შეხვედრა 11: პირველი ML მოდელის აგება: Scikit-learn

წინა შეხვედრებზე ჩვენ ვისწავლეთ, როგორ გავამზადოთ ტექსტური მონაცემები და გავეცანით მანქანური სწავლების თეორიულ საფუძვლებს. დღეს კი დროა, დავიწყოთ ჩვენი პირველი, სრულფასოვანი მანქანური სწავლების მოდელის აგება. ჩვენ შევქმნით მოდელს, რომელიც შეძლებს ახალი, უცნობი წინადადებებიდან განზრახვის ამოცნობას. ამისთვის გამოვიყენებთ ერთ-ერთ ყველაზე პოპულარულ და მძლავრ ბიბლიოთეკას - **Scikit-learn-ს**. ამ შეხვედრის ბოლოს შენ შეძლებ, ააგო და გაწვრთნა შენი პირველი მანქანური სწავლების მოდელი. მოდელი, რომელსაც შეეძლება მომხმარებლის მიერ დაწერილი ტექსტის გაანალიზება, ამ ტექსტიდან მომხმარებლის განზრახვის ამოცნობა.

1. Scikit-learn-ის ბიბლიოთეკა

Scikit-learn-ი არის Python-ის ღია ბიბლიოთეკა, რომელიც მანქანური სწავლების მოდელების შექმნის, გაწვრთნისა და შეფასებისთვის არის განკუთვნილი. ის 2007 წელს შეიქმნაGoogle Summer of Code-ის პროექტის ფარგლებში და საკმაოდ მოკლე დროში გახდა ერთ-ერთი ყველაზე პოპულარული და ფუნდამენტური ბიბლიოთეკა მონაცემთა მეცნიერებაში. მისი მთავარი მიზანია, მანქანური სწავლება გახადოს ხელმისაწვდომი ყველა დონის მომხმარებლისთვის, როგორც დამწყებთათვის ასევე პროფესიონალებისთვის. Scikit-learn-ს შეუძლია გადაჭრას ისეთი ამოცანები, როგორიცაა კლასიფიკაცია, რეგრესია, კლასტერიზაცია და სხვა.

1.1. Scikit-learn-ის ძირითადი შესაძლებლობები

Scikit-learn-ის ერთ-ერთი უდიდესი უპირატესობა მისი მრავალფუნქციურობაა. ის გვაძლევს საშუალებას, მარტივად გამოვიყენოთ სხვადასხვა ტიპის მანქანური სწავლების ალგორითმები.

1.1.1. კლასიფიკაცია (Classification)

კლასიფიკაციას ჩვენ წინა თავებში გავეცანით. ეს არის მონაცემების კატეგორიებად დაყოფის პროცესი. მაგალითად, სპამის ფილტრი აკლასიფიცირებს შემოსულ წერილს „სპამად“ ან „არა-სპამად“. სურათის ამომცნობი მოდელი კი ამოიცნობს, არის თუ არა გამოსახულებაზე „კატა“, „ძაღლი“ ან „ჩიტი“. ამ შეხვედრაში ჩვენ სწორედ კლასიფიკაციის მოდელს შევქმნით, რომელიც მომხმარებლის შეტყობინებების განზრახვას ამოიცნობს.

1.1.2. რეგრესია (Regression)

რეგრესია გამოიყენება რიცხვითი მნიშვნელობების პროგნოზირებისთვის. მაგალითად, რეგრესიის მოდელი შეიძლება გამოვიყენოთ სახლის ფასის, საფონდო ბირჟის ცვლილების ან ტემპერატურის პროგნოზის დროს. მოდელი წარსულ მონაცემებზე დაყრდნობით სწავლობს და ცდილობს იწინასწარმეტყველოს მომავალი მნიშვნელობა.

1.1.3. კლასტერიზაცია (Clustering)

კლასტერიზაცია არის მონაცემების ჯგუფებად დაყოფის პროცესი წინასწარ განსაზღვრული კატეგორიების გარეშე. ამ შემთხვევაში, მოდელი თავად ეძებს მსგავსებებს მონაცემებს შორის და ქმნის ჯგუფებს. მაგალითად, ონლაინ მაღაზიამ შეიძლება გამოიყენოს კლასტერიზაცია, რათა მომხმარებლები მათი შესყიდვის ისტორიის მიხედვით დააჯგუფოს. ამის შემდეგ, მაღაზიას შეეძლება თითოეულ ჯგუფს ინდივიდუალური რეკლამა გაუგზავნოს.

1.2. მოდელის შექმნის 4 ძირითადი ეტაპი

Scikit-learn-ში მოდელის შექმნა, გაწვრთნა და გამოყენება ყოველთვის ოთხი მარტივი ეტაპისგან შედგება:

1. **იმპორტი (Import):** შემოგვაქვს საჭირო ხელსაწყო ჩვენს პროგრამაში.
2. **ინიციალიზაცია (Initialize):** ვახორციელებთ მის ინიციალიზაციას, ანუ ვამზადებთ მას სამუშაოდ.
3. **ვარჯიში (Train/Fit):** ვავარჯიშებთ მოდელს მონაცემებზე.
4. **პროგნოზირება (Predict):** ვეუბნებით მოდელს, რომ ახალი მონაცემების მიღებისას გამოიყენოს ის ცოდნა, რომელიც ვარჯიშის დროს მიიღო და დაგვიბრუნოს სწორი პასუხი. ან ვარჯიშისას მიღებული ცოდნის საფუძველზე შემოგვთავაზოს პროგნოზი.

2. ტექსტის ვექტორიზაცია

როგორც წინა შეხვედრებზე ვისაუბრეთ, მანქანური სწავლების მოდელებს არ შეუძლიათ ტექსტური ინფორმაციის დამუშავება, მათ მხოლოდ რიცხვებთან მუშაობა შეუძლიათ. ამიტომ, სანამ მოდელს გავწვრთნით, ტექსტი რიცხვების ვექტორად უნდა გადავაქციოთ. ამ პროცესს **ვექტორიზაცია** ეწოდება. ვექტორიზაცია არის ტექსტის ციფრულ ფორმატში გადაყვანის პროცესი, სადაც ყოველი სიტყვა ან წინადადება წარმოდგენილია რიცხვების სიის, ანუ ვექტორის სახით. ვექტორიზაციის შეთხვევაში თითოულ სიტყვას უბრალოდ რიცხვით მნიშვნელობას კი არ ვანიჭებთ, არამედ – რიცხვით კოორდინატებს. ანუ თითოეულ სიტყვას ვანიჭებთ უნიკალურ კოორდინატებს. ამგვარად, ვაქცევთ ადამიანის ბუნებრივ ენას კოორდინატებად.

2.1. CountVectorizer: ტექსტების გადაქცევა რიცხვების ვექტორებად

CountVectorizer არის Scikit-learn-ის ინსტრუმენტი, რომელიც ტექსტს რიცხვების სიებად (ვექტორებად) გარდაქმნის. ის მუშაობს **Bag-of-Words** მოდელის პრინციპით, რომელსაც უკვე გავეცანით. ეს პროცესი ორ ნაბიჯს მოიცავს:

- **ნაბიჯი 1:** აანალიზებს ყველა წინადადებას და ქმნის ყველა უნიკალური სიტყვის ლექსიკონს.
- **ნაბიჯი 2:** თითოეულ წინადადებაში ითვლის, რამდენჯერ გეხვდება ეს სიტყვები.

```
# CountVectorizer-ის იმპორტი
from sklearn.feature_extraction.text import CountVectorizer

# მონაცემები
sentences = ["გამარჯობა!", "როგორ ხარ?"]

# ვექტორიზატორის ინიციალიზაცია
vectorizer = CountVectorizer()

# ვექტორიზატორის დავარჯიშება და ტრანსფორმირება
X = vectorizer.fit_transform(sentences)

# შედეგის დაბეჭდვა
```

```
print(X.toarray())
# შედეგი: [[1 0 0]
#          [0 1 1]]


# ლექსიკონის დათვალიერება
print(vectorizer.get_feature_names_out())
# შედეგი: ['გამარჯობა', 'როგორ', 'ხარ']
```
ამ მაგალითში, CountVectorizer-მა შექმნა სიტყვების ლექსიკონი და შემდეგ დაითვალა თითოეული სიტყვის სიხშირე შესაბამის წინადადებაში.

ინტერაქტიული სავარჯიშო 1:

გამოიყენე `CountVectorizer`, რათა ტექსტი `["კომპიუტერი მაგარია!", "მანქანური სწავლება რთულია?"]` ვექტორად გადააქციო.

```
# CountVectorizer-ის იმპორტი

from sklearn.feature_extraction.text import CountVectorizer


# მონაცემები

sentences = ["კომპიუტერი მაგარია!", "მანქანური სწავლება რთულია?"]


# ვექტორიზატორის ინიციალიზაცია

vectorizer = CountVectorizer()


# ვექტორიზატორის დავარჯიშება და ტრანსფორმირება

X = vectorizer.fit_transform(sentences)


# დაბეჭდე მიღებული ვექტორი

print(X.toarray())
```

```
#შედეგი:

# [[1 1 0 0 0]

# [0 0 1 1 1]]
```

3. მოდელის აგება და ვარჯიში

ახლა, როდესაც ტექსტი რიცხვებად გადავაქციეთ, შეგვიძლია ჩვენი მოდელი გავავარჯიშოთ.

3.1. ლოჯისტიკური რეგრესია (Logistic Regression)

ლოჯისტიკური რეგრესია არის მარტივი, მაგრამ მძლავრი ალგორითმი, რომელიც კლასიფიკაციისთვის გამოიყენება. ის ჩვენს შემთხვევაში დაეხმარება მოდელს, რომ მომხმარებლის შეტყობინება სწორ განზრახვას მიაკუთვნოს.

3.2. რა შეუძლია ლოგიკურ რეგრესიას

წარმოიდგინე, რომ ლოჯისტიკური რეგრესია არის პროგრამა, რომელსაც შეუძლია გამოთვალოს, არის თუ არა კერძი, რომლის მომზადებასაც აპირებ, გემრიელი. შენ მას აძლევ რეცეპტების მონაცემებს (ინგრედიენტები, მომზადების ტემპერატურა და ა.შ.) და შენთვის კარგად ცნობილი კერძების შეფასებას (გემრიელი/უგემური). მოდელი ამ მონაცემების მიხედვით სწავლობს, რა შემთხვევაშია კერძი გემრიელი. შემდეგ, როცა ახალი კერძის რეცეპტს აძლევ, ის პროგნოზირებს, იქნება თუ არა ის გემრიელი.

3.3. `model.fit(X_train, y_train)`: როგორ გავწვრთნათ მოდელი?

`fit()` მეთოდი არის ყველაზე მნიშვნელოვანი ეტაპი, რადგან სწორედ ამ დროს სწავლობს მოდელი. .

ანალოგია: წარმოიდგინე, რომ გამოცდისთვის ემზადები. `X_train` არის სასწავლო მასალა, მაგალითად, წიგნები და სავარჯიშოები. `y_train` კი არის ამ სავარჯიშოების სწორი პასუხები. შენ სწავლობ წიგნებიდან (`X_train`) და დავალებების პასუხებს ამოწმებ (`y_train`) ის გამოყენებით.

პროგრამირებაში ეს ასე მუშაობს: მოდელი იღებს ორ მონაცემს: `X_train` (ვექტორიდან წინადადებას) და `y_train` (მის შესაბამის განზრახვას). ამ მონაცემების საფუძველზე, მოდელი ამყარებს მათ შორის კავშირებს და სწავლობს, თუ რომელი სიტყვები ან სიტყვების კომბინაციებია ყველაზე მნიშვნელოვანი კონკრეტული განზრახვის ამოსაცნობად. მაგალითად, თუ ჩვენს მოდელს მივაწვდით მონაცემებს, სადაც სიტყვა „გამარჯობა“ ყოველთვის დაკავშირებულია `greeting` განზრახვასთან, ის მიხვდება, რომ ამ ორს შორის მჭიდრო კავშირია. ანალოგიურად, თუ სიტყვა „ნახვამდის“ ყოველთვის `goodbye` განზრახვასთანაა დაკავშირებული, მოდელი ამასაც დაიმახსოვრებს. ამგვარად, მოდელი ქმნის ერთგვარ „შიდა ლექსიკონს“, სადაც თითოეულ სიტყვას ან სიტყვების კომბინაციას კონკრეტული განზრახვა უკავშირდება. ეს პროცესი, რომელსაც **ტრენინგი** ეწოდება, მოდელის საბოლოო ცოდნის ბაზის შექმნას ემსახურება.

ინტერაქტიული სავარჯიშო 2:

გამოიყენე `fit()` მეთოდი, რათა გაავარჯიშო `LogisticRegression` მოდელი მოცემულ მონაცემებზე.

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression

# მონაცემები
texts = ["გამარჯობა", "როგორ ხარ", "ნახვამდის"]
intents = ["greeting", "question", "goodbye"]

# მონაცემების ვექტორად გარდაქმნა
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(texts)
y = intents

# მოდელის ინიციალიზაცია
model = LogisticRegression()

# აქ დაწერე შენი კოდი აქ: გამოიყენე fit() მეთოდი მოდელის გასავარჯიშებლად
# მეთოდს გადასცე X და y ცვლადები
model.fit(X, y)

# ამ კოდს ნუ შეცვლით
print("მოდელი გაწვრთნილია!")
```

3.3. `model.predict(X_test)`: როგორ გამოვიყენოთ მოდელი ახალი მონაცემების შესაფასებლად?

როდესაც მოდელი უკვე გაწვრთნილია, შეგვიძლია მას ახალი, უცნობი მონაცემები მივაწოდოთ და ვთხოვოთ, რომ პროგნოზი გააკეთოს. `predict()` მეთოდი სწორედ ამისთვის გამოიყენება.

```
# საჭირო ბიბლიოთეკების იმპორტი
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression

# მონაცემები
texts = [
    "გამარჯობა!",
    "საღამი",
    "მოგესალმები",
```

```
    "დილა მშვიდობის",

    "როგორ ხარ?",
    "რა გქვია?",
    "რას საქმიანობ?",
    "დახმარება შეგიძლია?",

    "ნახვამდის.",
    "კარგად იყავი",
    "დროებით",
    "აბა ჰე",

    "მშვენიერია",
    "კარგია",
    "გეთანხმები",
    "რა თქმა უნდა"
]

intents = [
    "მისალმება",
    "მისალმება",
    "მისალმება",
    "მისალმება",

    "კითხვა",
    "კითხვა",
    "კითხვა",
    "კითხვა",

    "დამშვიდობება",
    "დამშვიდობება",
    "დამშვიდობება",
    "დამშვიდობება",

    "დადასტურება",
    "დადასტურება",
    "დადასტურება",
    "დადასტურება"
]

# მონაცემების ვექტორებად გარდაქმნა
```

```
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(texts)
y = intents

# მოდელის ინიციალიზაცია
model = LogisticRegression()

# მოდელის ვარჯიში
model.fit(X, y)

# ახალი ტექსტი პროგნოზირებისთვის
new_text = ["გამარჯობა ყველას!"]

# ახალი ტექსტის ვექტორად გარდაქმნა (იგივე ვექტორიზატორის გამოყენებით)
X_new = vectorizer.transform(new_text)

# პროგნოზის გაკეთება
prediction = model.predict(X_new)

print(f"პროგნოზი: {prediction}") # შედეგი: ['მისალმება']
```

დავალება 11: "ემოციური ფონის ანალიზი"

შექმნი მოდელი, რომელიც შეძლებს ტექსტიდან ემოციური ფონის (სენტიმენტის) ამოკითხვას. ჩვენ გამოვიყენებთ კინო-მიმოხილვების მცირე მონაცემთა ბაზას. Scikit-learn-ის `CountVectorizer`-ისა და `LogisticRegression`-ის გამოყენებით, გაავარჯიშე მოდელი, რათა მან იწინასწარმეტყველოს უცნობი მიმოხილვა **დადებითია** (`positive`) თუ **უარყოფითი** (`negative`).

დავალება 11.1: შეცდომის პოვნა და გასწორება

მოცემულ კოდში დაშვებულია 1 ლოგიკური შეცდომა. შენი ამოცანაა, იპოვო და გაასწორო ის. კოდმა, პროგრამამ უნდა შეძლოს ახალი მიმოხილვის სენტიმენტის სწორად განსაზღვრა.

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression

# მონაცემთა ბაზა (ლექსიკონის ფორმატში)
data = {
    "საოცარი ფილმია, ყველამ უნდა ნახოს.": "positive",
```

```
    "საშინლად მოსაწყენი ფილმი იყო.": "negative",
    "ძალიან მომეწონა მსახიობების თამაში.": "positive",
    "არ მომეწონა, დროის კარგვაა.": "negative",
    "შესანიშნავი რეჟისურა და სიუჟეტი.": "positive",
    "სუსტი სცენარი და დიალოგები.": "negative",
    "ერთ-ერთი საუკეთესო ფილმი, რაც მინახავს.": "positive",
    "ძალიან იმედგაცრუებული დავრჩი.": "negative",
    "დადებითი ემოციებით დავიმუხტე.": "positive",
    "არ გირჩევთ ამ ფილმის ყურებას.": "negative",
    "ნამდვილად ღირს ყურება.": "positive",
    "სრული იმედგაცრუება.": "negative",
    "საუკეთესო კომედიაა ბოლო წლებში.": "positive",
    "მოსაწყენი და გაწელილი სიუჟეტი.": "negative"
}

# მონაცემების მომზადება: ლექსიკონიდან სიების შექმნა
reviews = list(data.keys())
sentiments = list(data.values())

# ტექსტის ვექტორიზაცია
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(reviews)
y = sentiments

# მოდელის შექმნა და ვარჯიში
model = LogisticRegression()
model.fit(X, y)

# ახალი ტექსტი პროგნოზირებისთვის
new_review = ["ეს საოცარი ფილმია"]

# ახალი ტექსტის ვექტორიზაცია
X_new = vectorizer.fit_transform(new_review) # მინიშნება: ახალ მონაცემებს ხელახალი სწავლება არ სჭირდება.

prediction = model.predict(X_new)
print(f"პროგნოზირებული სენტიმენტი: {prediction[0]}")
```

დავალება 11.2: კოდის დასრულება

ამ კპრორგამულ კოდს აკლია ერთი მნიშვნელოვანი ხაზი, რის გარეშეც მოდელი ვერ იმუშავებს. შეავსე გამოტოვებული ნაწილი.

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression

# მონაცემთა ბაზა (ლექსიკონის ფორმატში)
data = {
    "საოცარი ფილმია, ყველამ უნდა ნახოს.": "positive",
    "საშინლად მოსაწყენი ფილმი იყო.": "negative",
    "ძალიან მომეწონა მსახიობების თამაში.": "positive",
    "არ მომეწონა, დროის კარგვაა.": "negative",
    "შესანიშნავი რეჟისურა და სიუჟეტი.": "positive",
    "სუსტი სცენარი და დიალოგები.": "negative",
    "ერთ-ერთი საუკეთესო ფილმი, რაც მინახავს.": "positive",
    "ძალიან იმედგაცრუებული დავრჩი.": "negative",
    "დადებითი ემოციებით დავიმუხტე.": "positive",
    "არ გირჩევთ ამ ფილმის ყურებას.": "negative",
    "ნამდვილად ღირს ყურება.": "positive",
    "სრული იმედგაცრუება.": "negative",
    "საუკეთესო კომედიაა ბოლო წლებში.": "positive",
    "მოსაწყენი და გაწელილი სიუჟეტი.": "negative"
}

# მონაცემების მომზადება: ლექსიკონიდან სიების შექმნა
reviews = list(data.keys())
sentiments = list(data.values())

# ტექსტის ვექტორიზაცია
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(reviews)
y = sentiments

# მოდელის შექმნა
model = LogisticRegression()

# *** დაწერე შენი კოდი აქ: დაამატე ხაზი, რომელიც მოდელს მონაცემებზე გაავარჯიშებს ***


# ახალი ტექსტი პროგნოზირებისთვის
new_review = ["ეს საოცარი ფილმია"]
```

```
X_new = vectorizer.transform(new_review)

prediction = model.predict(X_new)
print(f"პროგნოზირებული სენტიმენტი: {prediction[0]}")
```

დავალება 11.3: შეიმუშავე პრორგამული კოდი
ახლა შენი ჯერია შეიმუშავო პროგრამული კოდი, რომელიც ზემოთ მოცემულ მონაცემთა ბაზას გამოიყენებს, გაავარჯიშებს მოდელს და იწინასწარმეტყველებს ახალი მიმოხილვის ("ეს ფილმი შედევრია") სენტიმენტს.

```
# დაწერე შენი კოდი აქ:
# 1. შემოიტანე საჭირო ბიბლიოთეკები (CountVectorizer, LogisticRegression).
# 2. შექმენი მონაცემთა ბაზა dictionary-ს სახით.
# 3. ლექსიკონიდან შექმენი ორი სია: ტექსტები და სენტიმენტები.
# 4. მოახდინე ტექსტის ვექტორიზაცია.
# 5. შექმენი და გაავარჯიშე LogisticRegression მოდელი.
# 6. გააკეთე პროგნოზი ახალ მიმოხილვაზე და დაბეჭდე შედეგი.
```

სწორი პასუხი (პროგრამული კოდი სრულად):

```
# 1. საჭირო ბიბლიოთეკების იმპორტი
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression

# 2. მონაცემთა ბაზის შექმნა (ლექსიკონის ფორმატში)
data = {
    "საოცარი ფილმია, ყველამ უნდა ნახოს.": "positive",
    "საშინლად მოსაწყენი ფილმი იყო.": "negative",
    "ძალიან მომეწონა მსახიობების თამაში.": "positive",
    "არ მომეწონა, დროის კარგვაა.": "negative",
    "შესანიშნავი რეჟისურა და სიუჟეტი.": "positive",
    "სუსტი სცენარი და დიალოგები.": "negative",
    "ერთ-ერთი საუკეთესო ფილმი, რაც მინახავს.": "positive",
    "ძალიან იმედგაცრუებული დავრჩი.": "negative",
    "დადებითი ემოციებით დავიმუხტე.": "positive",
```

```
    "არ გირჩევთ ამ ფილმის ყურებას.": "negative",
    "ნამდვილად ღირს ყურება.": "positive",
    "სრული იმედგაცრუება.": "negative",
    "საუკეთესო კომედიაა ბოლო წლებში.": "positive",
    "მოსაწყენი და გაწელილი სიუჟეტი.": "negative",
    "მსახიობების შესრულება უმაღლეს დონეზეა.": "positive",
    "ყველაზე ცუდი ფილმი, რაც მინახავს.": "negative"
}

# 3. მონაცემების მომზადება: ლექსიკონიდან სიების შექმნა
reviews = list(data.keys())
sentiments = list(data.values())

# 4. ტექსტის ვექტორიზაცია: ტექსტის გადაქცევა რიცხვებად
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(reviews)
y = sentiments

# 5. მოდელის შექმნა და ვარჯიში
model = LogisticRegression()
model.fit(X, y)

# 6. ახალი ტექსტი პროგნოზირებისთვის
new_review = ["ეს საოცარი ფილმია"]

# 7. ახალი ტექსტის ვექტორად გარდაქმნა (მხოლოდ transform)
X_new = vectorizer.transform(new_review)

# 8. პროგნოზის გაკეთება და შედეგის დაბეჭდვა
prediction = model.predict(X_new)
print(f"ახალი მიმოხილვა: '{new_review[0]}'")
print(f"პროგნოზირებული სენტიმენტი: {prediction[0]}")
```