
	Politechnika Bydgoska im. J.J. Śniadeckich Wydział Telekomunikacji, Informatyki i Elektrotechniki		
<b>Przedmiot</b>	Algorytmy i eksploracja danych		
<b>Prowadzący</b>	dr inż. Michał Kruczkowski		
<b>Temat</b>	Uczenie z nadzorem		
<b>Studenci</b>	Adam Szreiber, Cezary Naskręt		
<b>Nr ćw.</b>	4	<b>Data wykonania</b>	16.10.2023

## 1. Cel ćwiczenia

Celem dzisiejszego ćwiczenia było zapoznanie się z różnymi aspektami redukcji wymiarowości danych i wykonywanie praktycznych kroków związanych z tym procesem, takich jak obliczanie wymiarów wczytanych danych, zapisywanie przetworzonych danych do pliku csv, implementacja algorytmów redukcji wymiarowości takich jak PCA, obliczanie wariancji zbioru i wartości wariancji wyjaśnionej, obliczanie sumy składowej i poznanie innych sposobów redukcji wymiarowości.

## 2. Przebieg laboratorium

### 2.1. Zadanie 1

Zbiór danych:

<https://archive.ics.uci.edu/ml/datasets/SmartphoneBased+Recognition+of+Human+Activities+and+Postural+Transitions>

- Treningowy: X\_train.txt / y\_train.txt
- Testowy: X\_test.txt / y\_test.txt

Link podany w ćwiczeniu nie jest już dostępny. Natomiast te same dane znalazłem pod linkiem:

<https://archive.ics.uci.edu/dataset/341/smartphone+based+recognition+of+human+activities+and+postural+transitions>

## 2.2. Zadanie 2

Zbuduj modele predykcyjne na podstawie następujących algorytmów zakładając domyślne

parametry:

- SVM
- kNN
- Decision Tree
- Random Forest

Na początku importuje wszystkie biblioteki, jakie będę używał w projekcie oraz wczytuję dane treningowe i testowe.

```
# import libraries
import numpy as np

from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier

# load train data
X = np.loadtxt("Train/X_train.txt")
y = np.loadtxt("Train/y_train.txt")

# load test data
X_test = np.loadtxt("Test/X_test.txt")
y_test = np.loadtxt("Test/y_test.txt")
```

Kod dla modelu SVM.

```
#####
# make SVM model
#####
svm = SVC(probability=True).fit(X, y)
svm_predict = svm.predict(X_test)

# calculate accurate predictions
correct = len(y_test) - np.count_nonzero(np.subtract(y_test,
svm_predict))
print("Model SVM poprawnie zaklasyfikował " + str(correct) + " próbek na "
+ str(len(y_test)))
```

## Kod dla modelu kNN.

```
#####  
# make kNN model  
#####  
knn = KNeighborsClassifier().fit(X, y)  
knn_predict = knn.predict(X_test)  
  
# calculate accurate predictions  
correct = len(y_test) - np.count_nonzero(np.subtract(y_test,  
knn_predict))  
print("Model kNN poprawnie zaklasyfikował " + str(correct) + " próbek na  
" + str(len(y_test)))
```

## Kod dla modelu Decision Tree.

```
#####  
# make Decision Tree model  
#####  
decisionTree = tree.DecisionTreeClassifier().fit(X, y)  
decisionTree_predict = decisionTree.predict(X_test)  
  
# calculate accurate predictions  
correct = len(y_test) - np.count_nonzero(np.subtract(y_test,  
decisionTree_predict))  
print("Model Decision Tree poprawnie zaklasyfikował " + str(correct) + "  
próbek na " + str(len(y_test)))
```

## Kod dla modelu Random Forest.

```
#####  
# make Random Forest  
#####  
forest = RandomForestClassifier().fit(X, y)  
forest_predict = forest.predict(X_test)  
  
# calculate accurate predictions  
correct = len(y_test) - np.count_nonzero(np.subtract(y_test,  
forest_predict))  
print("Model Random Forest poprawnie zaklasyfikował " + str(correct) + "  
próbek na " + str(len(y_test)))
```

Kilka razy uruchamiam powyższy kod i otrzymuje następujące wyniki.

```
Model SVM poprawnie zaklasyfikował 2961 próbek na 3162
Model kNN poprawnie zaklasyfikował 2798 próbek na 3162
Model Decision Tree poprawnie zaklasyfikował 2532 próbek na 3162
Model Random Forest poprawnie zaklasyfikował 2883 próbek na 3162

Model SVM poprawnie zaklasyfikował 2961 próbek na 3162
Model kNN poprawnie zaklasyfikował 2798 próbek na 3162
Model Decision Tree poprawnie zaklasyfikował 2578 próbek na 3162
Model Random Forest poprawnie zaklasyfikował 2886 próbek na 3162

Model SVM poprawnie zaklasyfikował 2961 próbek na 3162
Model kNN poprawnie zaklasyfikował 2798 próbek na 3162
Model Decision Tree poprawnie zaklasyfikował 2557 próbek na 3162
Model Random Forest poprawnie zaklasyfikował 2885 próbek na 3162
```

### 2.3. Zadanie 3

Na podstawie zdefiniowanych metryk oceny klasyfikatorów oceń skuteczność klasyfikacji algorytmów z zadania 2

- Confusion matrix: TP / TN / FP / FN
- ACC
- Recall
- F1
- AUC

Na początku importuje wszystkie biblioteki, jakie będę używał w projekcie oraz wczytuję dane treningowe i testowe.

```
# import libraries
import numpy as np

from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score

# load train data
X = np.loadtxt("Train/X_train.txt")
y = np.loadtxt("Train/y_train.txt")
X = X[0:100]
y = y[0:100]

# load test data
X_test = np.loadtxt("Test/X_test.txt")
y_test = np.loadtxt("Test/y_test.txt")
```

## Kod dla modelu SVM.

```
#####
# make SVM model
#####
svm = SVC(probability=True).fit(X, y)
svm_predict = svm.predict(X_test)

# calculate accurate predictions
print("SVM")
print(confusion_matrix(y_test, svm_predict))
print(accuracy_score(y_test, svm_predict))
print(recall_score(y_test, svm_predict, average='micro'))
print(f1_score(y_test, svm_predict, average='micro'))
print(roc_auc_score(y, svm.predict_proba(X), multi_class='ovr'))
```

## Kod dla kNN.

```
#####
# make kNN model
#####
knn = KNeighborsClassifier().fit(X, y)
knn_predict = knn.predict(X_test)

# calculate accurate predictions
print("kNN")
print(confusion_matrix(y_test, knn_predict))
print(accuracy_score(y_test, knn_predict))
print(recall_score(y_test, knn_predict, average='micro'))
print(f1_score(y_test, knn_predict, average='micro'))
print(roc_auc_score(y, knn.predict_proba(X), multi_class='ovr'))
```

## Kod dla Decision Tree.

```
#####
# make Decision Tree model
#####
decisionTree = tree.DecisionTreeClassifier().fit(X, y)
decisionTree_predict = decisionTree.predict(X_test)

# calculate accurate predictions
print("Decision Tree")
print(confusion_matrix(y_test, decisionTree_predict))
print(accuracy_score(y_test, decisionTree_predict))
print(recall_score(y_test, decisionTree_predict, average='micro'))
print(f1_score(y_test, decisionTree_predict, average='micro'))
print(roc_auc_score(y, decisionTree.predict_proba(X),
multi_class='ovr'))
```

## Kod dla Random Forest.

```
#####
# make Random Forest
#####
forest = RandomForestClassifier().fit(X, y)
forest_predict = forest.predict(X_test)

# calculate accurate predictions
print("Random Forest")
print(confusion_matrix(y_test, forest_predict))
print(accuracy_score(y_test, forest_predict))
print(recall_score(y_test, forest_predict, average='micro'))
print(f1_score(y_test, forest_predict, average='micro'))
print(roc_auc_score(y, forest.predict_proba(X), multi_class='ovr'))
```

## Wydruk z konsoli kompilatora dla modelu SVM.

```
SVM
[[496  0  0  0  0  0  0  0  0  0  0  0]
 [453  0  0 18  0  0  0  0  0  0  0  0]
 [420  0  0  0  0  0  0  0  0  0  0  0]
 [  0  0  0 144 363  1  0  0  0  0  0  0]
 [  0  0  0 20536  0  0  0  0  0  0  0  0]
 [  0  0  0  0 545  0  0  0  0  0  0  0]
 [  7  0  0 14  1  1  0  0  0  0  0  0]
 [  1  0  0  8  0  1  0  0  0  0  0  0]
 [  1  0  0 20  0 11  0  0  0  0  0  0]
 [  0  0  0  1  0 24  0  0  0  0  0  0]
 [ 16  0  0 20  1 12  0  0  0  0  0  0]
 [  1  0  0  0  0 26  0  0  0  0  0  0]]
0.5442757748260595
0.5442757748260595
0.5442757748260595
0.9427169336963152
kNN
```

Dalsza część wydruku z konsoli kompilatora dla modeli kNN i Decision Tree.

kNN

```
[[496 0 0 0 0 0 0 0 0 0 0 0 0]
[457 0 0 14 0 0 0 0 0 0 0 0 0]
[420 0 0 0 0 0 0 0 0 0 0 0 0]
[ 0 0 0 209 298 1 0 0 0 0 0 0 0]
[ 0 0 0 38 517 1 0 0 0 0 0 0 0]
[ 1 0 0 35 7 502 0 0 0 0 0 0 0]
[ 7 0 0 13 0 0 1 0 0 0 2 0 0]
[ 1 0 0 5 0 0 0 0 0 2 2 0 0]
[ 1 0 0 5 0 2 0 0 5 2 17 0 0]
[ 0 0 0 0 0 0 0 0 1 23 1 0 0]
[ 16 0 0 5 1 2 0 0 7 2 16 0 0]
[ 2 0 0 0 0 0 0 0 0 21 4 0 0]]
```

0.5594560404807084

0.5594560404807084

0.5594560404807084

0.9938133792373556

Decision Tree

```
[[244 0 0 0 221 0 0 0 0 0 31 0]
[362 0 0 3 7 0 0 0 10 1 88 0]
[355 0 0 0 34 0 0 0 0 0 31 0]
[ 0 0 0 252 244 0 2 1 9 0 0 0]
[ 0 0 0 204 327 0 1 0 23 0 1 0]
[ 2 0 0 1 0 532 0 2 6 2 0 0]
[ 1 0 0 2 0 0 4 0 2 0 14 0]
[ 0 0 0 2 0 0 2 2 2 0 2 0]
[ 7 0 0 1 0 0 4 0 7 4 9 0]
[ 1 0 0 0 0 0 0 1 6 6 10 1]
[ 12 0 0 1 1 0 2 0 4 3 26 0]
[ 7 0 0 2 0 0 0 2 2 1 12 1]]
```

0.4430740037950664

0.4430740037950664

0.4430740037950664

1.0

Ostatnia część wydruku z konsoli dla modelu Random Forest.

```
Random Forest
[[493 0 0 0 0 0 2 0 0 0 1 0]
 [377 0 0 0 0 0 61 0 0 0 33 0]
 [413 0 0 0 0 0 5 0 0 0 2 0]
 [ 0 0 0 207 295 5 1 0 0 0 0 0]
 [ 0 0 0 8 548 0 0 0 0 0 0 0]
 [ 1 0 0 79 10 455 0 0 0 0 0 0]
 [ 3 0 0 3 2 0 10 0 0 0 5 0]
 [ 0 0 0 4 0 0 1 3 1 0 1 0]
 [ 2 0 0 5 1 0 0 1 5 0 18 0]
 [ 2 0 0 2 0 0 0 0 4 10 7 0]
 [ 4 0 0 3 1 1 1 0 4 1 34 0]
 [ 3 0 0 0 0 1 0 0 1 9 13 0]]
0.558191018342821
0.558191018342821
0.558191018342821
1.0
```

## 2.4. Zadanie 4

Dokonaj wyboru najlepszego algorytmu klasyfikacji na podstawie kros-walidacji (CV) dla 5 próbek. Jako rezultat zwróć następujące parametry:

- Wartości średniej z wyników klasyfikacji
- Średniego odchylenia standardowego z wyników klasyfikacji

Tak jak w poprzednich zadaniach, na początku importuje biblioteki i wczytuje dane treningowe oraz dane testowe.

```
# import libraries
import numpy as np
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

# load train data
X = np.loadtxt("Train/X_train.txt")
y = np.loadtxt("Train/y_train.txt")

# load test data
X_test = np.loadtxt("Test/X_test.txt")
y_test = np.loadtxt("Test/y_test.txt")
```



### Wykonanie kros-walidacji dla SVM.

```
#####  
# make SVM model  
#####  
svm = SVC(probability=True).fit(X, y)  
svm_predict = svm.predict(X_test)  
  
# calculate accurate predictions  
print("SVM")  
scores = cross_val_score(svm, X, y, cv=5)  
print("%0.2f accuracy with a standard deviation of %0.2f" %  
      (scores.mean(), scores.std()))
```

### Wykonanie kros-walidacji dla kNN.

```
#####  
# make kNN model  
#####  
knn = KNeighborsClassifier().fit(X, y)  
knn_predict = knn.predict(X_test)  
  
# calculate accurate predictions  
print("kNN")  
scores = cross_val_score(knn, X, y, cv=5)  
print("%0.2f accuracy with a standard deviation of %0.2f" %  
      (scores.mean(), scores.std()))
```

### Wykonanie kros-walidacji dla Decision Tree.

```
#####  
# make Decision Tree model  
#####  
decisionTree = DecisionTreeClassifier().fit(X, y)  
decisionTree_predict = decisionTree.predict(X_test)  
  
# calculate accurate predictions  
print("Decision Tree")  
scores = cross_val_score(decisionTree, X, y, cv=5)  
print("%0.2f accuracy with a standard deviation of %0.2f" %  
      (scores.mean(), scores.std()))
```

### Wykonanie kros-walidacji dla Random Forest.

```
#####
# make Random Forest
#####
forest = RandomForestClassifier().fit(X, y)
forest_predict = forest.predict(X_test)

# calculate accurate predictions
print("Random Forest")
scores = cross_val_score(forest, X, y, cv=5)
print("%.2f accuracy with a standard deviation of %.2f" %
      (scores.mean(), scores.std()))
```

Wyniki kros-walidacji dla wszystkich czterech modeli zawierające wartości średniej z wyników klasyfikacji oraz średniego odchylenia standardowego z wyników klasyfikacji.

```
SVM
0.92 accuracy with a standard deviation of 0.02
kNN
0.89 accuracy with a standard deviation of 0.01
Decision Tree
0.84 accuracy with a standard deviation of 0.02
Random Forest
0.91 accuracy with a standard deviation of 0.02
```

## 2.5. Zadanie 5

Zaprezentuj graficznie wynik klasyfikacji i testowania wyselekcjonowanego algorytmu:

- Wykres typu scatterplot
- Etykieta klasy -> unikatowy kolor i kształt
- Wykres powinien zawierać dla danego klasyfikatora następujące podwykresy:
  - o Rozkład próbek treningowych / testowych z podziałem na klasy
  - o Wynik trenowania modelu z podziałem na klasy
  - o Wynik testowania modelu z podziałem na klasy

Kod programu znajduje się poniżej. Na początku importuje niezbędne biblioteki i wczytuje dane testowe. Następnie skaluje dane w taki sposób, aby posiadały dwa wymiary za pomocą algorytmu PCA. W kolejnym kroku trenuje model SVC. Formatuje odpowiednio dane za pomocą metody `DecisionBoundaryDisplay.from_estimator()` z biblioteki `sklearn.inspection`. Wyświetlam otrzymane wyniki.

```
import pandas as pd # to load the dataframe
from sklearn.decomposition import PCA # to apply PCA
import matplotlib.pyplot as plt
import numpy as np
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

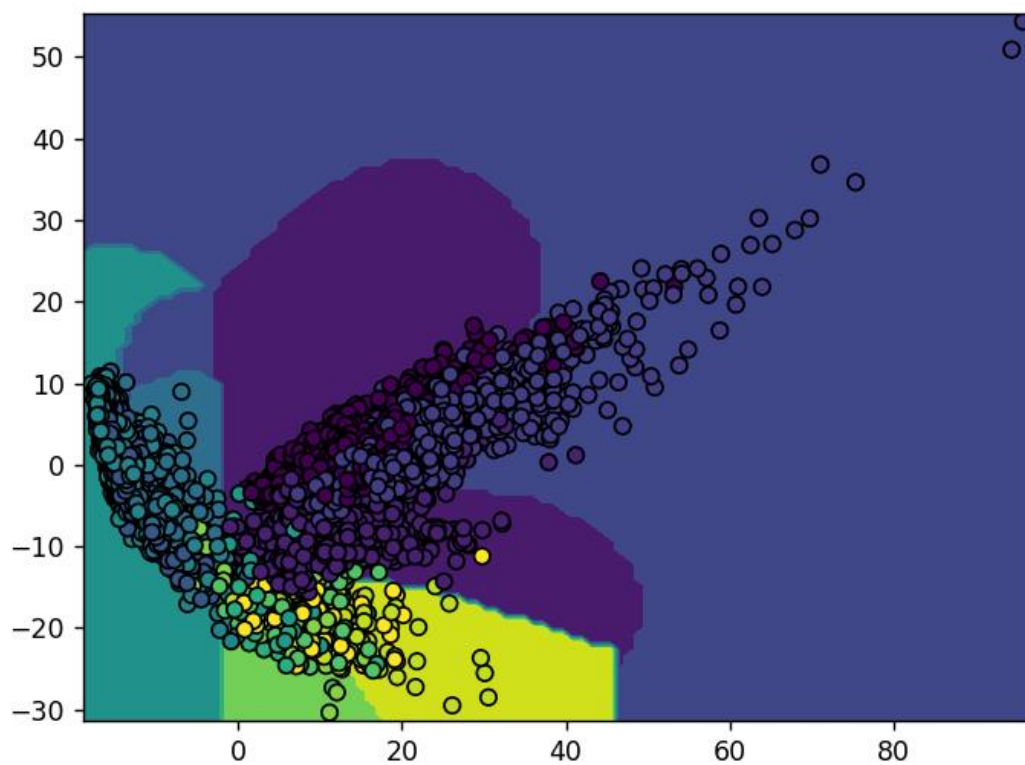
# load train data
X = np.loadtxt("Train/X_train.txt")
y = np.loadtxt("Train/y_train.txt")

# scale data
scalar = StandardScaler()
scaled_data = pd.DataFrame(scalar.fit_transform(pd.DataFrame(X))) #
scaling the data
X = PCA(n_components=2).fit_transform(scaled_data)

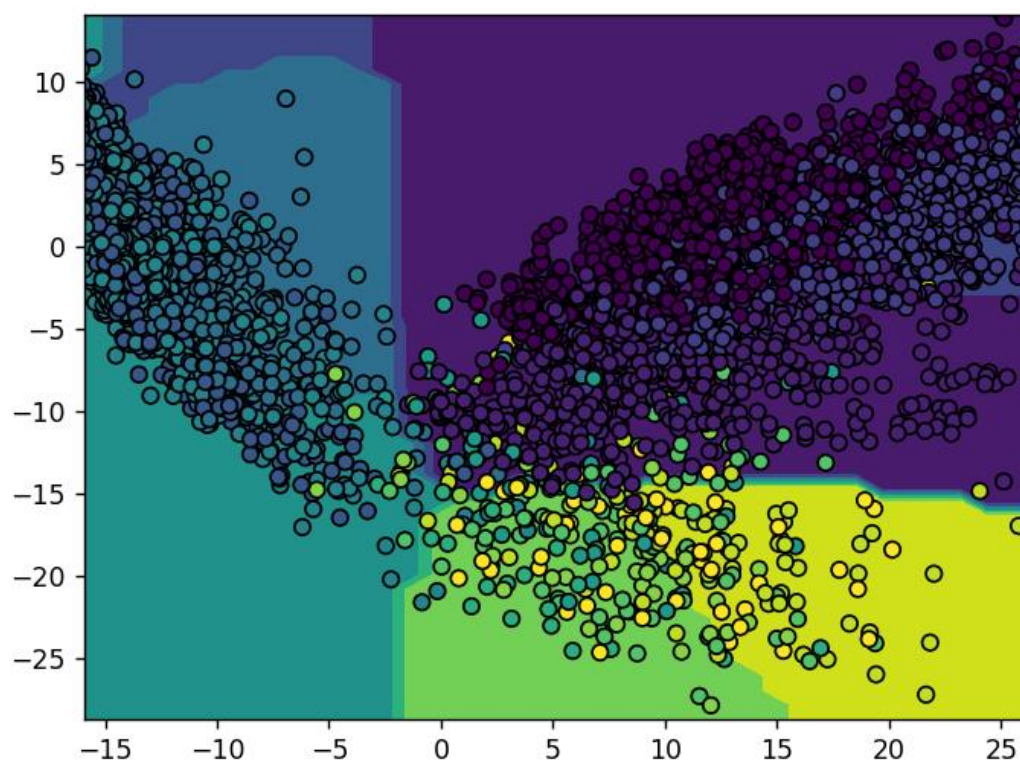
# train model
svm = SVC(probability=True).fit(X, y)

# display trained model
disp = DecisionBoundaryDisplay.from_estimator(
    svm, X, response_method="predict",
)
disp.ax_.scatter(X[:, 0], X[:, 1], c=y, edgecolor="k")
plt.show()
```

Wynikowa graficzna reprezentacja wytrenowanego modelu.



Poniżej znajduje się przybliżenie fragmentu w punkcie, gdzie dzielą się wszystkie klasy.



## 1. Wnioski

Klasyfikacja to zadanie w dziedzinie uczenia maszynowego, które polega na przypisywaniu obiektów lub danych do jednej z wielu klas lub etykiet na podstawie cech i wzorców zawartych w danych. Celem klasyfikacji jest zrozumienie, która klasa lub etykieta najlepiej opisuje dane wejściowe.

Regresja to technika analizy danych i uczenia maszynowego, która ma na celu przewidywanie wartości numerycznych na podstawie innych zmiennych, nazywanych predyktorami. Jako wynik otrzymujemy tutaj wartość ciągłą zamiast konkretnej klasy jak to było w przypadku klasyfikacji.

Algorytm SVM (Support Vector Machine) to technika uczenia maszynowego wykorzystywana zarówno w zadaniach klasyfikacji, jak i regresji. Jego głównym celem jest znalezienie hiperpłaszczyzny (w przypadku klasyfikacji) lub funkcji (w przypadku regresji), która najlepiej oddziela dane na różne klasy lub przewiduje wartości numeryczne.

Powyższy algorytm działa w następujący sposób. Na początku zbierane są dane. Każda próbka danych ma swoje cechy takie jak np. współrzędne. Następnie tworzona jest linia lub płaszczyzna, która dzieli wcześniej przygotowany zbiór na klasy. W kolejnych krokach algorytm linia ta jest przesuwana w taki sposób, aby być jak najdalej od danych z różnych grup.

Algorytm k Nearest Neighbors (kNN) to algorytm w dziedzinie uczenia maszynowego, stosowany tak jak poprzedni w zadaniach klasyfikacji i regresji. Jego działanie opiera się na założeniu, że obiekty o podobnych cechach znajdują się blisko siebie w przestrzeni cech.

Klasyfikator k-najbliższych sąsiadów jest klasyfikatorem leniwym. W tym przypadku oznacza to, że nie tworzy on funkcji dyskryminacyjnej, na podstawie której, będą oceniane próbki w gotowym modelu, tylko zapamiętuje on cały zbiór. Na początku wybierana jest wartość parametru k oznaczająca liczbę najbliższych sąsiadów oraz wybierana jest metryka odległości. Następnie dla każdej próbki testowej znajduwane jest k sąsiadów. Następnie, zgodnie z zasadą demokracji, próbkę testową przypisywana jest klasa, która występuje najczęściej wśród sąsiadów.

Kolejnym algorytmem do klasyfikacji danych jest Decision Tree. W procesie tworzenia modelu budowane jest drzewo binarne. Na początku tworzony jest korzeń, który zawiera cały zbiór. Następnie dzieli się go na dwie części wobec cechy mającej największy przyrost informacji. Proces ten jest powtarzany do uzyskania liści, w których pozostają dane tylko z jednej klasy.

Ostatnim algorytmem testowanym na dzisiejszym laboratorium jest Random Forest. To zaawansowany algorytm uczenia maszynowego, który opiera się na tworzeniu wielu drzew decyzyjnych z losowo wybranymi podzbiorami danych treningowych. Każde drzewo jest nieco inne, co sprawia, że algorytm jest bardziej stabilny i dokładny. W przypadku klasyfikacji, drzewa głosują, aby określić przynależność do klasy, a w przypadku regresji, drzewa przewidują wartości, które są uśredniane, aby uzyskać ostateczną prognozę. Algorytm jest skuteczny w radzeniu sobie z problemami nadmiernej złożoności i jest często używany w różnych zadaniach uczenia maszynowego.