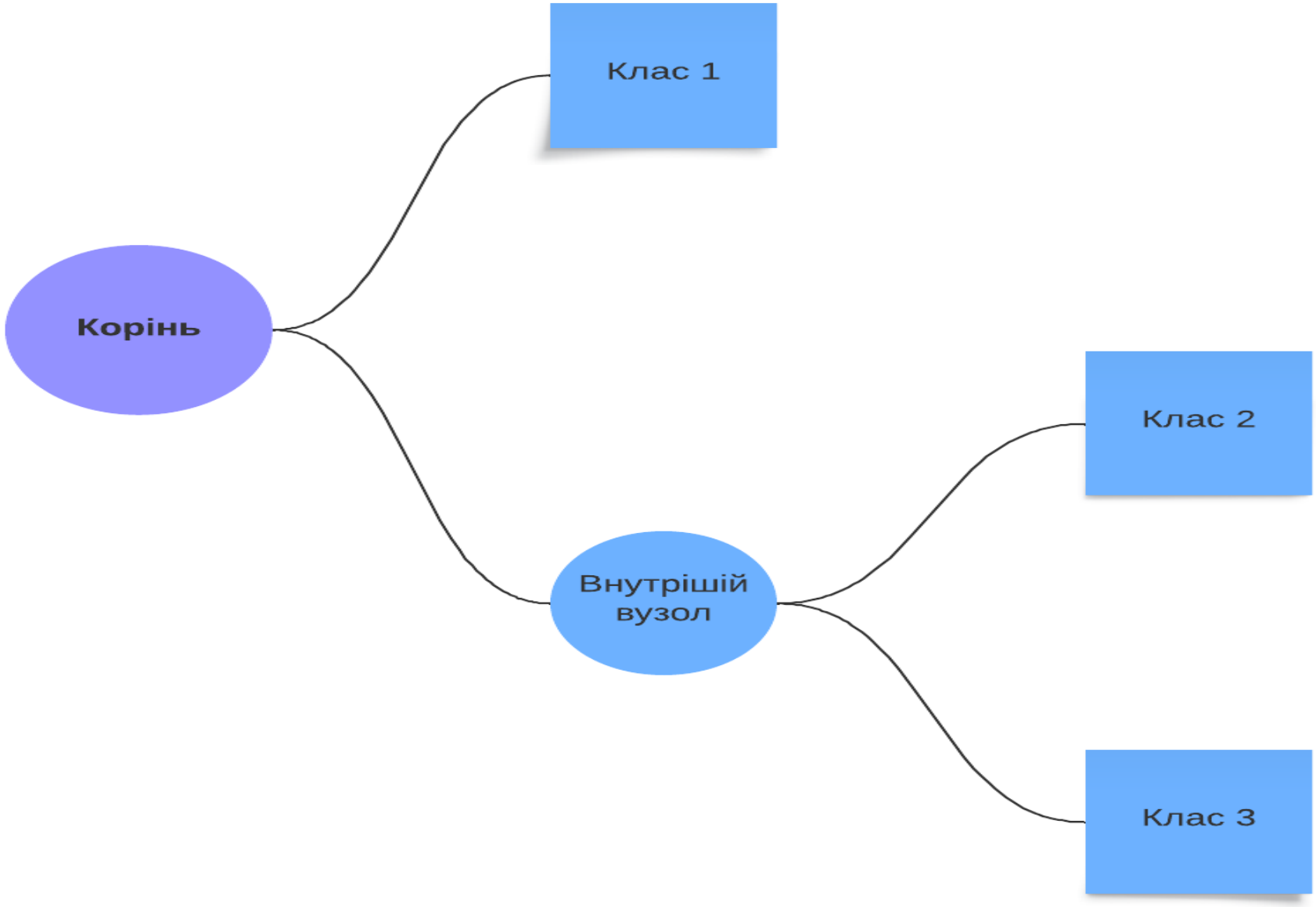
Предмет дослідження: побудова дерев ухвалення рішень за допомогою Haskell

Дерево ухвалення рішень

Його можна зобразити у вигляді графу $G = (V, E)$, який складається з непорожньої множини вузлів V та множини ребер(«гілок») E, та має наступні властивості:

- Ребра є впорядкованими парами вершин (v, w), тобто граф є направленим
- Граф є ациклічним
- Існує єдиний вузол, у який не входить жодного ребра(корінь дерева)
- Усі вузли, окрім кореня, мають єдине вхідне ребро
- Існує шлях $(v_1, v_2), (v_2, v_3), \dots, (v_{n-1}, v_n)$ від кореня до кожного з вузлів

- Якщо існує шлях з вузла v до w, $v \neq w$, v називають *батьківським*(або прабатьківським, якщо шлях довший за 1 ребро) вузлом w. Відповідно, w називають *нащадком*. Якщо у вузла відсутні нащадки – він є «*листком*», або кінцевим вузлом. Решта вузлів(окрім кореня) – *внутрішні*.


Клас 1
Корінь
Клас 2
Внутрішій вузол
Клас 3

ID3

$$E(D) = -\sum_{i=1}^{m} p_i \cdot log_2(p_i)$$

Де p_i – пропорція елементів D, яка належать класу i, m – кількість класів, а логарифмічна функція за основою 2 використовується через кодування інформації комп’ютерами в бітах.

Приріст інформації для атрибуту A відносно множини D:

$$Gain(D, A) = E(D) - \sum_{j=1}^{n} \frac{|D_j|}{|D|} E(D_j)$$

Тут множина розбита на n частин відповідно до значень атрибуту A, D_j — підмножина D, у якої атрибут A має значення j. Приріст інформації працює лише з категорійними даними.

C4.5

$$GainRatio(D, A) = \frac{Gain(D, A)}{SplitInformation(D, A)}$$

$$SplitInformation(D, A) = -\sum_{j=1}^{n} \frac{|D_j|}{|D|} \cdot log_2\left(\frac{|D_j|}{|D|}\right)$$

$SplitInformation$ – інформація, отриманої від розділення множини D на n підмножин на основі значення атрибуту A

```
data DecisionTree = MyNull |
                    Leaf AttValue |
                    Node AttName [(AttValue, DecisionTree)]
                  deriving (Eq, Show)
```

А також задамо вигляд розбиття датасету:

```
type Partition = [(AttValue, DataSet)]
```

- AttValue – значення атрибуту, AttName – його назва
- Leaf – листок дерева, міститиме AttName залежної змінної
- Node – корінь та внутрішні вузли, міститиме AttName та піддерево, утворене розбиттям за цим атрибутом

```
partitionData :: DataSet -> Attribute -> Partition

partitionData (title, rows) (attName, attVals)
```

```
  = [(attName', (title', rows')) | (attName', rows') <- attNameRows']
  where
    title'    = remove attName title --прибираємо з заголовку
    attVals'  = map (lookUpAtt attName title) rows --знаходимо і зберігаємо значення атрибуту з усіх рядків
    rows'     = map (removeAtt attName title) rows --прибираємо з рядків
    --оновлюємо прив'язку значеннь атрибутів та рядків
    attNameRows' = foldr addToMapping (zip attVals (repeat [])) (zip attVals' rows')
```

fishingData

Атрибути:

- $outlook = sunny, overcast, rainy$
- $temp = cool, mild, hot$
- $humidity = normal, hight$
- $wind = calm, windy$
- $result = good, bad$

Partition за outlook

sunny

temp	humidity	wind	result
hot	high	calm	good
cool	normal	windy	good
mild	high	windy	good
hot	normal	calm	good
temp	humidity	wind	result

overcast

temp	humidity	wind	result
hot	high	calm	good
cool	normal	windy	good
mild	high	windy	good
hot	normal	calm	good
temp	humidity	wind	result

rainy

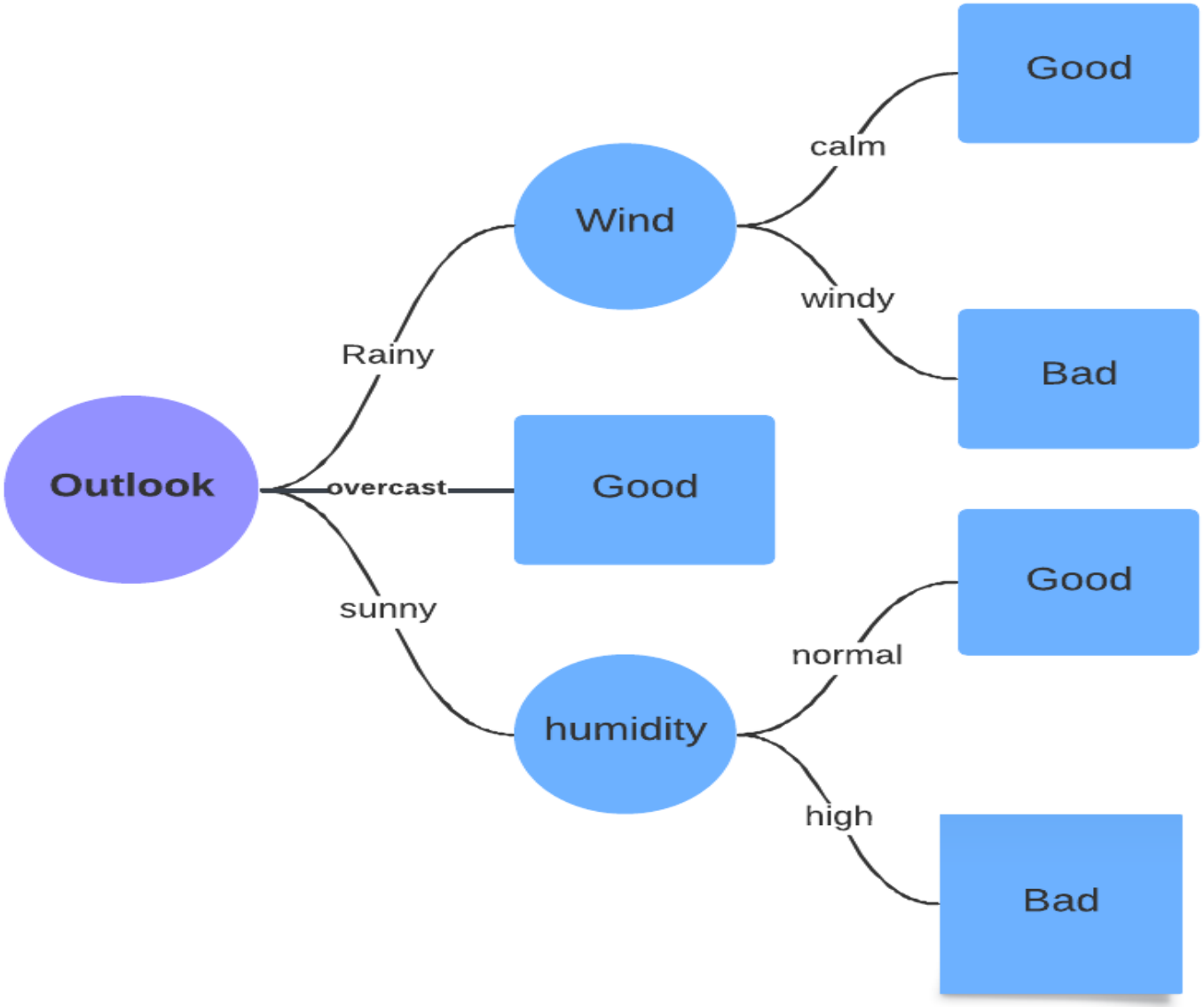
temp	humidity	wind	result
mild	high	calm	good
cool	normal	calm	good
cool	normal	windy	bad
mild	normal	calm	good
mild	high	windy	bad

```
bestGainAtt :: AttSelector

bestGainAtt dataset@(title, _) attribute@(attName, _)
  = selectedAtt
  where
    remainingAtts = remove attName title
    gains = map (\att -> gain dataset att attribute) remainingAtts --обчислюємо приріст для решти атрибутів
    maxGain = maximum gains
    selectedAtt = remainingAtts !! fromJust (elemIndex maxGain gains)
```

```
buildTree ds@(title, rows) att@(attName, attVals) selector

  -- якщо значення в датасеті для атрибуту однакові додаємо листок

  | allSame attValsInRows = Leaf (head attValsInRows)

  -- якщо ні - додаємо вузол

  | otherwise = Node attName' [(attVal', buildTree ds' att selector) | (attVal', ds') <- partit]

  where

    -- беремо значення атрибуту з датасету

    attValsInRows = map (lookUpAtt attName title) rows

    -- застосовуємо селектор

    nextAtt@(attName', attVals') = selector ds att

    -- розділяємо датасет за атрибутом

    partit = partitionData ds nextAtt
```

>>buildTree fishingData result bestGainAtt

```
"outlook": {
            "sunny":{
                    "humidity": {
                                "high":"bad",
                                "normal":"good"}},
            "overcast":"good",
            "rainy":{ "wind": {
                                "windy":"bad",
                                "calm":"good"}}}
```


Outlook
Rainy
overcast
sunny
Wind
Good
humidity
calm
windy
Good
Bad
normal
high
Good
Bad

		Реальні значення			
		y_1	y_2	…	y_k
Передбачені значення	y_1	$y_{1,1}$	*FP*	…	$y_{1,k}$
	y_2	*FN*	*TP*	…	*FN*
	…	…	…	…	…
	y_k	$y_{k,1}$	*FP*	…	$y_{k,k}$

Таб. 3.2 Матриця для k класів

Метрики

$$PPV(y_i) = \frac{TP(y_i)}{TP(y_i)+FP(y_i)}$$

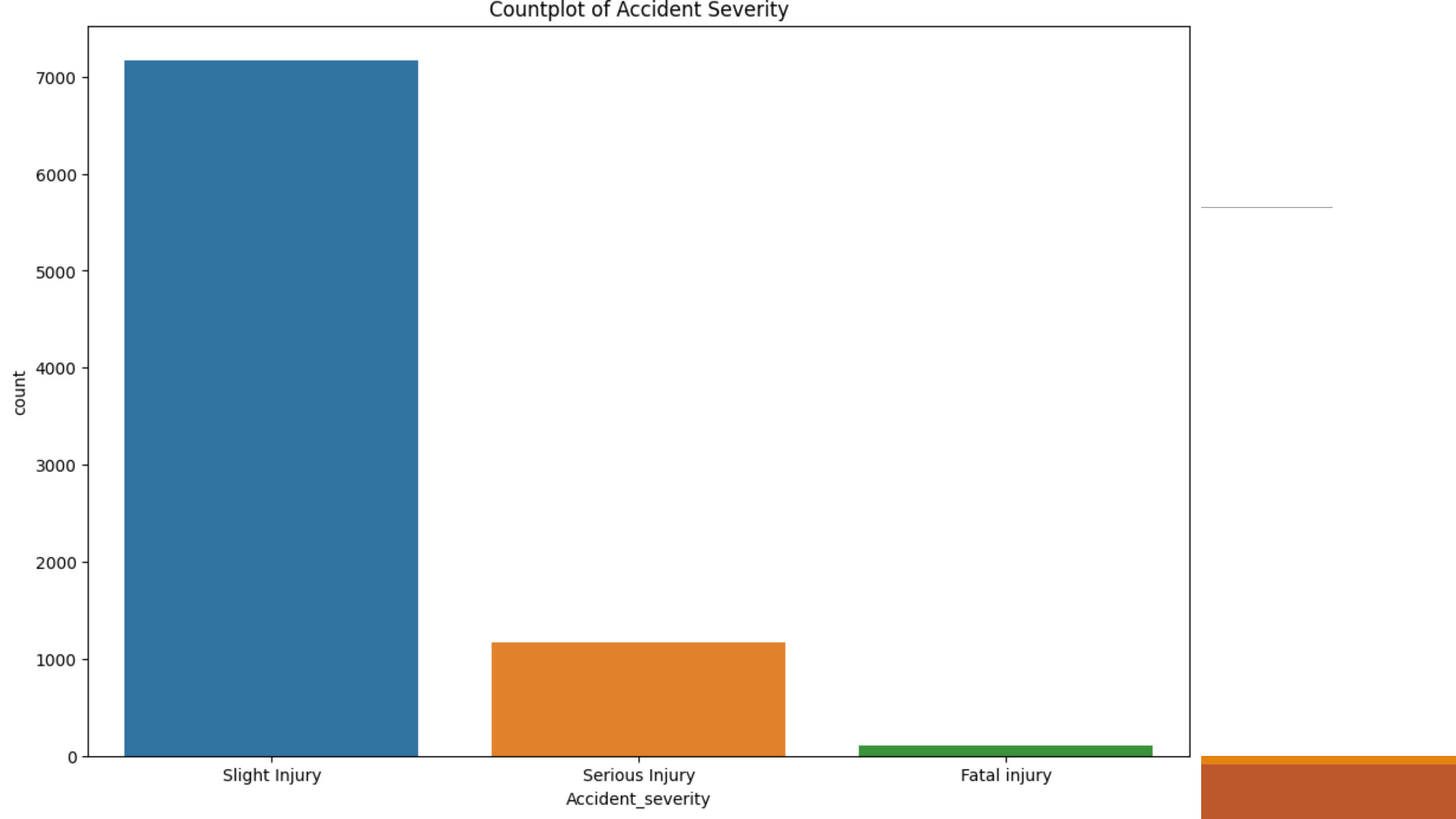
$$TPR(y_i) = \frac{TP(y_i)}{TP(y_i)+FN(y_i)}$$

$$F_1(y_i) = 2 \cdot \frac{TPR(y_i) \cdot PPV(y_i)}{TPR(y_i)+PPV(y_i)}$$

Road Accident Severity in India

https://www.kaggle.com/datasets/s3programmer/road-accident-severity-in-india

k	*AttName*	*AttValue*
1	day	Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
2	driver_age	18-30, 31-50, Over 51, Unknown driver age, Under 18
3	driver_education	Junior high school, Above high school, High school, Unknown driver education, Elementary school, Illiterate, Writing & reading
4	driver_sex	"Male", "Female", "Unknown driver sex"
…	…	…
28	accident_severity	Serious Injury, Slight Injury, Fatal injury


Countplot of Accident Severity
7000
6000
5000
4000
3000
2000
1000
0
count
Slight Injury
Serious Injury
Fatal injury
Accident_severity

```
main :: IO ()
main = do
    let k = 10
    rows <- tab
    shuffledRows <- shuffleM rows
    let n = length shuffledRows `div` k
        folds = chunksOf n shuffledRows

    results <- kFoldCrossValidation t folds classifierAttribute bestGainAtt

    --після згортки маємо суми метрик
    let (avgAcc, avgPrecision, avgRecall, avgF1Score) = foldr (\(acc, prec, recl, f1)
(accSum, precSum, recSum, f1Sum) ->
            (accSum + acc, precSum + prec, recSum + recl, f1Sum + f1)) (0, 0, 0, 0) results
        avgAcc' = avgAcc / fromIntegral k
        avgPrecision' = avgPrecision / fromIntegral k
        avgRecall' = avgRecall / fromIntegral k
        avgF1Score' = avgF1Score / fromIntegral k
    printf "Average Accuracy: %.4f\n" avgAcc'
    printf "Average Precision: %.4f\n" avgPrecision'
    printf "Average Recall: %.4f\n" avgRecall'
    printf "Average F1: %.4f\n" avgF1Score'
```

У результаті, при $k = 10$

Average Accuracy: 0.8582

Average Precision: 0.4415

Average Recall: 0.4645

Average F1: 0.2224

При k=23, по 75 рядків:

Average Accuracy: 0.7941

Average Precision: 0.7318

Average Recall: 0.6577

Average F1: 0,346

Дякую за увагу!