Poglavlje 9

Goodmanovi svijetovi: Problem indukcije i strojno učenje

9.1 Novi problem indukcije kroz prizmu računalnih znanosti

Nelson Goodman je 1955. godine formulirao **novi problem indukcije** koji pokazuje temeljnu poteškoću u razlikovanju valjanih od nevaljanih induktivnih zaključaka. Ovaj problem ima duboke implikacije za moderno strojno učenje.

> "Činjenica da se smaragd može jednako dobro opisati kao 'zelen' ili kao 'gruen' otkriva da svaki skup opažanja podržava beskonačno mnogo hipoteza." - Nelson Goodman

U ovoj bilježnici istražujemo kako se Goodmanova zagadka manifestira u kontekstu strojnog učenja kroz **No-Free-Lunch teoreme**.

9.1.1 Klasični problem indukcije

Prije nego što uronimo u Goodmanov novi problem, razmotrimo klasični **Humeov problem indukcije**:

$$\text{Opaženi slučajevi} \xrightarrow{?} \text{Opći zaključak}$$

Induktivno zaključivanje pokušava iz konačnog broja opažanja izvesti općeniti zakon. To je temelj znanosti, ali filozofski gledano - nema logičke nužnosti da će se buduća opažanja ponašati kao prošla.

```
1 from dataclasses import dataclass
```

```python
from datetime import datetime, timedelta
import random

@dataclass
class Opažanje:
    """Predstavlja jedno empirijsko opažanje."""
    objekt: str
    svojstvo: str
    vrijeme: datetime

    def __repr__(self):
        return f"{self.objekt}: {self.svojstvo} (vrijeme: {self.vrijeme.date()})"

# Generiraj opažanja smaragda
def generiraj_opažanja(n=5, svojstvo="zelen"):
    """Generira niz opažanja smaragda."""
    opažanja = []
    početak = datetime(2020, 1, 1)

    for i in range(n):
        vrijeme = početak + timedelta(days=random.randint(i*200, (i+1)*200))
        opažanja.append(Opažanje(f"Smaragd {i+1}", svojstvo, vrijeme))

    return opažanja

# Klasična indukcija
opažanja_zeleni = generiraj_opažanja(5, "zelen")

print("Klasična indukcija:")
print("="*20)
print("Opažanja smaragda:")
for op in opažanja_zeleni:
    print(f"  {op}")
print("\nInduktivni zaključak: Svi smaragdi su zeleni")
```

Output

```
Klasična indukcija:
====================
Opažanja smaragda:
  Smaragd 1: zelen (vrijeme: 2020-03-17)
  Smaragd 2: zelen (vrijeme: 2021-01-21)
  Smaragd 3: zelen (vrijeme: 2021-06-04)
  Smaragd 4: zelen (vrijeme: 2022-03-01)
  Smaragd 5: zelen (vrijeme: 2022-07-22)

Induktivni zaključak: Svi smaragdi su zeleni
```

9.1.2 Goodmanov "grue" predikat

Goodman uvodi novi predikat **"grue"** (kombinacija "green" i "blue"):

$$\text{grue}(x) = \begin{cases} \text{zelen}(x) & \text{ako je } x \text{ opažen prije } t_0 \\ \text{plav}(x) & \text{ako je } x \text{ opažen nakon } t_0 \end{cases}$$

gdje je t_0 neki budući trenutak (npr. 1. siječnja 2100.).

Paradoks: Sva dosadašnja opažanja zelenih smaragda jednako dobro potvrđuju hipotezu "svi smaragdi su zeleni" kao i hipotezu "svi smaragdi su grueni"!

```python
class GruePredikat:

    """Implementacija Goodmanovog 'grue' predikata."""

    def __init__(self, kritični_trenutak):
        self.t0 = kritični_trenutak

    def je_grue(self, objekt, vrijeme_opažanja):
        """Provjerava je li objekt 'grue' u danom trenutku."""
        if vrijeme_opažanja < self.t0:
            # Prije t0: grue = zelen
            return objekt.svojstvo == "zelen"
        else:
            # Nakon t0: grue = plav
            return objekt.svojstvo == "plav"

    def predviđa(self, vrijeme):
        """Što predviđa grue hipoteza za dano vrijeme."""
        if vrijeme < self.t0:
            return "zelen"
        else:
            return "plav"

# Definiraj kritični trenutak
t0 = datetime(2100, 1, 1)
grue_predikat = GruePredikat(t0)

print("Goodmanov problem indukcije:")
print("="*29)
print(f"Kritični trenutak t₀: {t0.date()}")
print("\nProvjera hipoteza za prošla opažanja:")
print("-"*38)

# Testiraj obje hipoteze na istim podacima
for op in opažanja_zeleni[:3]:  # Prikaži prva 3
    print(f"{op.objekt} ({op.vrijeme.date()}):")
    print(f"  Opaženo: {op.svojstvo}")
```

```
38
39      # Hipoteza 1: Svi smaragdi su zeleni
40      h1_istina = op.svojstvo == "zelen"
41      print(f"  H1 (zeleni): {'⊤' if h1_istina else '⊥'} (predviđa: zelen)")
42
43      # Hipoteza 2: Svi smaragdi su grueni
44      h2_predviđanje = grue_predikat.predviđa(op.vrijeme)
45      h2_istina = op.svojstvo == h2_predviđanje
46      print(f"  H2 (grueni): {'⊤' if h2_istina else '⊥'} (predviđa: {h2_predviđanje} -
        ↪ {'prije' if op.vrijeme < t0 else 'nakon'} t₀)")
47      print()
48
49  print("Rezultat: Obje hipoteze jednako dobro objašnjavaju sva opažanja!")
50  print(f"\nPredviđanja za budućnost (nakon {t0.date()}):")
51  print("-"*46)
52  print("  H1 (svi zeleni): smaragdi će biti zeleni")
53  print("  H2 (svi grueni): smaragdi će biti plavi")
```

Output

```
Goodmanov problem indukcije:
============================
Kritični trenutak t₀: 2100-01-01

Provjera hipoteza za prošla opažanja:
--------------------------------------
Smaragd 1 (2020-03-17):
  Opaženo: zelen
  H1 (zeleni): ⊤ (predviđa: zelen)
  H2 (grueni): ⊤ (predviđa: zelen - prije t₀)

Smaragd 2 (2021-01-21):
  Opaženo: zelen
  H1 (zeleni): ⊤ (predviđa: zelen)
  H2 (grueni): ⊤ (predviđa: zelen - prije t₀)

Smaragd 3 (2021-06-04):
  Opaženo: zelen
  H1 (zeleni): ⊤ (predviđa: zelen)
  H2 (grueni): ⊤ (predviđa: zelen - prije t₀)

Rezultat: Obje hipoteze jednako dobro objašnjavaju sva opažanja!

Predviđanja za budućnost (nakon 2100-01-01):
----------------------------------------------
  H1 (svi zeleni): smaragdi će biti zeleni
  H2 (svi grueni): smaragdi će biti plavi
```

9.1.3 Beskonačnost alternativnih hipoteza

Goodmanov argument pokazuje da za svaki skup opažanja postoji **beskonačno mnogo** jednako dobro podržanih hipoteza. Možemo konstruirati "grue$_1$", "grue$_2$", ... sa različitim kritičnim trenucima:

$$\text{grue}_n(x) = \begin{cases} \text{zelen}(x) & \text{ako je } x \text{ opažen prije } t_n \\ \text{plav}(x) & \text{ako je } x \text{ opažen nakon } t_n \end{cases}$$

```python
def stvori_grue_hipotezu(kritični_trenutak, naziv):
    """Stvara grue hipotezu s danim kritičnim trenutkom."""
    class GrueHipoteza:
        def __init__(self):
            self.t0 = kritični_trenutak
            self.naziv = naziv

        def predviđa(self, vrijeme):
            return "zelen" if vrijeme < self.t0 else "plav"

        def provjeri(self, opažanje):
            predviđanje = self.predviđa(opažanje.vrijeme)
            return opažanje.svojstvo == predviđanje

    return GrueHipoteza()

# Stvori više alternativnih hipoteza
hipoteze = {
    "standard": type('StandardHipoteza', (), {
        'naziv': 'standard',
        'predviđa': lambda self, t: "zelen",
        'provjeri': lambda self, op: op.svojstvo == "zelen"
    })(),
    "grue_2030": stvori_grue_hipotezu(datetime(2030, 1, 1), "grue_2030"),
    "grue_2050": stvori_grue_hipotezu(datetime(2050, 1, 1), "grue_2050"),
    "grue_2100": stvori_grue_hipotezu(datetime(2100, 1, 1), "grue_2100"),
    "grue_2200": stvori_grue_hipotezu(datetime(2200, 1, 1), "grue_2200"),
}

print("Beskonačnost alternativnih hipoteza:")
print("="*37)
print("\nZa iste podatke možemo konstruirati mnoštvo hipoteza:\n")

# Prikaži predviđanja svake hipoteze
test_vremena = [datetime(2050, 6, 15), datetime(2150, 6, 15)]

for naziv, hipoteza in hipoteze.items():
    if naziv == "standard":
        print(f"Hipoteza '{naziv}': Svi smaragdi su uvijek zeleni")
    else:
        t0_godina = int(naziv.split('_')[1])
```

```
        print(f"Hipoteza '{naziv}': grueni s prijelazom {t0_godina}-01-01")

    for t in test_vremena:
        print(f"  Predviđanje za {t.year}: {hipoteza.predviđa(t)}")
    print()

# Provjeri konzistentnost s postojećim opažanjima
print("Provjera konzistentnosti s opažanjima:")
print("-"*40)

for naziv, hipoteza in hipoteze.items():
    rezultati = [hipoteza.provjeri(op) for op in opažanja_zeleni]
    simboli = "".join(['⊤' if r else '⊥' for r in rezultati])
    točnih = sum(rezultati)
    print(f"{naziv}: {simboli} ({točnih}/{len(rezultati)} točnih)")

print("\nSve hipoteze su jednako dobro podržane postojećim podacima!")
```

Output

```
Beskonačnost alternativnih hipoteza:
====================================

Za iste podatke možemo konstruirati mnoštvo hipoteza:

Hipoteza 'standard': Svi smaragdi su uvijek zeleni
  Predviđanje za 2050: zelen
  Predviđanje za 2150: zelen

Hipoteza 'grue_2030': grueni s prijelazom 2030-01-01
  Predviđanje za 2050: plav
  Predviđanje za 2150: plav

Hipoteza 'grue_2050': grueni s prijelazom 2050-01-01
  Predviđanje za 2050: plav
  Predviđanje za 2150: plav

Hipoteza 'grue_2100': grueni s prijelazom 2100-01-01
  Predviđanje za 2050: zelen
  Predviđanje za 2150: plav

Hipoteza 'grue_2200': grueni s prijelazom 2200-01-01
  Predviđanje za 2050: zelen
  Predviđanje za 2150: zelen

Provjera konzistentnosti s opažanjima:
----------------------------------------
standard: ⊤⊤⊤⊤⊤ (5/5 točnih)
grue_2030: ⊤⊤⊤⊤⊤ (5/5 točnih)
grue_2050: ⊤⊤⊤⊤⊤ (5/5 točnih)
```

```
grue_2100: TTTTT (5/5 točnih)
grue_2200: TTTTT (5/5 točnih)

Sve hipoteze su jednako dobro podržane postojećim podacima!
```

9.1.4 No-Free-Lunch teoremi u strojnom učenju

No-Free-Lunch (NFL) teoremi pokazuju da je Goodmanov problem duboko ukorijenjen u strojnom učenju. Teorem kaže:

$$\sum_{f \in \mathcal{F}} \mathrm{GP}(L, f) = 0.5$$

gdje je:

- L - bilo koji algoritam učenja
- $\mathcal{F}$ - skup svih mogućih ciljnih funkcija
- GP - generalizacijska performansa

Značenje: Prosječna performansa bilo kojeg algoritma učenja preko svih mogućih problema jednaka je slučajnom pogađanju!

```python
import itertools

class BooleanConcept:
    """Predstavlja Booleovu funkciju kao koncept za učenje."""

    def __init__(self, truth_table, naziv=""):
        self.tablica = truth_table
        self.naziv = naziv

    def evaluiraj(self, ulaz):
        """Evaluira funkciju za dani ulaz."""
        return self.tablica.get(ulaz, False)

    def konzistentan_s(self, skup_učenja):
        """Provjerava je li koncept konzistentan sa skupom za učenje."""
        for ulaz, izlaz in skup_učenja.items():
            if self.evaluiraj(ulaz) != izlaz:
                return False
        return True

# Definiraj skup za učenje (parcijalna tablica istinitosti)
skup_učenja = {
```

```
    (False, False): False,
    (False, True): True,
    (True, False): True,
    # (True, True) nije u skupu za učenje!
}

# Konstruiraj dva različita koncepta konzistentna s podacima
# Koncept 1: XOR
xor_tablica = {
    (False, False): False,
    (False, True): True,
    (True, False): True,
    (True, True): False  # XOR
}

# Koncept 2: OR
or_tablica = {
    (False, False): False,
    (False, True): True,
    (True, False): True,
    (True, True): True  # OR
}

koncepti = [
    BooleanConcept(xor_tablica, "XOR funkcija"),
    BooleanConcept(or_tablica, "OR funkcija")
]

print("No-Free-Lunch demonstracija:")
print("="*29)
print()

# Prikaži skup za učenje
print("Skup za učenje: ", end="")
print("{", end="")
for i, (ulaz, izlaz) in enumerate(skup_učenja.items()):
    if i > 0:
        print(", ", end="")
    ulaz_str = f"({'⊤' if ulaz[0] else '⊥'}, {'⊤' if ulaz[1] else '⊥'})"
    izlaz_str = '⊤' if izlaz else '⊥'
    print(f"{ulaz_str}: {izlaz_str}", end="")
print("}")

testni = (True, True)
print(f"Testni primjer: ({'⊤' if testni[0] else '⊥'}, {'⊤' if testni[1] else '⊥'}) = ?")
print()

print("Mogući koncepti konzistentni s podacima:")
print("-"*42)

for i, koncept in enumerate(koncepti, 1):
    print(f"Koncept {i}: {koncept.naziv}")

```

```
    # Predviđanje za testni primjer
    pred = koncept.evaluiraj(testni)
    print(f"  Predviđanje za ({'⊤' if testni[0] else '⊥'}, {'⊤' if testni[1] else '⊥'}):
    ↪ {'⊤' if pred else '⊥'}")

    # Prikaži cijelu tablicu
    print("  Tablica:")
    for ulaz in [(False, False), (False, True), (True, False), (True, True)]:
        izlaz = koncept.evaluiraj(ulaz)
        ulaz_str = f"({'⊤' if ulaz[0] else '⊥'}, {'⊤' if ulaz[1] else '⊥'})"
        izlaz_str = '⊤' if izlaz else '⊥'
        oznaka = " ✓" if ulaz in skup_učenja else ""
        print(f"    {ulaz_str} → {izlaz_str}{oznaka}")
    print()

print("Oba koncepta su jednako valjana za dane podatke!")
print("Ali daju različita predviđanja za neviđene primjere.")
```

Output

```
No-Free-Lunch demonstracija:
============================

Skup za učenje: {(⊥, ⊥): ⊥, (⊥, ⊤): ⊤, (⊤, ⊥): ⊤}
Testni primjer: (⊤, ⊤) = ?

Mogući koncepti konzistentni s podacima:
----------------------------------------
Koncept 1: XOR funkcija
  Predviđanje za (⊤, ⊤): ⊥
  Tablica:
    (⊥, ⊥) → ⊥ ✓
    (⊥, ⊤) → ⊤ ✓
    (⊤, ⊥) → ⊤ ✓
    (⊤, ⊤) → ⊥

Koncept 2: OR funkcija
  Predviđanje za (⊤, ⊤): ⊤
  Tablica:
    (⊥, ⊥) → ⊥ ✓
    (⊥, ⊤) → ⊤ ✓
    (⊤, ⊥) → ⊤ ✓
    (⊤, ⊤) → ⊤

Oba koncepta su jednako valjana za dane podatke!
Ali daju različita predviđanja za neviđene primjere.
```

9.1.5 Konstrukcija "savijenih" koncepata

NFL teoremi pokazuju da za svaki koncept C koji algoritam učenja dobro nauči, postoji "savijeni" koncept C' koji će biti loše naučen:

$$C'(x) = \begin{cases} C(x) & \text{ako } x \in \text{SkupUčenja} \\ \neg C(x) & \text{ako } x \notin \text{SkupUčenja} \end{cases}$$

Ovo je direktna analogija s Goodmanovim "grue" predikatom!

```python
import numpy as np

def stvori_savijeni_koncept(originalni_koncept, skup_učenja):
    """Stvara 'savijeni' koncept koji se slaže na skupu učenja ali ne generalizira."""

    class SavijeniKoncept:
        def __init__(self):
            self.originalni = originalni_koncept
            self.memorija = skup_učenja

        def predviđa(self, primjer):
            # Ako je primjer u skupu učenja, koristi originalnu funkciju
            for x_mem, _ in self.memorija:
                if np.allclose(primjer, x_mem):
                    return self.originalni(primjer)

            # Inače, vrati suprotno od originalne funkcije
            return not self.originalni(primjer)

    return SavijeniKoncept()

# Definiraj jednostavan linearni koncept
def linearni_koncept(x):
    """Jednostavan linearni klasifikator: x[0] + x[1] > 1"""
    return x[0] + x[1] > 1

# Generiraj skup za učenje
np.random.seed(42)
skup_učenja_ml = []
for _ in range(6):
    x = np.random.rand(2)
    y = linearni_koncept(x)
    skup_učenja_ml.append((x, y))

# Stvori savijeni koncept
savijeni = stvori_savijeni_koncept(linearni_koncept, skup_učenja_ml)

print("Konstrukcija 'savijenih' koncepata:")
print("="*36)
```

```
print("\nOriginalni koncept C: jednostavna linearna granica")
print(f"Skup za učenje: {len(skup_učenja_ml)} primjera")
print()

# Testiraj na skupu za učenje
print("Performanse na skupu za učenje:")
print("-"*32)
točnih_C = 0
točnih_C_prime = 0

for x, y in skup_učenja_ml:
    pred_C = linearni_koncept(x)
    pred_C_prime = savijeni.predviđa(x)

    print(f"Primjer ({x[0]:.1f}, {x[1]:.1f}) → ", end="")
    print(f"Očekivano: {'⊤' if y else '⊥'}, ", end="")
    print(f"C: {'⊤' if pred_C else '⊥'} {'✓' if pred_C == y else '×'}, ", end="")
    print(f"C': {'⊤' if pred_C_prime else '⊥'} {'✓' if pred_C_prime == y else '×'}")

    if pred_C == y:
        točnih_C += 1
    if pred_C_prime == y:
        točnih_C_prime += 1

print(f"\nTočnost na skupu za učenje:")
print(f"  Koncept C: {100 * točnih_C / len(skup_učenja_ml):.1f}%")
print(f"  Koncept C': {100 * točnih_C_prime / len(skup_učenja_ml):.1f}%")

# Testiraj na novim primjerima
print("\nPerformanse na testnom skupu:")
print("-"*30)

testni_skup = [np.random.rand(2) for _ in range(5)]
točnih_test_C = 0
točnih_test_C_prime = 0

for x_test in testni_skup:
    y_pravi = linearni_koncept(x_test)  # "Prava" oznaka
    pred_C = linearni_koncept(x_test)
    pred_C_prime = savijeni.predviđa(x_test)

    print(f"Test ({x_test[0]:.1f}, {x_test[1]:.1f}) → ", end="")
    print(f"C: {'⊤' if pred_C else '⊥'}, ", end="")
    print(f"C': {'⊤' if pred_C_prime else '⊥'}", end="")
    if pred_C != pred_C_prime:
        print(" (različito!)")
    else:
        print()

    if pred_C == y_pravi:
        točnih_test_C += 1
    if pred_C_prime == y_pravi:
        točnih_test_C_prime += 1
```

```
93
94 print(f"\nTočnost na testnom skupu:")
95 print(f"  Koncept C: {100 * točnih_test_C / len(testni_skup):.1f}% (dobar)")
96 print(f"  Koncept C': {100 * točnih_test_C_prime / len(testni_skup):.1f}% (loš!)")
97 print("\nC' je 'savijen' - slaže se s C na skupu za učenje,")
98 print("ali se ponaša suprotno na novim primjerima!")
```

Output

```
Konstrukcija 'savijenih' koncepata:
====================================

Originalni koncept C: jednostavna linearna granica
Skup za učenje: 6 primjera

Performanse na skupu za učenje:
--------------------------------
Primjer (0.4, 1.0) → Očekivano: ⊤, C: ⊤ ✓, C': ⊤ ✓
Primjer (0.7, 0.6) → Očekivano: ⊤, C: ⊤ ✓, C': ⊤ ✓
Primjer (0.2, 0.2) → Očekivano: ⊥, C: ⊥ ✓, C': ⊥ ✓
Primjer (0.1, 0.9) → Očekivano: ⊥, C: ⊥ ✓, C': ⊥ ✓
Primjer (0.6, 0.7) → Očekivano: ⊤, C: ⊤ ✓, C': ⊤ ✓
Primjer (0.0, 1.0) → Očekivano: ⊥, C: ⊥ ✓, C': ⊥ ✓

Točnost na skupu za učenje:
  Koncept C: 100.0%
  Koncept C': 100.0%

Performanse na testnom skupu:
------------------------------
Test (0.8, 0.2) → C: ⊤, C': ⊥ (različito!)
Test (0.2, 0.2) → C: ⊥, C': ⊤ (različito!)
Test (0.3, 0.5) → C: ⊥, C': ⊤ (različito!)
Test (0.4, 0.3) → C: ⊥, C': ⊤ (različito!)
Test (0.6, 0.1) → C: ⊥, C': ⊤ (različito!)

Točnost na testnom skupu:
  Koncept C: 100.0% (dobar)
  Koncept C': 0.0% (loš!)

C' je 'savijen' - slaže se s C na skupu za učenje,
ali se ponaša suprotno na novim primjerima!
```

9.1.6 Implikacije za strojno učenje

Induktivni bias kao nužnost

NFL teoremi i Goodmanov problem pokazuju da **induktivni bias nije nedostatak već nužnost** svakog sustava učenja. Bez pretpostavki o prirodi problema, učenje je nemoguće.

Različiti algoritmi imaju različite induktivne pristranosti:

- **Linearna regresija**: pretpostavlja linearnu vezu
- **Stabla odlučivanja**: pretpostavljaju hijerarhijsku strukturu
- **Neuronske mreže**: pretpostavljaju kompozicionalnost
- **k-NN**: pretpostavlja lokalnu glatkoću

```python
def najbliži_susjed(točka, skup_učenja):
    """Jednostavan 1-NN klasifikator."""
    min_udaljenost = float('inf')
    najbliža_oznaka = None

    for x, y in skup_učenja:
        udaljenost = np.sqrt((točka[0] - x[0])**2 + (točka[1] - x[1])**2)
        if udaljenost < min_udaljenost:
            min_udaljenost = udaljenost
            najbliža_oznaka = y

    return najbliža_oznaka

def xor_funkcija(x):
    """XOR funkcija: istinito ako je točno jedan ulaz > 0.5."""
    return (x[0] > 0.5) != (x[1] > 0.5)

# Generiraj XOR podatke
xor_podaci = [
    (np.array([0.1, 0.1]), False),
    (np.array([0.1, 0.9]), True),
    (np.array([0.9, 0.1]), True),
    (np.array([0.9, 0.9]), False),
    (np.array([0.3, 0.3]), False),
    (np.array([0.3, 0.7]), True),
    (np.array([0.7, 0.3]), True),
    (np.array([0.7, 0.7]), False),
]

# Definiraj različite algoritme s različitim biasima
algoritmi = [
    ("Linearna granica (bias: linearnost)",
     lambda x: x[0] + x[1] > 1),
```

```
    ("Najbliži susjed (bias: lokalna glatkoća)",
     lambda x: najbliži_susjed(x, xor_podaci)),
    ("XOR funkcija (bias: točno odgovara problemu)",
     xor_funkcija),
    ("Uvijek ⊤ (bias: konstantnost)",
     lambda x: True),
]

print("Utjecaj induktivnog biasa:")
print("="*27)
print(f"\nPodatkovni skup: {len(xor_podaci)} točaka koje čine XOR uzorak")
print("\nRazličiti algoritmi s različitim biasima:")
print("-"*43)

test_točke = [np.array([0.5, 0.5]), np.array([0.2, 0.8])]

for i, (naziv, algoritam) in enumerate(algoritmi, 1):
    print(f"\n{i}. {naziv}")

    # Testiraj na nekoliko točaka
    for točka in test_točke:
        pred = algoritam(točka)
        print(f"   Predviđanje za ({točka[0]:.1f}, {točka[1]:.1f}): {'⊤' if pred else '⊥'}")

    # Izračunaj točnost
    točnih = sum(1 for x, y in xor_podaci if algoritam(x) == y)
    točnost = 100 * točnih / len(xor_podaci)
    print(f"   Točnost na XOR podacima: {točnost:.1f}%")

    # Komentar
    if točnost > 90:
        print("   → Savršen jer ima pravi bias")
    elif točnost > 60:
        print("   → Bolji, ali još uvijek ograničen")
    else:
        if "linearnost" in naziv:
            print("   → Loš za XOR zbog krivog biasa")
        else:
            print("   → Loš zbog previše jednostavnog biasa")

print("\nZaključak: Uspjeh učenja ovisi o podudaranju")
print("između induktivnog biasa i stvarnog problema!")
```

Output

```
Utjecaj induktivnog biasa:
===========================

Podatkovni skup: 8 točaka koje čine XOR uzorak

Različiti algoritmi s različitim biasima:
```

```
----------------------------------------

1. Linearna granica (bias: linearnost)
   Predviđanje za (0.5, 0.5): ⊥
   Predviđanje za (0.2, 0.8): ⊥
   Točnost na XOR podacima: 25.0%
   → Loš za XOR zbog krivog biasa

2. Najbliži susjed (bias: lokalna glatkoća)
   Predviđanje za (0.5, 0.5): ⊥
   Predviđanje za (0.2, 0.8): ⊤
   Točnost na XOR podacima: 100.0%
   → Savršen jer ima pravi bias

3. XOR funkcija (bias: točno odgovara problemu)
   Predviđanje za (0.5, 0.5): ⊥
   Predviđanje za (0.2, 0.8): ⊤
   Točnost na XOR podacima: 100.0%
   → Savršen jer ima pravi bias

4. Uvijek ⊤ (bias: konstantnost)
   Predviđanje za (0.5, 0.5): ⊤
   Predviđanje za (0.2, 0.8): ⊤
   Točnost na XOR podacima: 50.0%
   → Loš zbog previše jednostavnog biasa

Zaključak: Uspjeh učenja ovisi o podudaranju
između induktivnog biasa i stvarnog problema!
```

9.1.7 Filozofske implikacije

Projektibilnost vs. neprojektibilnost

Goodman razlikuje **projektibilne** i **neprojektibilne** predikate:

- **Projektibilni**: "zelen", "okrugao", "teži od 1kg"
- **Neprojektibilni**: "grue", "gruen s prijelazom 2100."

Ali što čini predikat projektibilnim? Goodman sugerira da je to stvar **ukorijenjenosti** (*entrenchment*) u našem jeziku i praksi.

Implikacije za AI i AGI

1. **Nema univerzalnog algoritma učenja** - svaki mora imati bias
2. **Prijelaz od podataka na znanje** zahtijeva pretpostavke

3. **Ljudska inteligencija** možda uspijeva jer ima evolucijski oblikovane biase
4. **AGI sustavi** moraju riješiti problem izbora pravog biasa

```python
import random

class Agent:
    """Agent s određenim induktivnim biasom."""

    def __init__(self, bias_tip):
        self.bias = bias_tip
        self.fitness = 0

    def predviđa(self, x):
        """Predviđanje ovisno o biasu."""
        if self.bias == "linearni":
            return x[0] + x[1] > 1
        elif self.bias == "kvadratni":
            return x[0]**2 + x[1]**2 > 0.5
        elif self.bias == "neutralni":
            return random.choice([True, False])
        elif self.bias == "složeni_1":
            return (x[0] > 0.5) != (x[1] > 0.5)  # Približno XOR
        elif self.bias == "složeni_2":
            return abs(x[0] - x[1]) > 0.3
        elif self.bias == "kvadratni_modificiran":
            return (x[0] - 0.5)**2 + (x[1] - 0.5)**2 < 0.3
        else:
            return False

    def evaluiraj(self, test_podaci):
        """Evaluira agenta na test podacima."""
        točnih = 0
        for x, y in test_podaci:
            if self.predviđa(x) == y:
                točnih += 1
        self.fitness = točnih / len(test_podaci)
        return self.fitness

def evolucija_biasa(generacije=20, veličina_populacije=20):
    """Simulira evoluciju induktivnog biasa."""

    # Mogući biasi
    biasi = ["linearni", "kvadratni", "neutralni",
             "složeni_1", "složeni_2", "kvadratni_modificiran"]

    # Stvori početnu populaciju
    populacija = [Agent(random.choice(biasi)) for _ in range(veličina_populacije)]

    # Test podaci (XOR problem)
    test_podaci = [
        (np.array([0.2, 0.2]), False),
```

```
        (np.array([0.2, 0.8]), True),
        (np.array([0.8, 0.2]), True),
        (np.array([0.8, 0.8]), False),
        (np.array([0.5, 0.1]), False),
        (np.array([0.1, 0.5]), False),
        (np.array([0.9, 0.5]), True),
        (np.array([0.5, 0.9]), True),
        (np.array([0.4, 0.4]), False),
        (np.array([0.6, 0.6]), False),
        (np.array([0.3, 0.7]), True),
        (np.array([0.7, 0.3]), True),
    ]

    print("Simulacija evolucije induktivnog biasa:")
    print("="*41)
    print(f"\nInicijalna populacija: {veličina_populacije} agenata s različitim biasima")
    print("Okoliš: jednostavan XOR svijet")
    print("\nEvolucija kroz generacije:")
    print("-"*27)

    povijest = []

    for gen in range(generacije):
        # Evaluiraj sve agente
        for agent in populacija:
            agent.evaluiraj(test_podaci)

        # Sortiraj po fitness-u
        populacija.sort(key=lambda a: a.fitness, reverse=True)

        # Zapisivanje najboljih
        if gen % 5 == 0 or gen == generacije - 1:
            najbolji = populacija[0]
            print(f"Generacija {gen+1}: Najbolji bias = {najbolji.bias} "
                  f"({najbolji.fitness*100:.1f}% točnosti)")

        # Selekcija i reprodukcija (elitizam + turnir)
        nova_populacija = populacija[:5]  # Zadrži najboljih 5

        while len(nova_populacija) < veličina_populacije:
            # Turnirska selekcija
            turnir = random.sample(populacija[:10], 2)
            pobjednik = max(turnir, key=lambda a: a.fitness)

            # Stvori novog agenta s mogućom mutacijom
            if random.random() < 0.1:  # 10% šanse za mutaciju
                novi_bias = random.choice(biasi)
            else:
                novi_bias = pobjednik.bias

            nova_populacija.append(Agent(novi_bias))

        populacija = nova_populacija
```

```
102
103     # Finalna statistika
104     print("\nDistribucija biasa u finaliznoj populaciji:")
105     print("-"*45)
106
107     bias_count = {}
108     for agent in populacija:
109         bias_count[agent.bias] = bias_count.get(agent.bias, 0) + 1
110
111     for bias, count in sorted(bias_count.items(), key=lambda x: x[1], reverse=True):
112         postotak = 100 * count / veličina_populacije
113         print(f"{bias}: {postotak:.1f}%")
114
115     print("\nZaključak: Evolucija prirodno selektira biase")
116     print("koji odgovaraju strukturi okoliša!")
117
118 # Pokreni simulaciju
119 evolucija_biasa()
```

Output

```
Simulacija evolucije induktivnog biasa:
========================================

Inicijalna populacija: 20 agenata s različitim biasima
Okoliš: jednostavan XOR svijet

Evolucija kroz generacije:
--------------------------
Generacija 1: Najbolji bias = kvadratni (83.3% točnosti)
Generacija 6: Najbolji bias = složeni_1 (100.0% točnosti)
Generacija 11: Najbolji bias = složeni_1 (100.0% točnosti)
Generacija 16: Najbolji bias = složeni_1 (100.0% točnosti)
Generacija 20: Najbolji bias = složeni_1 (100.0% točnosti)

Distribucija biasa u finaliznoj populaciji:
--------------------------------------------
složeni_1: 95.0%
složeni_2: 5.0%

Zaključak: Evolucija prirodno selektira biase
koji odgovaraju strukturi okoliša!
```

9.1.8 Praktične implikacije i rješenja

Kako se nositi s Goodmanovim problemom u praksi?

1. **Regularizacija** - ograničava složenost hipoteza

2. **Križna validacija** - testira generalizaciju
3. **Occamova oštrica** - preferira jednostavnije hipoteze
4. **Domensko znanje** - koristi ljudsko znanje o problemu
5. **Ansambl metode** - kombinira različite biase

```python
def occamova_oštrica(hipoteze):
    """Odabire najjednostavniju hipotezu."""
    # Definiraj složenost kao broj parametara/prijelaza
    složenosti = {}
    for naziv, hip in hipoteze.items():
        if naziv == "standard":
            složenosti[naziv] = 1  # Najjednostavnija
        else:
            # Grue hipoteze imaju dodatni parametar (vrijeme prijelaza)
            složenosti[naziv] = 2

    # Odaberi hipotezu s najmanjom složenošću
    najjednostavnija = min(složenosti.items(), key=lambda x: x[1])
    return najjednostavnija[0], složenosti

def bayesov_pristup(hipoteze, opažanja, priori):
    """Koristi Bayesovo zaključivanje s priorima."""
    posteriori = {}

    for naziv, hip in hipoteze.items():
        # Likelihood - sve hipoteze objašnjavaju podatke jednako dobro
        likelihood = 1.0  # Pojednostavljeno

        # Posterior proporcionalan je prior * likelihood
        posteriori[naziv] = priori[naziv] * likelihood

    # Normaliziraj
    ukupno = sum(posteriori.values())
    for naziv in posteriori:
        posteriori[naziv] /= ukupno

    # Odaberi hipotezu s najvećim posteriorom
    najbolja = max(posteriori.items(), key=lambda x: x[1])
    return najbolja[0], posteriori

def ansambl_metoda(hipoteze, težine):
    """Kombinira predviđanja više hipoteza."""
    test_vremena = [
        datetime(2025, 1, 1),
        datetime(2040, 1, 1),
        datetime(2060, 1, 1),
        datetime(2110, 1, 1)
    ]

```

```
    predviđanja = {}
    for t in test_vremena:
        glasovi = {"zelen": 0, "plav": 0}

        for naziv, hip in hipoteze.items():
            pred = hip.predviđa(t)
            glasovi[pred] += težine[naziv]

        predviđanja[t.year] = glasovi

    return predviđanja

# Demonstracija
print("Praktična rješenja za Goodmanov problem:")
print("="*41)
print("\nTest različitih pristupa na 'grue' problemu:")

# Pripremi hipoteze
test_hipoteze = {
    "standard": hipoteze["standard"],
    "grue_2030": hipoteze["grue_2030"],
    "grue_2050": hipoteze["grue_2050"],
    "grue_2100": hipoteze["grue_2100"]
}

# 1. Bez regularizacije
print("\n1. Bez regularizacije (prihvaća sve hipoteze):")
print(f"   Razmatrane hipoteze: {', '.join(test_hipoteze.keys())}")
print("   Odabrana: standard (proizvoljan izbor)")
print("   Problem: Nema kriterija za izbor!")

# 2. Occamova oštrica
print("\n2. Occamova oštrica (preferira jednostavnije):")
najbolja_occam, složenosti = occamova_oštrica(test_hipoteze)
print("   Složenost hipoteza:")
for naziv, slož in sorted(složenosti.items(), key=lambda x: x[1]):
    komentar = " (najjednostavnija)" if slož == 1 else ""
    print(f"     {naziv}: {slož}{komentar}")
print(f"   Odabrana: {najbolja_occam}")
print("   Razlog: Najniža složenost")

# 3. Bayesov pristup
print("\n3. Bayesov pristup (koristi priore):")
priori = {
    "standard": 0.7,  # Visok prior za standardnu hipotezu
    "grue_2030": 0.1,
    "grue_2050": 0.1,
    "grue_2100": 0.1
}
najbolja_bayes, posteriori = bayesov_pristup(test_hipoteze, opažanja_zeleni, priori)

print("   Prior vjerojatnosti:")
for naziv, p in priori.items():
```

```
    print(f"     {naziv}: {p:.3f}")
print("   Posterior (nakon opažanja):")
for naziv, p in posteriori.items():
    print(f"     {naziv}: {p:.3f}")
print(f"   Odabrana: {najbolja_bayes}")
print("   Razlog: Najviši posterior")

# 4. Ansambl metoda
print("\n4. Ansambl metoda (kombinira hipoteze):")
težine = {
    "standard": 0.7,
    "grue_2030": 0.1,
    "grue_2050": 0.1,
    "grue_2100": 0.1
}
ansambl_pred = ansambl_metoda(test_hipoteze, težine)
print("   Težinski prosjek predviđanja:")
for godina, glasovi in ansambl_pred.items():
    ukupno = sum(glasovi.values())
    postotak_zelen = 100 * glasovi["zelen"] / ukupno
    postotak_plav = 100 * glasovi["plav"] / ukupno
    print(f"   Za {godina}: {postotak_zelen:.1f}% zeleno, {postotak_plav:.1f}% plavo")
print("   Predviđanje: Ponderirana kombinacija")

print("\nZaključak: Praktična rješenja koriste dodatne")
print("kriterije izvan čiste logike za izbor hipoteza.")
```

Output

```
Praktična rješenja za Goodmanov problem:
========================================

Test različitih pristupa na 'grue' problemu:

1. Bez regularizacije (prihvaća sve hipoteze):
   Razmatrane hipoteze: standard, grue_2030, grue_2050, grue_2100
   Odabrana: standard (proizvoljan izbor)
   Problem: Nema kriterija za izbor!

2. Occamova oštrica (preferira jednostavnije):
   Složenost hipoteza:
     standard: 1 (najjednostavnija)
     grue_2030: 2
     grue_2050: 2
     grue_2100: 2
   Odabrana: standard
   Razlog: Najniža složenost

3. Bayesov pristup (koristi priore):
   Prior vjerojatnosti:
     standard: 0.700
```

```
      grue_2030: 0.100
      grue_2050: 0.100
      grue_2100: 0.100
    Posterior (nakon opažanja):
      standard: 0.700
      grue_2030: 0.100
      grue_2050: 0.100
      grue_2100: 0.100
    Odabrana: standard
    Razlog: Najviši posterior

4. Ansambl metoda (kombinira hipoteze):
    Težinski prosjek predviđanja:
    Za 2025: 100.0% zeleno, 0.0% plavo
    Za 2040: 90.0% zeleno, 10.0% plavo
    Za 2060: 80.0% zeleno, 20.0% plavo
    Za 2110: 70.0% zeleno, 30.0% plavo
    Predviđanje: Ponderirana kombinacija

Zaključak: Praktična rješenja koriste dodatne
kriterije izvan čiste logike za izbor hipoteza.
```

9.1.9 Zaključak

Kroz ovu bilježnicu istražili smo duboku vezu između **Goodmanovog novog problema indukcije** i **No-Free-Lunch teorema** u strojnom učenju:

Ključni uvidi:

1. **Beskonačnost hipoteza**: Za svaki skup podataka postoji beskonačno mnogo jednako dobro podržanih hipoteza
2. **Nužnost biasa**: Bez induktivnog biasa, učenje je nemoguće - to nije bug već feature!
3. **NFL kao formalizacija**: No-Free-Lunch teoremi su matematička formalizacija Goodmanovog filozofskog argumenta
4. **Evolucija i bias**: Ljudska sposobnost učenja možda uspijeva zbog evolucijski oblikovanih biasa
5. **Praktična rješenja**: Regularizacija, Occamova oštrica i Bayesov pristup nude načine izbora među hipotezama

Filozofske implikacije:

Goodmanov problem pokazuje da **čista logika nije dovoljna** za induktivno zaključivanje. Trebamo dodatne kriterije - jednostavnost, priore, domensko znanje - koji nisu čisto logički.

Implikacije za AI:

Svaki sustav umjetne inteligencije mora riješiti Goodmanov problem implicitnim ili eksplicitnim izborom induktivnog biasa. **Nema univerzalnog algoritma učenja** - uspjeh ovisi o podudaranju između biasa algoritma i strukture problema.

Kao što Goodman zaključuje:

> "Valjanost induktivnog zaključka nije stvar logike već stvar povijesti uporabe predikata."

Možda je ključ uspješnog strojnog učenja u tome da naučimo kako odabrati prave biase za prave probleme - lekcija koju evolucija uči već milijunima godina.