Sieci neuronowe i sztuczna inteligencja – laboratorium 2

17.03.2023

Monika Błyszcz, 236623

ZAD1.

1. Uczenie nadzorowane

Najczęściej wykorzystywane w uczeniu maszynowym. Mamy dane wejściowe (x) i dane wyjściowe (y) i używamy algorytmu do uczenia się funkcji mapowania z wejścia na wyjście y= f(x).

Celem jest tak dokładne przybliżenie funkcji mapowania, aby po uzyskaniu nowych danych wejściowych (x) można było przewidzieć zmienne wyjściowe (y) dla tych danych.

Problemy z nadzorowanym uczeniem można podzielić na problemy:

- Regresji - występuje, gdy zmienna wyjściowa jest wartością rzeczywistą, taką jak "dolary" lub "waga".
- Klasyfikacji - występuje, gdy zmienną wyjściową jest kategoria, na przykład "choroba" i "brak choroby"

Przykłady algorytmów wykorzystujących uczenie nadzorowane:

- **Regresja logistyczna**

Regresja logistyczna jest szybka i stosunkowo nieskomplikowana, jest metodą klasyfikacji binarnej. Celem algorytmu jest znalezienie funkcji regresji logistycznej p(x) takiej, aby przewidywane odpowiedzi p(xi) były jak najbardziej zbliżone do rzeczywistej odpowiedzi yi dla każdej obserwacji *i = [1, ..., n]*. Rzeczywista odpowiedź może wynosić 0 lub 1 (klasyfikacja binarna). Po uzyskaniu funkcji regresji logistycznej można jej użyć do przewidywania wyników dla nowych i niewidocznych danych wejściowych, zakładając, że podstawowa zależność matematyczna pozostaje niezmieniona. Klasyfikacja binarna ma 4 możliwe typy wyników:

- Prawdziwe negatywy: poprawnie przewidywane ujemne (zera)
- Prawdziwe pozytywy: prawidłowo przewidywane pozytywne (jedynki)
- Fałszywie ujemne: nieprawidłowo przewidywane wartości ujemne (zera)
- Fałszywie dodatnie: nieprawidłowo przewidziane wyniki dodatnie (jedynki)

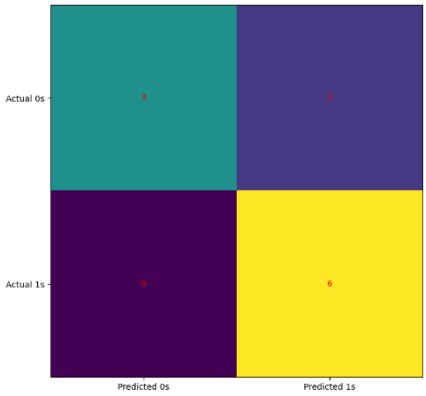
Implementacja regresji wygląda następująco: przygotowujemy dane x i y (tablica dwuwymiarowa), tworzymy obiekt klasy *LogisticRegression*, na którym wywołujemy metodę *fit* podając jej za argumenty zbiór treningowy, docelowe wartości oraz ewentualne parametry. Gdy model się nauczy to możemy wywołać metodę *predict*, która zwróci przewidywaną wartość.

Przygotowano kod wizualizujący regresje logistyczną dla poniższego zestawu danych.

```python
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
#data
x = np.arange(10).reshape(-1, 1)
y = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

#model
model = LogisticRegression(solver='liblinear', random_state=0).fit(x, y)

#attributes of model
cl = model.classes_
inter = model.intercept_
coef = model.coef_
#evaluate model
model.predict_proba(x)
model.predict(x)
score = model.score(x, y)
cm = confusion_matrix(y, model.predict(x))

#heatmap
fig, ax = plt.subplots(figsize=(8, 8))
ax.imshow(cm)
ax.grid(False)
ax.xaxis.set(ticks=(0, 1), ticklabels=('Predicted 0s', 'Predicted 1s'))
ax.yaxis.set(ticks=(0, 1), ticklabels=('Actual 0s', 'Actual 1s'))
ax.set_ylim(1.5, -0.5)
for i in range(2):
    for j in range(2):
        ax.text(j, i, cm[i, j], ha='center', va='center', color='red')
plt.show()

#print values
print(f"binary classification: {cl}")
print(f"intercept: {inter}") #b0
print(f"slope: {coef}") #b1
print(f"predict: {model.predict(x)}") #predict
print(f"accuracy: {score}")
print(f"confusion matrix: {cm}")
print(f"More comprehensive report on the classification: {classification_report(y, model.predict(x))}")#more comprehensive report on the classification
```

Poniżej przedstawiono wyniki regresji logistycznej i mapę cieplną:

```
C:\Users\Mo\AppData\Local\Microsoft\WindowsApps\python3.10.exe
binary classification: [0 1]
intercept: [-1.04608067]
slope: [[0.51491375]]
predict: [0 0 0 1 1 1 1 1 1 1]
accuracy: 0.9
confusion matrix: [[3 1]
 [0 6]]

More comprehensive report on the classification
              precision    recall  f1-score   support

           0       1.00      0.75      0.86         4
           1       0.86      1.00      0.92         6

    accuracy                           0.90        10
   macro avg       0.93      0.88      0.89        10
weighted avg       0.91      0.90      0.90        10


Process finished with exit code 0
```



Uzyskana macierz pokazuje, że mamy:

- Trzy prawdziwe negatywne prognozy: Pierwsze trzy obserwacje to zera przewidziane poprawnie.
- Brak fałszywych negatywnych prognoz: Są to te, które błędnie przewidywano jako zera.
- Jedna fałszywie pozytywna prognoza: Czwarta obserwacja to zero, które zostało błędnie przewidziane jako jedno.
- Sześć prawdziwie pozytywnych prognoz: Ostatnie sześć obserwacji to obserwacje przewidziane poprawnie.

Dodatkowo, powyższa macierz została zwizualizowana za pomocą heatmapy w celu łatwiejszej interpretacji.

- **Drzewo decyzyjne**

Drzewa decyzyjnego używa do klasyfikacji danych, czyli do przypisania obserwacji zbioru danych do jednej z klas. Na każdej gałęzi (węźle) drzewa dokonujemy podziału na 2 i więcej mniejszych zbiorów, by jak najlepiej odseparować od siebie klasy. Budowę drzewa decyzyjnego zatrzymujemy gdy nasz zbiór osiągnął minimalną liczbę obserwacji (np. na liściu znajduje się mniej niż 5% wszystkich

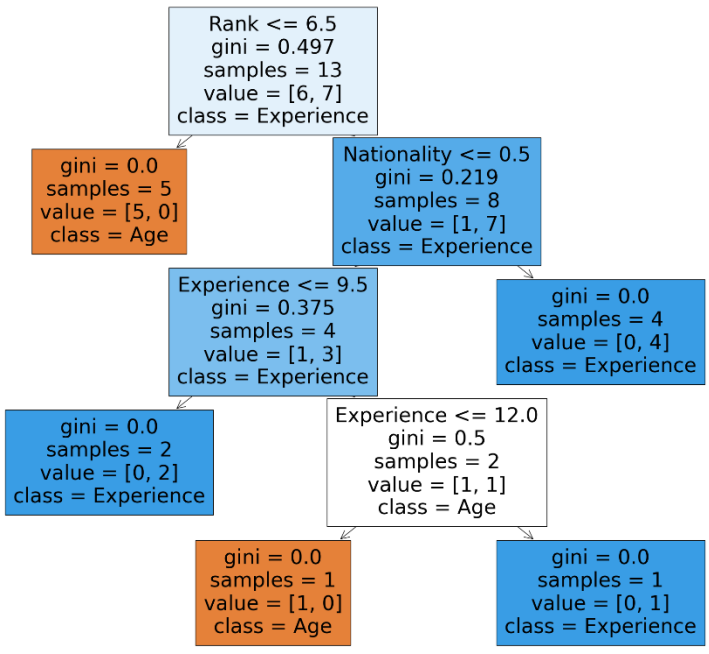
obserwacji) lub zbiór jest czysty (zawiera tylko jedną obserwację). Aby uruchomić drzewo decyzyjne, wszystko dane muszą być liczbowe.

W poniższym kodzie osoba będzie próbowała zdecydować czy powinna pójść na komedię czy nie. Posiadamy zestaw danych, w którym posiadamy informacje czy ta osoba poszła (czy też nie) na komedię, która była grana w mieście. Znamy też komika, który wtedy występował. Czyli znamy pewne preferencje tej osoby.

```
import pandas
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt

df = pandas.read_csv("data.csv") #Load dataset

#Converting values to numbers (Decision Tree Requirement)
d = {'UK': 0, 'USA': 1, 'N': 2} # Means convert the values 'UK' to 0, 'USA' to 1, and 'N' to 2.
df['Nationality'] = df['Nationality'].map(d)
d = {'YES': 1, 'NO': 0}
df['Go'] = df['Go'].map(d)

#Separation of the feature column (x) from the target column (y)
features = ['Age', 'Experience', 'Rank', 'Nationality', 'Go']

array = df.values
# The feature columns are the columns that we try to predict from, and the target column is the column with the values we try to predict.
X = array[:, :-1]
Y = array[:, -1]

dtree = DecisionTreeClassifier()
dtree = dtree.fit(X, Y)

fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(dtree,
                   feature_names=df.columns,
                   class_names=features,
                   filled=True)
fig.savefig("decision_tree_2.png")


#Use predict() method to predict new values - #What would the answer be if the comedy rank was 6?
print( f"Predict:  {dtree.predict([[40, 10, 6, 1]])}")

print("[1] means 'GO'")
print("[0] means 'NO'")
```

Drzewo decyzyjne daje różne wyniki, jeśli zostanie uruchomione wystarczająco dużo razy, nawet jeśli zasilamy je tymi samymi danymi. Dlatego, że drzewo decyzyjne nie daje nam stuprocentowej pewności. Opiera się na prawdopodobieństwie wyniku, a odpowiedź będzie się różnić. Wyniki można interpretować następująco:

- Rank <= 6,5 – komik o randze 5,6 i niżej będzie podążał w lewo (true)
- Gini = 0,487 – jakość spilt (liczba z zakresu [0,0; 0,5], gdzie 0,0 oznacza, że wszystkie próbki otrzymały ten sam wynik, a 0,5 że podział został zakończony dokładnie na pośrodku)
- Samples = 13 – jest 13 komików, czyli 13 próbek poddawanych analizie
- Value = [6,7] – 6 „NO”, 7 „GO” jako NO- nie idź, GO- idź na tą komedię



Możemy przewidywać wyniki np. *Jaka byłaby odpowiedź, gdyby ranga komedii wynosiła 6?* Otrzymujemy, że prawdopodobnie by nie poszła na tą komedię.

```
C:\Users\Mo\AppData\Local\Microsoft\WindowsApps\python3.10.
Predict:  [0]
[1] means 'GO'
[0] means 'NO'

Process finished with exit code 0
```

2. Uczenie nienadzorowane

Mamy tylko dane wejściowe (X) i nie posiadamy odpowiadających im danych wyjściowych. Celem takiego uczenia jest modelowanie podstawowej struktury lub rozkładu danych, aby dowiedzieć się więcej o danych. Problemy z nienadzorowanym uczeniem można podzielić na problemy:

- Asocjacji, które polega na znalezieniu reguł opisujących duże części danych, szukania zależności np. osoby, które kupują X, również mają tendencję do kupowania Y
- Klastrowania, które polega na tym, że chcemy znaleźć nieodłączne grupy w danych np. grupowanie klientów według zachowań zakupowych

- **Algorytm Apriori**

Algorytm Apriori to model uczenia maszynowego używany w uczeniu reguł asocjacyjnych do identyfikowania częstych zestawów elementów ze zbioru danych. Wykorzystuje:trzy macierze do znalezienia najlepszych reguł asocjacyjnych ze zbioru danych, dzięki czemu jego podejście do zbiorów danych jest skuteczne. Macierze obejmują:

- **support**: prawdopodobieństwo że określona transakcja zawiera X i jak Y (jak popularny jest dany zestaw towarów)
- **confidence**: prawdopodobieństwo zakupu przedmiotu Y przy zakupie przedmiotu X, wyrażonym jako {X -> Y}
- **lift**: prawdopodobieństwo zakupu przedmiotu Y gdy przedmiot X zostanie zakupiony (kontrolując jednocześnie popularność Y)

Model ten został szeroko zastosowany w zestawach danych transakcji przez dużych sprzedawców detalicznych w celu określenia towarów, które klienci często kupują razem z dużym prawdopodobieństwem. Np. gdy klienci kupują komputer, to kupują też oprogramowanie. Można zapisać to za pomocą takiej formuły:

computer => finacial_management_software, [support = 5%,confidence = 75%]

W poniższym kodzie analizowano zestaw danych transakcji klienta zawierający zapisy transakcji klientów z konkretną firmą w danym tygodniu. Celem algorytmy było poznanie najlepszej reguły skojarzenia z tego zestawu danych i zwrócenie tej reguły właścicielowi firmy. Korzystając z tej reguły, właściciel firmy może teraz oferować klientom pewne oferty, co zwiększy sprzedaż i zysk

```
import pandas as pd
from apyori import apriori
data = pd.read_csv("Market_Basket_Optimisation.csv")

# Intiglizing the list
transacts = []
# populating a list of transactions
for i in range(0, 7500):
  transacts.append([str(data.values[i,j]) for j in range(0, 20)])

rule = apriori(transactions = transacts, min_support = 0.003, min_confidence = 0.2, min_lift = 3, min_length = 2, max_length = 2)

output = list(rule) # returns a non-tabular output
# putting output into a pandas dataframe
new *
def inspect(output):
    lhs         = [tuple(result[2][0][0])[0] for result in output]
    rhs         = [tuple(result[2][0][1])[0] for result in output]
    support     = [result[1] for result in output]
    confidence = [result[2][0][2] for result in output]
    lift        = [result[2][0][3] for result in output]
    return list(zip(lhs, rhs, support, confidence, lift))


output_DataFrame = pd.DataFrame(inspect(output), columns = ['Left_Hand_Side', 'Right_Hand_Side', 'Support', 'Confidence', 'Lift'])
print(output_DataFrame)

print(output_DataFrame.nlargest(n = 10, columns = 'Lift'))
```

Wyniki nieposortowane:

```
C:\Users\Mo\AppData\Local\Microsoft\WindowsApps\python3.10.exe C:\Users\Mo\
        Left_Hand_Side Right_Hand_Side   Support  Confidence      Lift
0          light cream         chicken  0.004533    0.290598  4.843305
1  mushroom cream sauce        escalope  0.005733    0.300699  3.790327
2                pasta        escalope  0.005867    0.372881  4.700185
3         fromage blanc           honey  0.003333    0.245098  5.178128
4         herb & pepper     ground beef  0.016000    0.323450  3.291555
5          tomato sauce     ground beef  0.005333    0.377358  3.840147
6          light cream        olive oil  0.003200    0.205128  3.120612
7     whole wheat pasta        olive oil  0.008000    0.271493  4.130221
8                pasta          shrimp  0.005067    0.322034  4.514494
```

Wyniki posortowane wg malejącej kolejności kolumny

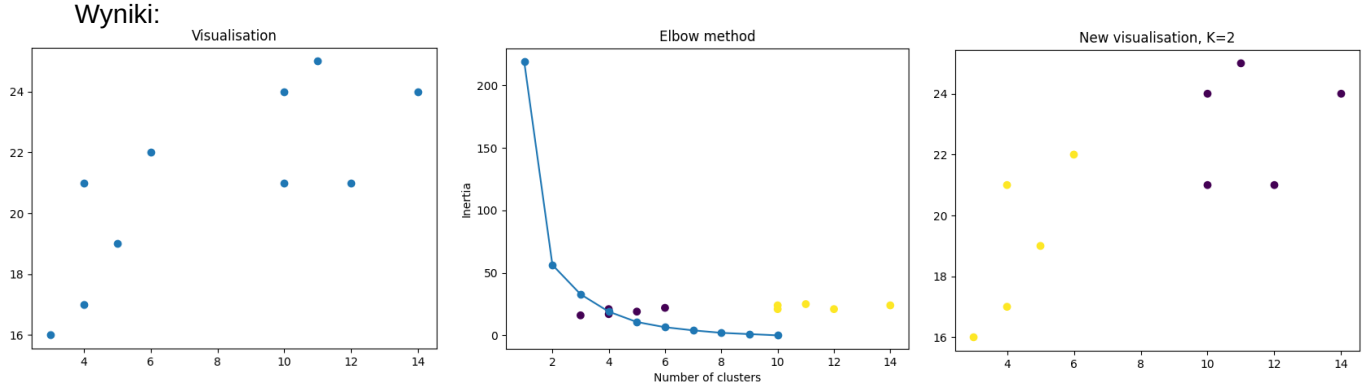
```
        Left_Hand_Side Right_Hand_Side   Support  Confidence      Lift
3         fromage blanc           honey  0.003333    0.245098  5.178128
0           light cream         chicken  0.004533    0.290598  4.843305
2                 pasta        escalope  0.005867    0.372881  4.700185
8                 pasta          shrimp  0.005067    0.322034  4.514494
7     whole wheat pasta        olive oil  0.008000    0.271493  4.130221
5           tomato sauce     ground beef  0.005333    0.377358  3.840147
1  mushroom cream sauce        escalope  0.005733    0.300699  3.790327
4         herb & pepper     ground beef  0.016000    0.323450  3.291555
6           light cream        olive oil  0.003200    0.205128  3.120612
```

K-means to nienadzorowana metoda uczenia się służąca do grupowania punktów danych. Algorytm iteracyjnie dzieli punkty danych na klastry K, minimalizując wariancję w każdym klastrze. Schemat działania: każdy punkt danych jest losowo przypisywany do jednego z klastrów K. Następnie obliczamy centroid (funkcjonalnie centrum) każdego klastra i ponownie przypisujemy każdy punkt danych do klastra z najbliższym centroidem. Powtarzamy ten proces, dopóki przypisania klastra dla każdego punktu danych nie będą się zmieniać. K-oznacza, że grupowanie wymaga wybrania K, liczby klastrów, w które chcemy pogrupować dane. Dobrą metodą dla określenia wartości K jest elbow, który pozwala wykreślić bezwładność (metrykę opartą na odległości) i wizualizować punkt (tzw elbow), w którym zaczyna ona maleć liniowo. Pozwala to na oszacowanie najlepszej wartości K dla naszych danych.

Poniżej kod implementujący K-means

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

plt.scatter(x, y)
plt.title('Visualisation')
plt.show()
#Now we utilize the elbow method to visualize the intertia for different values of K:
data = list(zip(x, y)) #list of data
inertias = []

for i in range(1,11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)

plt.plot(range(1,11), inertias, marker='o')
plt.title('Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')

#We can see that the "elbow" on the graph above (where the interia becomes more linear) is at K=2.
# We can then fit our K-means algorithm one more time and plot the different clusters assigned to the data:
kmeans = KMeans(n_clusters=2)
kmeans.fit(data)

plt.scatter(x, y, c=kmeans.labels_)
plt.title('New visualisation, K=2')
plt.show()
```

Wyniki:



Po wyznaczeniu K za pomocą metody Elbow możemy zauważyć, że istnieją 2 klastry danych, co zostało przedstawione na ostatnim wykresie.

3. Uczenie częściowo nadzorowane

Mamy dużo danych wejściowych (X) i częściowo znane dane wyjściowe (Y). Te problemy znajdują się pośrednio pomiędzy nadzorowanym i nienadzorowanym uczeniem się. Np. możemy mieć archiwum zdjęć, w którym tylko część zdjęć jest oznaczona (pies/kot/osoba) a reszta nieoznaczona

- **Algorytm rozprzestrzeniania się etykiet (Label Spreading)**

Algorytm jest inspirowany techniką z psychologii eksperymentalnej zwaną rozprzestrzenianiem sieci aktywacji. Algorytm rozprzestrzeniania etykiet jest dostępny w bibliotece uczenia maszynowego Pythona scikit-learn za pośrednictwem klasy *LabelSpreading*. Model może być dopasowany tak samo jak każdy inny model klasyfikacji, wywołując funkcję fit() i używany do przewidywania nowych danych za pomocą funkcji *predict*(). Co ważne, zestaw danych uczenia dostarczany do funkcji fit() musi zawierać przykłady z etykietami, które są zakodowane porządkowo (zgodnie z normalnym) i przykłady bez etykiet oznaczone etykietą -1. Następnie model określi etykietę dla nieoznaczonych przykładów w ramach dopasowania modelu. Po dopasowaniu modelu szacowane etykiety dla oznaczonych i nieoznakowanych danych w zestawie danych uczenia są dostępne za pośrednictwem atrybutu *"transduction_"* w klasie *LabelSpreading*.

```
from numpy import concatenate
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.semi_supervised import LabelSpreading

# prepare semi-supervised learning dataset
X, y = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=0, random_state=1)
# split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.50, random_state=1, stratify=y)
# split train into labeled and unlabeled
X_train_lab, X_test_unlab, y_train_lab, y_test_unlab = train_test_split(X_train, y_train, test_size=0.50, random_state=1, stratify=y_train)
# summarize training set size
print('Labeled Train Set:', X_train_lab.shape, y_train_lab.shape)
print('Unlabeled Train Set:', X_test_unlab.shape, y_test_unlab.shape)
# summarize test set size
print('Test Set:', X_test.shape, y_test.shape)

# create the training dataset input
X_train_mixed = concatenate((X_train_lab, X_test_unlab))
# create "no label" for unlabeled data
nolabel = [-1 for _ in range(len(y_test_unlab))]
# recombine training dataset labels
y_train_mixed = concatenate((y_train_lab, nolabel))
# define model
model = LabelSpreading()
# fit model on training dataset
model.fit(X_train_mixed, y_train_mixed)
# make predictions on hold out test set
yhat = model.predict(X_test)
# calculate score for test set
score = accuracy_score(y_test, yhat)
# summarize score
print('Accuracy: %.3f' % (score*100))
```

Wyniki pokazują nam, że mamy zestaw danych testowych składający się z 500 wierszy, oznaczony zestaw danych treningowych z 250 wierszami i 250 wierszy nieoznaczonych danych. Nadzorowany algorytm uczenia będzie składał się 250 wierszy, z których można wytrenować model. Częściowo nadzorowany algorytm uczenia będzie miał 250 oznaczonych wierszy, a także 250 nieoznaczonych wierszy, które można wykorzystać na wiele sposobów do ulepszenia oznaczonego zestawu danych treningowych.

```
C:\Users\Mo\AppData\Local\Microsoft\WindowsApps\python3.10.exe
Labeled Train Set: (250, 2) (250,)
Unlabeled Train Set: (250, 2) (250,)
Test Set: (500, 2) (500,)
Accuracy: 85.400

Process finished with exit code 0
```

Dokładność uczenia częściowo nadzorowanego wynosi 85,400. Można opracować model uczenia nadzorowanego by porównać dokładność wyników.

- **Algorytm samokształcący się (Self Training Classifier)**

Działanie algorytmu wygląda następująco:

1. Zbieramy wszystkie dane oznaczone i nieoznakowane, ale używamy tylko oznaczonych do trenowania pierwszego nadzorowanego modelu.
2. Następnie używamy tego modelu do przewidywania klasy nieoznaczonych danych.
3. Wybieramy obserwacje, które spełniają nasze wstępnie zdefiniowane kryteria (np. prawdopodobieństwo przewidywania wynosi >90% lub należy do 10 najlepszych obserwacji o

najwyższym prawdopodobieństwie przewidywania) i łączymy te pseudoetykiety z oznaczonymi danymi.

4. Powtarzamy ten proces, trenując nowy nadzorowany model za pomocą obserwacji z etykietami i pseudoetykietami. Następnie ponownie dokonujemy prognoz i dodajemy nowo wybrane obserwacje do pseudo-oznaczonej puli.
5. Powtarzamy te kroki, aż zakończymy etykietowanie wszystkich danych, żadne dodatkowe nieoznaczone obserwacje nie spełnią naszych kryteriów pseudoetykietowania lub osiągniemy określoną maksymalną liczbę iteracji.

```python
import pandas as pd
from sklearn.model_selection import train_test_split # for splitting data into train and test samples
from sklearn.svm import SVC # for Support Vector Classification baseline model
from sklearn.semi_supervised import SelfTrainingClassifier # for Semi-Supervised learning
from sklearn.metrics import classification_report # for model evaluation metrics
# Read in data
df = pd.read_csv('marketing_campaign.csv', encoding='utf-8', delimiter=';',usecols=['ID', 'Year_Birth', 'Marital_Status', 'Income', 'Kidhome', 'Teenhome', 'MntWines', 'MntMeatProducts'])

#We will try to predict if our supermarket customer has dependents (children/teenagers) at home or not. Create a flag to denote whether the person has any dependants at home (either kids or teens)
df['Dependents_Flag']=df.apply(lambda x: 1 if x['Kidhome']+x['Teenhome']>0 else 0, axis=1)

#Test data will be used to evaluate model performance, while labeled and unlabeled data will be used to train our models.
df_train, df_test = train_test_split(df, test_size=0.25, random_state=0)
print('Size of train dataframe: ', df_train.shape[0])
print('Size of test dataframe: ', df_test.shape[0])

# Create a flag for label masking
df_train['Random_Mask'] = True
df_train.loc[df_train.sample(frac=0.05, random_state=0).index, 'Random_Mask'] = False

# Create a new target colum with labels. The 1's and 0's are original labels and -1 represents unlabeled (masked) data
df_train['Dependents_Target']=df_train.apply(lambda x: x['Dependents_Flag'] if x['Random_Mask']==False else -1, axis=1)

# Show target value distribution
print('Target Value Distribution:')
print(df_train['Dependents_Target'].value_counts())

#Basic Data Prep
# Select only records with known labels
df_train_labeled=df_train[df_train['Dependents_Target']!=-1]
# Select data for modeling
X_baseline=df_train_labeled[['MntMeatProducts', 'MntWines']]
y_baseline=df_train_labeled['Dependents_Target'].values
# Put test data into an array
X_test=df_test[['MntMeatProducts', 'MntWines']]
y_test=df_test['Dependents_Flag'].values

X_train=df_train[['MntMeatProducts', 'MntWines']]
y_train=df_train['Dependents_Target'].values

#Model Fitting (2) - Specify SVC model parameters
model_svc = SVC(kernel='rbf',
                probability=True, # Need to enable to be able to use predict_proba
                C=1.0, # default = 1.0
                gamma='scale', # default = 'scale',
                random_state=0)
# Specify Self-Training model parameters
self_training_model = SelfTrainingClassifier(base_estimator=model_svc, # An estimator object implementing fit and predict_proba.
                                             threshold=0.7, # default=0.75, The decision threshold for use with criterion='threshold'. Should be in [0, 1).
                                             criterion='threshold', # {'threshold', 'k_best'}, default='threshold',
                                             # The selection criterion used to select which labels to add to the training set. If 'threshold',
                                             # pseudo-labels with prediction probabilities above threshold are added to the dataset. If 'k_best',
                                             # the k_best pseudo-labels with highest prediction probabilities are added to the dataset.
                                             #k_best=50, # default=10, The amount of samples to add in each iteration. Only used when criterion='k_best'.
                                             max_iter=100, # default=10, Maximum number of iterations allowed. Should be greater than or equal to 0.
                                             # If it is None, the classifier will continue to predict labels until no new pseudo-labels are added, or all unlabeled samples have been labeled.
                                             verbose=True) # default=False, Verbosity prints some information after each iteration

# Fit the model
clf_ST = self_training_model.fit(X_train, y_train)
#Model Evaluation (3)
print('')
print('-------- Self Training Model - Summary --------')
print('Base Estimator: ', clf_ST.base_estimator_)
print('Classes: ', clf_ST.classes_)
print('Transduction Labels: ', clf_ST.transduction_)
print('Iteration When Sample Was Labeled: ', clf_ST.labeled_iter_)
print('Number of Features: ', clf_ST.n_features_in_)
print('Feature Names: ', clf_ST.feature_names_in_)
print('Number of Iterations: ', clf_ST.n_iter_)
print('Termination Condition: ', clf_ST.termination_condition_)
print('')
print('------- Self Training Model - Evaluation on Test Data -----')
accuracy_score_ST = clf_ST.score(X_test, y_test)
print('Accuracy Score %3f:' % (accuracy_score_ST*100))
# Look at classification report to evaluate the model
print(classification_report(y_test, clf_ST.predict(X_test)))
```

Wynik:

```
C:\Users\Mo\AppData\Local\Microsoft\WindowsApps\python3.10.exe
Size of train dataframe:  1680
Size of test dataframe:  560
Target Value Distribution:
-1    1596
 1      58
 0      26
Name: Dependents_Target, dtype: int64
End of iteration 1, added 1432 new labels.
End of iteration 2, added 127 new labels.
End of iteration 3, added 16 new labels.
End of iteration 4, added 3 new labels.

-------- Self Training Model - Summary --------
Base Estimator:  SVC(probability=True, random_state=0)
Classes:  [0 1]
Transduction Labels:  [0 1 1 ... 1 1 0]
Iteration When Sample Was Labeled:  [1 2 1 ... 1 1 3]
Number of Features:  2
Feature Names:  ['MntMeatProducts' 'MntWines']
Number of Iterations:  5
Termination Condition:  no_change

------- Self Training Model - Evaluation on Test Data -----
Accuracy Score 83.571429:
              precision    recall  f1-score   support

           0       0.80      0.54      0.64       154
           1       0.84      0.95      0.89       406

    accuracy                           0.84       560
   macro avg       0.82      0.74      0.77       560
weighted avg       0.83      0.84      0.82       560


Process finished with exit code 0
```

Wynik:

W uczeniu częściowo nadzorowanym uzyskujemy dokładność 83,57. Można opracować dane sposobem uczenia nadzorowanego by porównać dokładność. Prawdopodobnie będzie niższa.

ZAD2.

Do wykonania zadania wybrano bazę danych kwiatów. Oceniono nas

- Regresję logistyczną (LR)
- Drzewa klasyfikacji i regresu (CART).

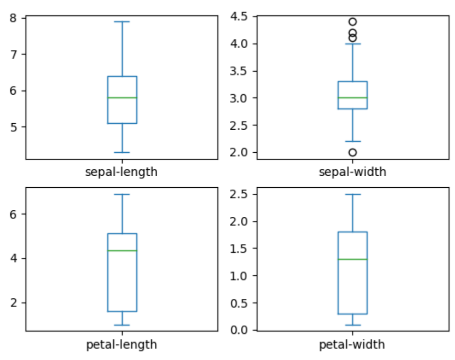
Powyższe algorytmy wchodzą w skład uczenia nadzorowanego. Kod oceniający jakość modeli wygląda następująco:

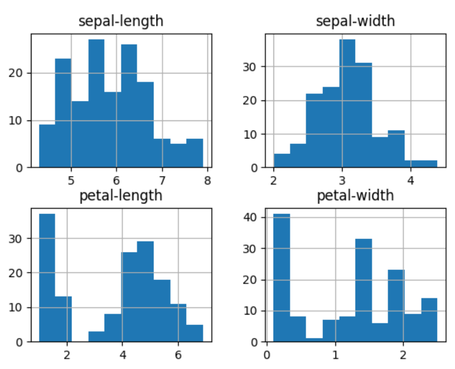
```
import pandas
from sklearn.metrics import classification_report, confusion_matrix
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
#Load dataset
url="https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names=["sepal-length", "sepal-width", "petal-length", "petal-width", "class"]
dataset = pandas.read_csv(url, names=names)

dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False) #Box and whisher plots
plt.show()
dataset.hist() #Histograms
plt.show()
scatter_matrix(dataset) #Scatter plot matrix
plt.show()

#Split-out validation dataset
array = dataset.values
X = array[:, :-1]
Y = array[:, -1]
validation_size =0.20
seed = 7
X_train, X_validation, Y_train, Y_validation = model_selection.train_test_split(X, Y, test_size=validation_size, random_state=seed)
scoring = 'accuracy'

#Spot Check Algorithms
models =[]
models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ovr'))) #Logistic Regression
models.append(('CART', DecisionTreeClassifier())) #Decision Tree

#Logistic Regression#---------------------------------------------
modelLR = models[0][1].fit(X, Y) #model
cl = modelLR.classes_
inter = modelLR.intercept_
coef = modelLR.coef_
#evaluate model
modelLR.predict_proba(X)
modelLR.predict(X)
score = modelLR.score(X, Y)
cm = confusion_matrix(Y, modelLR.predict(X))
#heatmap
fig, ax = plt.subplots(figsize=(8, 8))
ax.imshow(cm)
ax.grid(False)
ax.xaxis.set(ticks=(0, 1), ticklabels=('Predicted 0s', 'Predicted 1s'))
ax.yaxis.set(ticks=(0, 1), ticklabels=('Actual 0s', 'Actual 1s'))
ax.set_ylim(1.5, -0.5)
for i in range(2):
    for j in range(2):
        ax.text(j, i, cm[i, j], ha='center', va='center', color='red')
plt.show()
print('----------------------------------------------')
print('LOGISTIC REGRESSION')
print(' ')
print(f"binary classification: {cl}")
print(f"intercept: {inter}") #b0
print(f"slope: {coef}") #b1
print(f"predict: {modelLR.predict(X)}") #predict
print(f"accuracy: {score:f}")
print(f"confusion matrix: {cm}")
print(f"More comprehensive report on the classification: {classification_report(Y, modelLR.predict(X))}")#more comprehensive report on the classification
print('----------------------------------------------')
#Decision Tree #--------------------------------------
modelDT = models[1][1].fit(X, Y)
fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(modelDT,
                   feature_names=dataset.columns,
                   class_names=names,
                   filled=True)
fig.savefig("decision_tree.png")

#Evaluate each model in turn ------------------------------------
results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=seed, shuffle=True)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg ="%s: %f (%f)" % (name, cv_results.mean(),cv_results.std())
    print(msg)

#Compare Algorithms
fig =plt.figure()
fig.suptitle('Algorithm Comparison')
ax =fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

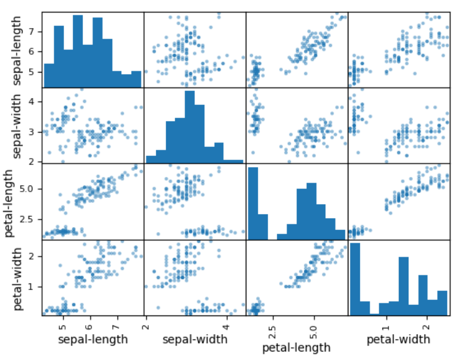
```
LOGISTIC REGRESSION ---------------------------------------
binary classification: ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
intercept: [ 0.26560617  1.08542374 -1.21471458]
slope: [[ 0.41498833  1.46129739 -2.26214118 -1.0290951 ]
 [ 0.41663969 -1.60083319  0.57765763 -1.38553843]
 [-1.70752515 -1.53426834  2.47097168  2.55538211]]
predict: ['Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-virginica' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-virginica' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-versicolor'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica']
accuracy: 0.960000
confusion matrix: [[50  0  0]
 [ 0 45  5]
 [ 0  1 49]]
More comprehensive report on the classification:                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        50
Iris-versicolor       0.98      0.90      0.94        50
 Iris-virginica       0.91      0.98      0.94        50

       accuracy                           0.96       150
      macro avg       0.96      0.96      0.96       150
   weighted avg       0.96      0.96      0.96       150

LR: 0.958333 (0.055902)
CART: 0.958333 (0.076830)
```

Oba algorytmy uzyskały identyczną dokładność. Drzewa decyzyjne są dosyć dokładne, ale kosztem dłuższego czasu uczenia się. Dla porównania, regresja logistyczna charakteryzuje się niskim czasem uczenia się przy nieco gorszej dokładności.

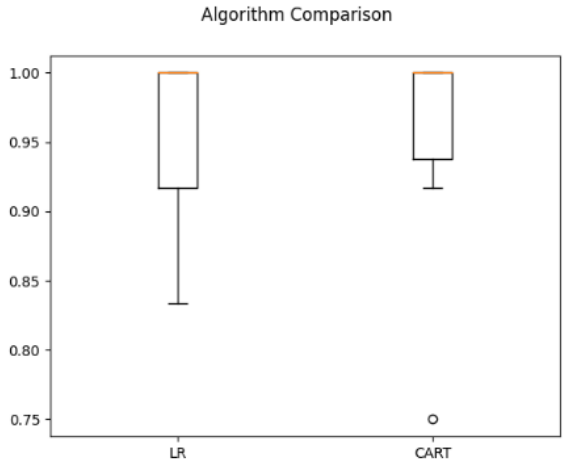
Poniżej przedstawiono wykresy pudełkowe, by zobaczyć jaki jest rozkład atrybutów wejściowych dla każdego kwiatu. Obok tych wykresów znajdują się histogramy, część z nich ma rozkład Gaussa. Wykonano tez wykres rozrzutu par atrybutów – ujawniają się wysokie korelacje.

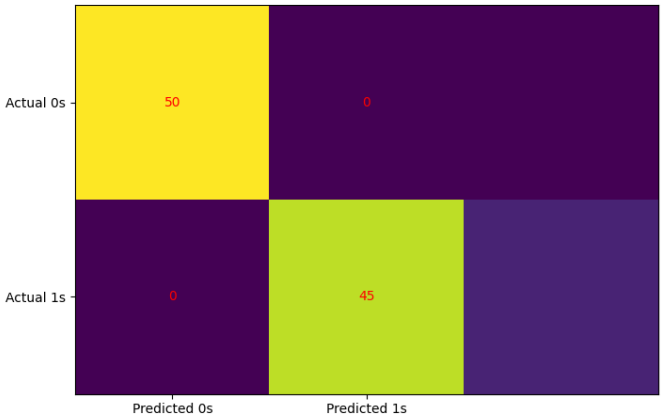






Poniżej przedstawiono mapę cieplną w celu lepszej wizualizacji wyników macierzy z regresji logistycznej. Analizując uzyskane wykresy pudełkowe możemy zauważyć, że są zgniecione w górnej części zakresu, a obie próbki uzyskują niemal 100% dokładności.





Poniżej przedstawiono drzewo decyzyjne.