פרק 9: מימד, קשר ותבנית – יסודות ליניאריים לאינטליגנציה מלאכותית

Chapter 9: Dimension, Correlation, and Pattern – Linear Foundations for Artificial Intelligence

ד"ר יורם סגל

ספטמבר 2025

תוכן העניינים

1 מבוא: עולמו הנסתר של הווקטור

Introduction: The Hidden World of the Vector

בפרק זה נחקור את הבסיס הפילוסופי והמתמטי של הייצוג הווקטורי בבינה מלאכותית. נראה כיצד כל יחידת מידע – תמונה, מילה, או חולה במרפאה – הופכת לנקודה במרחב רב-ממדי, וכיצד מושג זה מהווה את היסוד של כל AI מודרני.

1.1 מבט ראשון: מה רואים כשמביטים בתמונה?

כשאתם מביטים בתמונה של חתול, מה המוח שלכם באמת רואה? האם הוא רואה יצור פרוותי חמוד עם שפם? או שמא הוא רואה משהו עמוק יותר – רשת מורכבת של 1000000 נקודות אור, כל אחת מהן נושאת מספר המייצג עוצמת בהירות?

זוהי השאלה המרכזית שמניעה את כל תחום הבינה המלאכותית המודרני: **כיצד ניתן להפוך מידע לווקטור, ואיך ווקטורים הופכים למשמעות?**

1.2 דקארט והמצאת השפה המספרית של המרחב

ב-1637, כשרנה דקארט (René Descartes) פרסם את ספרו "La Géométrie", הוא לא יכול היה לדמיין שהרעיון הפשוט שלו – לייצג נקודה במרחב באמצעות מספרים – יהפוך ליסוד של מהפכה טכנולוגית שתגיע 400 שנה מאוחר יותר [1].

דקארט המציא את מערכת הצירים הקרטזית, ועמה את הרעיון המהפכני שכל נקודה במרחב ניתנת לתיאור מספרי. אך רק במאה ה-20, כשמדעני המחשב החלו לחשוב על דרכים לייצג מידע, הבינו שהרעיון של דקארט הוא לא רק כלי גיאומטרי – **הוא שפה אוניברסלית לתיאור המציאות**.

שאלה למחשבה: אם דקארט יכול היה לייצג נקודה במרחב תלת-ממדי עם 3 מספרים, כמה מספרים נדרשים כדי לייצג תמונת חתול?

1.3 הווקטור: לא רק חץ במרחב

בקורסים מתמטיים מסורתיים, וקטור מוגדר לרוב כ"בחרמב ץח" – יש לו כיוון ואורך. אך בעולם מדעי הנתונים ולמידת המכונה, הווקטור הוא משהו עמוק יותר בהרבה: **הוא ייצוג של ישות במרחב מופשט** [1].

דוגמה מהחיים – חולה במרפאה:

אם נתון לכם חולה במרפאה, איך תתארו אותו כווקטור? התשובה: כל תכונה נמדדת של החולה – גיל, משקל, לחץ דם, רמת סוכר, טמפרטורה – הופכת ל-tnenopmoc אחד בווקטור. אם יש לנו 50 תכונות נמדדות, החולה מיוצג כנקודה במרחב בעל 50 ממדים:

$$\vec{p}_{\text{patient}} = \begin{bmatrix} \text{age} \\ \text{weight} \\ \text{blood\_pressure} \\ \text{glucose} \\ \vdots \\ \text{feature}_{50} \end{bmatrix} \in \mathbb{R}^{50}$$

הסימון $\mathbb{R}^{50}$ משמעותו "םידממ 50 לש ידילקוא בחרמ" – מרחב שבו כל נקודה מוגדרת על ידי 50 מספרים ממשיים. זהו המרחב שבו **כל החולים שלנו חיים**, במובן מתמטי.

1.4 מתמונת חתול לווקטור: תהליך הוקטוריזציה

כפי שהדגיש ד"ר יורם סגל בהרצאתו, תמונה של 1000×1000 פיקסלים בצבע RGB מכילה 3×10^6 ערכים מספריים (כל פיקסל מיוצג על ידי 3 ערכים: אדום, ירוק, כחול).

המתמטיקה מאחורי הוקטוריזציה:

כדי להפוך אותה לייצוג וקטורי הניתן לניתוח על ידי מודל למידת מכונה, אנו מבצעים תהליך של "Flattening" (השטחה). המטריצה המבנית מפורקת לרצף של מספרים, היוצר וקטור באורך של שלושה מיליון.

אם $\mathbf{I} \in \mathbb{R}^{h \times w \times c}$ היא תמונה בגובה h, רוחב w ו-c ערוצי צבע, אזי הוקטוריזציה מוגדרת כ:

$$\text{vec}(\mathbf{I}) = \mathbf{v} \in \mathbb{R}^{h \cdot w \cdot c} \tag{1}$$

כאשר $\mathbf{v} = [I_{1,1,1}, I_{1,1,2}, \ldots, I_{h,w,c}]^T$ – רצף של כל הפיקסלים בזה אחר זה.

משמעות עמוקה: תמונת החתול הפכה לנקודה אחת ויחידה במרחב בעל שלושה מיליון ממדים. כל תמונה שונה במעט תהיה נקודה אחרת באותו מרחב עצום.

1.5 מכפלה סקלרית: מדד הדמיון האולטימטיבי

אחת הפעולות החשובות ביותר על וקטורים היא **מכפלה סקלרית** (Dot Product, Inner Product). היא מוגדרת כך:

הגדרה 1.1 – מכפלה סקלרית (אלגברית):

$$\langle \vec{u}, \vec{v} \rangle = \sum_{i=1}^{n} u_i v_i = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n \tag{2}$$

שאלה מהותית שהעלה המרצה: מה משמעות המכפלה הסקלרית? למה היא כל כך חשובה?

התשובה מגיעה מנוסחה חלופית, גיאומטרית, שגילה המתמטיקאי הגרמני Hermann von Helmholtz במאה ה-19:

$$\langle \vec{u}, \vec{v} \rangle = \|\vec{u}\| \cdot \|\vec{v}\| \cdot \cos(\theta) \tag{3}$$

כאשר θ היא הזווית בין שני הווקטורים, ו-$\|\vec{u}\|$ היא **הנורמה** (Norm) או אורך הווקטור:

$$\|\vec{u}\| = \sqrt{u_1^2 + u_2^2 + \cdots + u_n^2} = \sqrt{\sum_{i=1}^{n} u_i^2} \tag{4}$$

1.6 הוכחה: שתי ההגדרות זהות

משפט 1.1: ההגדרה האלגברית (משוואה 1.2) וההגדרה הגיאומטרית (משוואה 1.3) של מכפלה סקלרית שוות.

הוכחה:

צעד 1: משפט הקוסינוסים.

במשולש עם צלעות $\|\vec{u}\|$, $\|\vec{v}\|$ וזווית θ ביניהן, הצלע השלישית היא $\|\vec{u} - \vec{v}\|$. לפי משפט הקוסינוסים:

$$\|\vec{u} - \vec{v}\|^2 = \|\vec{u}\|^2 + \|\vec{v}\|^2 - 2\|\vec{u}\|\|\vec{v}\| \cos(\theta)$$

צעד 2: פיתוח אלגברי של האגף השמאלי.

נפתח את $\|\vec{u} - \vec{v}\|^2$ באמצעות ההגדרה הא לגברית:

$$\begin{aligned}
\|\vec{u} - \vec{v}\|^2 &= \sum_{i=1}^{n} (u_i - v_i)^2 \\
&= \sum_{i=1}^{n} (u_i^2 - 2u_i v_i + v_i^2) \\
&= \sum_{i=1}^{n} u_i^2 + \sum_{i=1}^{n} v_i^2 - 2\sum_{i=1}^{n} u_i v_i \\
&= \|\vec{u}\|^2 + \|\vec{v}\|^2 - 2\langle \vec{u}, \vec{v} \rangle \qquad (5)
\end{aligned}$$

צעד 3: השוואת הביטויים.

מצעד 1: $\|\vec{u} - \vec{v}\|^2 = \|\vec{u}\|^2 + \|\vec{v}\|^2 - 2\|\vec{u}\|\|\vec{v}\| \cos(\theta)$

מצעד 2: $\|\vec{u} - \vec{v}\|^2 = \|\vec{u}\|^2 + \|\vec{v}\|^2 - 2\langle \vec{u}, \vec{v} \rangle$

ביטול $\|\vec{u}\|^2 + \|\vec{v}\|^2$ משני האגפים מוביל ל:

$$2\langle \vec{u}, \vec{v} \rangle = 2\|\vec{u}\| \cdot \|\vec{v}\| \cdot \cos(\theta)$$

חלוקה ב-2 נותנת:

- $\langle \vec{u}, \vec{v} \rangle = \|\vec{u}\| \cdot \|\vec{v}\| \cdot \cos(\theta)$

משמעות עמוקה: מכפלה סקלרית היא **מדד דמיון גיאומטרי**.

- אם $\theta = 0°$ (וקטורים באותו כיוון): $\cos(\theta) = 1$ והמכפלה מקסימלית ← **דמיון מלא**

- אם $\theta = 90°$ (וקטורים אורתוגונליים): $\cos(\theta) = 0$ והמכפלה מתאפסת ← **אין דמיון כלל**

- אם $\theta = 180°$ (וקטורים מנוגדים): $\cos(\theta) = -1$ והמכפלה מינימלית ← **ניגוד מלא**

1.7 קשר ל-IA: Cosine Similarity ומנועי חיפוש

מנועי חיפוש מודרניים כמו Google ומערכות המלצה כמו Netflix משתמשים בווריאציה על מכפלה סקלרית שנקראת **Cosine Similarity** – דמיון קוסינוס:

$$\text{similarity}(\vec{u}, \vec{v}) = \frac{\langle \vec{u}, \vec{v} \rangle}{\|\vec{u}\| \cdot \|\vec{v}\|} = \cos(\theta) \tag{6}$$

חישוב זה **מנרמל** את המכפלה הסקלרית כך שהתוצאה תמיד בין -1 ל-$+1$, ללא תלות באורכי הווקטורים.

למה נורמליזציה חשובה?

בלי נורמליזציה, וקטור ארוך יקבל מכפלה סקלרית גדולה יותר רק בגלל האורך, לא בגלל הדמיון. Cosine Similarity פותר זאת על ידי מדידת הזווית בלבד.

דוגמה מהחיים – מנוע חיפוש:

כשאתם מקלידים שאילתה "machine learning tutorial", מנוע החיפוש:

1. **ממיר את השאילתה לווקטור** $\vec{q}$: כל מילה מקבלת משקל (למשל, באמצעות TF-IDF או Word2Vec [2])

2. **משווה את** $\vec{q}$ **לכל מסמך** $\vec{d_i}$ במאגר באמצעות Cosine Similarity

3. **מחזיר את המסמכים** עם הדמיון הגבוה ביותר (ציון קוסינוס הכי קרוב ל-1)

פונקציות NumPy למכפלה סקלרית ונורמה:

טבלה 1: פונקציות NumPy למכפלה סקלרית, נורמה ו-Cosine Similarity

Function	תפקיד	שימוש והסבר
np.dot(u, v)	מכפלה סקלרית	מחשבת $\sum u_i v_i$ – הליבה של כל חישוב דמיון
np.linalg.norm(u)	נורמה (L2)	מחשבת $\sqrt{\sum u_i^2}$ – אורך הווקטור
@ (אופרטור)	מכפלת מטריצות	תחביר קצר: u @ v זהה ל-np.dot(u, v)

הערה חשובה: פונקציות אלה הן הבסיס של כל אלגוריתם AI. נשתמש בהן שוב ושוב לאורך הספר.

פסאודו-קוד – Cosine Similarity:

1.8 Word Embeddings: המהפכה של Word2Vec

אחת ההצלחות המרשימות ביותר של ייצוג וקטורי היא **Word2Vec**, שפותחה על ידי Tomas Mikolov ועמיתיו ב-elgooG ב-2013 [2].

הרעיון המהפכני: לייצג כל מילה בשפה כווקטור בן ~ 300 ממדים, כך ש**מילים דומות במשמעות יהיו קרובות במרחב הווקטורי**.

הפלא המתמטי של Word2Vec:

אחד הממצאים המפורסמים ביותר הוא שחישובים אריתמטיים על וקטורי מילים מניבים תוצאות משמעותיות סמנטית:

$$\vec{\text{king}} - \vec{\text{man}} + \vec{\text{woman}} \approx \vec{\text{queen}}$$

שאלה שמעלה פלא: איך זה אפשרי? איך חיבור וחיסור של מספרים יכול לבטא יחסים סמנטיים?

התשובה: הווקטורים לומדים **יחסים**. הווקטור $\vec{\text{king}} - \vec{\text{man}}$ מייצג את המושג המופשט ”תירכז תוכלמ“, וכשמוסיפים $\vec{\text{woman}}$ מקבלים ”תישנ תוכלמ“ = $\vec{\text{queen}}$.

זה עובד בגלל **הנחת הדיסטריבוציה** (Distributional Hypothesis) שניסח הבלשן Zellig Harris ב-1954 [3]:

> *"Words that occur in similar contexts tend to have similar meanings"*
>
> "מילים שמופיעות בהקשרים דומים נוטות לשאת משמעות דומה"

אלגוריתם Word2Vec לומד וקטורים שמנבאים את הקונטקסט של מילה, ובכך **מקודדים משמעות סמנטית כמרחק גיאומטרי**.

Cosine Similarity – מימוש ממוקד

```python
import numpy as np

def cosine_similarity(u, v):
    """
␣␣␣␣Computes␣Cosine␣Similarity␣between␣two␣vectors.

␣␣␣␣Args:
␣␣␣␣␣␣␣␣u,␣v:␣NumPy␣vectors␣of␣the␣same␣length

␣␣␣␣Returns:
␣␣␣␣␣␣␣␣float:␣similarity␣score␣between␣-1␣and␣1
␣␣␣␣"""
    # Dot product - numerator
    dot_product = np.dot(u, v)

    # Norms - denominator
    norm_u = np.linalg.norm(u)
    norm_v = np.linalg.norm(v)

    # Cosine Similarity
    return dot_product / (norm_u * norm_v)

# Example: two documents (simulated TF-IDF vectors)
doc1 = np.array([1, 2, 3, 0, 0])
doc2 = np.array([1, 1, 0, 4, 5])

print(f"Similarity:␣{cosine_similarity(doc1,␣doc2):.3f}")
# Result: 0.277 - weak similarity
```

1.9 תרגיל תכנות עצמי 1.1 – חיפוש מילים דומות

מטרה: להבין איך Cosine Similarity משמש למציאת מילים דומות במודל Word2Vec.

משימה:

1. טענו וקטורי Word2Vec מוכנים (למשל, מ-misneG או מודל GloVe)
2. בחרו מילה (למשל, "computer")
3. חשבו Cosine Similarity בינה לבין כל המילים האחרות במאגר
4. הציגו את 10 המילים הדומות ביותר

פסאודו-קוד:

חיפוש מילים דומות ב-ceV2droW

```
from gensim.models import KeyedVectors

# טעינת מודל Word2Vec מוכן להוריד (מהאינטרנט)
model = KeyedVectors.load_word2vec_format(
    'GoogleNews-vectors-negative300.bin',
    binary=True
)

# מציאת מילים דומות - הפונקציה משתמשת ב-Cosine Similarity
similar_words = model.most_similar('computer', topn=10)

# Print results
print("Most␣similar␣words␣to␣'computer':")
for word, similarity in similar_words:
    print(f"␣␣{word}:␣{similarity:.3f}")
```

תוצאה צפויה:

- laptop: 0.82
- software: 0.78
- technology: 0.75
- hardware: 0.72
- PC: 0.70

1.10 אזהרה: הסכנה בווקטורים – Bias ודעות קדומות

ב-2016, חוקרים מאוניברסיטת Princeton גילו תופעה מדאיגה: מודלי Word2Vec שאומנו על טקסטים מהאינטרנט הטמיעו **דעות קדומות חברתיות** [4].

דוגמאות להטיות שנמצאו:

- $\vec{\text{man}} : \vec{\text{programmer}}$ כמו $\vec{\text{woman}} : \vec{\text{homemaker}}$
- $\vec{\text{man}} : \vec{\text{doctor}}$ כמו $\vec{\text{woman}} : \vec{\text{nurse}}$
- קשרים חזקים יותר בין שמות אירופאים למילים חיוביות לעומת שמות אפרו-אמריקאים

המודל למד שגברים קשורים למקצועות טכנולוגיים, ונשים למקצועות ביתיים – לא משום שזו אמת, אלא כי **כך התייחסו הטקסטים שממנו למד**.

שאלה מוסרית שהעלה ד"ר סגל: אם AI לומד מהעולם, והעולם מוטה – האם AI מחזק את המוטיות?

התשובה המחקרית: כן, אלא אם כן מתערבים באופן פעיל. זו אחת המשימות המרכזיות של תחום **Fairness in AI**. חוקרים כמו Tolga Bolukbasi ועמיתיו פיתחו שיטות ל**ניטרול הטיות** (Debiasing) בווקטורי מילים [5].

שיטת הניטרול:

השיטה מבוססת על זיהוי **כיווני הטיה** במרחב הווקטורי. למשל, הכיוון $\vec{d}_{\text{gender}} = \vec{\text{he}} - \vec{\text{she}}$ מייצג את ממד המגדר. לאחר מכן, מסירים את המרכיב הזה ממילים שאמורות להיות ניטרליות:

$$\vec{\text{doctor}}_{\text{debiased}} = \vec{\text{doctor}} - \text{proj}_{\vec{d}_{\text{gender}}}(\vec{\text{doctor}})$$

כאשר $\text{proj}_{\vec{d}}(\vec{v})$ היא ההטלה של $\vec{v}$ על הכיוון $\vec{d}$.

מסקנה: ייצוג וקטורי הוא כלי עוצמתי, אך הוא משקף את הנתונים שממנו למד. **אחריות החוקר היא להבטיח שהמודל לא מנציח אפליה.**

1.11 מרחבים וקטוריים: הפורמליזם המתמטי

עד כה דיברנו על וקטורים בצורה אינטואיטיבית. אך מה באמת הופך אוסף של מספרים לווקטור? התשובה מגיעה מ**תורת המרחבים הווקטוריים** (Vector Space Theory), שפותחה במאה ה-19 על ידי מתמטיקאים כמו Hermann Grassmann (1844) ו-onaeP eppesuiG (1888).

הגדרה 1.2 – מרחב וקטורי:

מרחב וקטורי V מעל שדה $\mathbb{F}$ (בדרך כלל $\mathbb{R}$ או $\mathbb{C}$) הוא קבוצה עם שתי פעולות:

1. **חיבור וקטורים:** $\vec{u} + \vec{v} \in V$ לכל $\vec{u}, \vec{v} \in V$
2. **כפל בסקלר:** $\alpha\vec{u} \in V$ לכל $\alpha \in \mathbb{F}$ ו-$\vec{u} \in V$

שמקיימות 8 אקסיומות:

טבלה 2: אקסיומות מרחב וקטורי

אקסיומה	**תכונה**
1	קומוטטיביות: $\vec{u}+\vec{v}=\vec{v}+\vec{u}$
2	אסוציאטיביות: $(\vec{u}+\vec{v})+\vec{w}=\vec{u}+(\vec{v}+\vec{w})$
3	קיום איבר אפס: קיים $\vec{0}$ כך ש-$\vec{u}+\vec{0}=\vec{u}$
4	קיום איבר נגדי: לכל $\vec{u}$ קיים $\vec{u}-$ כך ש-$\vec{u}+(-\vec{u})=\vec{0}$
5	כפל באחד: $1\cdot\vec{u}=\vec{u}$
6	דיסטריבוטיביות: $\alpha(\vec{u}+\vec{v})=\alpha\vec{u}+\alpha\vec{v}$
7	דיסטריבוטיביות: $(\alpha+\beta)\vec{u}=\alpha\vec{u}+\beta\vec{u}$
8	אסוציאטיביות כפל: $(\alpha\beta)\vec{u}=\alpha(\beta\vec{u})$

למה זה חשוב ל-IA?

ברגע שהוכחנו שאוסף של ישויות (תמונות, מסמכים, חולים) יוצר מרחב וקטורי, אנחנו יכולים להשתמש ב**כל הכלים של אלגברה ליניארית**:

- פירוק SVD (Singular Value Decomposition)
- ערכים עצמיים (Eigenvalues) ווקטורים עצמיים (Eigenvectors)
- הטלות (Projections) והטרנספורמציות ליניאריות
- PCA (Principal Component Analysis) להפחתת ממדיות

כל אלו הם כלים שנשתמש בהם לאורך הספר לניתוח ועיבוד נתונים.

1.12 המעבר למימד גבוה: ברכה או קללה?

ב-1961, הסטטיסטיקאי והמתמטיקאי Richard Bellman טבע את המונח המפורסם **"Curse of Dimensionality"** – קללת המימדיות [6].

הבעיה המתמטית:
Bellman הבין שככל שמוסיפים תכונות (ממדים) לנתונים, **נפח המרחב גדל באופן אקספוננציאלי**, והנתונים הופכים ספרסיים (Sparse) יותר ויותר.
דוגמה מספרית שהדגים המרצה:
נניח שאנחנו רוצים לדגום 10 נקודות לאורך כל ממד כדי לכסות את המרחב בצפיפות סבירה:

- ב-1 ממד: $10^1 = 10$ נקודות
- ב-2 ממדים: $10^2 = 100$ נקודות
- ב-3 ממדים: $10^3 = 1000$ נקודות
- ב-10 ממדים: $10^{10} = 10000000000$ נקודות (עשרה מיליארד!)
- ב-100 ממדים: 10^{100} – מספר גדול יותר מכמות האטומים ביקום!

שאלת המפתח שהעלה ד"ר סגל:
אם יש לנו תמונה חתול 1000×1000 פיקסלים (מיליון תכונות), כמה דוגמאות אימון נדרשות כדי לאמן מודל Brute Force?
התשובה המתמטית:
לפי כלל האצבע שנידון בהרצאה:

- **למידת מכונה קלאסית:** מינימום 30 דוגמאות לכל תכונה
- **למידה עמוקה:** מינימום 100 דוגמאות לכל תכונה

עבור מיליון תכונות: $100 \times 10^6 = 100000000$ דוגמאות (מאה מיליון!)
הפתרון – רשתות CNN:
זו הסיבה שרשתות Convolutional Neural Networks (CNN) חיוניות [7], [8]. הן מפחיתות באופן דרמטי את מספר הפרמטרים באמצעות:

1. **שיתוף משקלים** (Weight Sharing): אותו פילטר משמש בכל התמונה
2. **קישוריות מקומית** (Local Connectivity): כל נוירון מחובר רק לאזור קטן
3. **היררכיה של תכונות:** למידה הדרגתית מתכונות פשוטות (קצוות) למורכבות (פנים)

הישג מפורסם: רשת AlexNet של Krizhevsky, Sutskever ו-notniH (2012) אימנה על 1.2 מיליון תמונות עם ~ 60 מיליון פרמטרים, והשיגה פריצת דרך בסיווג תמונות [8].
זו תהיה נקודת המוצא לפרק הבא, שבו נצלול לעומקה של קללת המימדיות ונראה כיצד היא משפיעה על כל אלגוריתם למידה.

1.13 תרגיל תכנות עצמי 1.2 – השוואת מרחקים בממדים שונים

מטרה: להמחיש את קללת המימדיות באופן מעשי.

משימה:

1. צרו 100 נקודות אקראיות במרחבים בממדים שונים: 2, 10, 100, 1000
2. חשבו את המרחק האוקלידי בין כל זוג נקודות
3. חשבו את ממוצע המרחקים ואת סטיית התקן
4. הציגו גרף: ציר X = מימד, ציר Y = יחס סטיית תקן/ממוצע

פסאודו-קוד:

תוצאה צפויה: היחס שואף ל-0 ככל שהממד גדל – כל הנקודות נעשות "באותו מרחק" זו מזו, ומדדי מרחק מאבדים משמעות.

1.14 סיכום ומבט קדימה

מה למדנו בפרק זה?

1. **וקטור הוא ייצוג מופשט** – לא רק חץ, אלא כל ישות הניתנת לתיאור מספרי (תמונה, מילה, חולה)
2. **מכפלה סקלרית היא מדד דמיון גיאומטרי** – הבסיס לחיפוש, המלצות, ו-PLN
3. **Cosine Similarity הוא הכלי המרכזי** – מנרמל מרחקים ומודד זווית בלבד
4. **Word2Vec הוא דוגמה מבריקה** – מילים הופכות לווקטורים והסמנטיקה הופכת לגיאומטריה
5. **הטיות ב-IA הן בעיה אמיתית** – מודלים לומדים מהעולם, כולל דעות קדומות
6. **מרחבים וקטוריים הם הפורמליזם** – מבטיחים שנוכל להשתמש באלגברה ליניארית
7. **ממד גבוה = אתגר עצום** – קללת המימדיות דורשת ארכיטקטורות חכמות כמו CNN

מבט קדימה – פרק 2:

בפרק הבא נצלול לעומק **הדיכוטומיה של המידע** – מדוע יותר תכונות לא תמיד טוב יותר, ואיך להתמודד עם מרחבים בעלי ממד גבוה. נראה:

- הוכחה מתמטית מלאה של קללת המימדיות
- השפעה על אלגוריתמים: K-Nearest Neighbors, SVM, Decision Trees
- יחס פיצ'ר-דגימה: כמה נתונים באמת צריך?

המחשת קללת המימדיות

```python
import numpy as np
import matplotlib.pyplot as plt

def curse_of_dimensionality_demo(dims, n_points=100):
    """
␣␣␣␣Demonstrates␣the␣curse␣of␣dimensionality␣by␣computing␣distances.
␣␣␣␣"""
    results = []

    for d in dims:
        # Create random points
        points = np.random.rand(n_points, d)

        # Compute all distances
        distances = []
        for i in range(n_points):
            for j in range(i+1, n_points):
                dist = np.linalg.norm(points[i] - points[j])
                distances.append(dist)

        # Statistics
        mean_dist = np.mean(distances)
        std_dist = np.std(distances)
        ratio = std_dist / mean_dist  # When this approaches 0, the
curse is strong

        results.append(ratio)
        print(f"Dim={d}:␣Mean={mean_dist:.3f},␣Std={std_dist:.3f},␣Ratio
={ratio:.3f}")

    return results

# Run
dimensions = [2, 5, 10, 50, 100, 500, 1000]
ratios = curse_of_dimensionality_demo(dimensions)

# Plot
plt.plot(dimensions, ratios, marker='o')
plt.xlabel('Number␣of␣Dimensions')
plt.ylabel('Std/Mean␣Ratio')
plt.title('Curse␣of␣Dimensionality')
plt.grid(True)
plt.show()
```

- פתרונות: Feature Selection, PCA, Regularization

שאלת מחשבה לסיום:
אם תמונת חתול 1000×1000 היא נקודה במרחב מיליון-ממדי, וכמעט כל "הזזה" קטנה היא אקסטרפולציה – איך מודלים גנרטיביים כמו DALL-E מצליחים ליצור תמונות חדשות שנראות הגיוניות?
התשובה תחכה לפרק 5 על קונוולוציה ורשתות CNN.

מטלות וקריאה מורחבת

תרגיל 1.1: מצאו מילים דומות באמצעות Word2Vec (ראו פסאודו-קוד).
תרגיל 1.2: המחישו את קללת המימדיות (ראו פסאודו-קוד).
תרגיל 1.3: חשבו ידנית מכפלה סקלרית, נורמה, ו-ytiralimiS enisoC בין:

$$\vec{u} = [1, 2, 3], \quad \vec{v} = [4, 5, 6]$$

פתרון: $\langle \vec{u}, \vec{v} \rangle = 32$, $\|\vec{u}\| = \sqrt{14}$, $\|\vec{v}\| = \sqrt{77}$, דמיון $= 0.975 =$
קריאה מורחבת:

- [1] – Linear Algebra and Learning from Data, פרקים 1-2: יסודות אלגברה ליניארית
- [2] – המאמר המקורי על Word2Vec: "Efficient Estimation of Word Representations in Vector Space"
- [4] – מחקר על הטיות במודלי שפה: "Semantics Derived Automatically from Language Corpora Contain Human-like Biases"
- [9] – Deep Learning, פרק 5: למידת מכונה בסיסית

שאלות להעמקה:

1. מדוע Cosine Similarity עדיף על מרחק אוקלידי במרחבים רב-ממדיים?
2. האם ניתן להסיר לחלוטין הטיות ממודלי Word2Vec? מה המחיר?
3. איך רשתות CNN "מרמות" את קללת המימדיות?

סיום פרק 1

2 הדיכוטומיה של המידע: קללת המימדיות

The Information Dichotomy: Curse of Dimensionality

בפרק זה נחקור את אחד האתגרים המרכזיים של למידת מכונה: הפרדוקס שבו הוספת מידע (תכונות) עלולה להחליש את המודל במקום לחזק אותו. נבחן את היסודות המתמטיים של קללת המימדיות, נוכיח את השפעותיה, ונראה כיצד היא משפיעה על אלגוריתמים שונים.

2.1 הפרדוקס: מתי "יותר" זה "פחות"?

דמיינו רופא שמנסה לאבחן מחלה. בתחילה, יש לו שלוש תכונות: טמפרטורה, דופק, ולחץ דם. המודל שלו עובד היטב. לפתע, הוא מקבל גישה למעבדה מתקדמת שמודדת 1000 תכונות ביוכימיות. אינטואיטיבית, המידע הנוסף אמור לשפר את האבחנה. אך במציאות, דיוק המודל יורד.

למה זה קורה?

התשובה טמונה בתופעה מתמטית מפתיעה שגילה Richard Bellman ב-1957 [6]: ככל שמספר הממדים גדל, המרחב "מתנפח" באופן אקספוננציאלי, והנתונים הופכים דלילים (Sparse) – כמו כוכבים ביקום המתרחב.

2.2 בלמן והולדת המושג

Richard Ernest Bellman (1920–1984) היה מתמטיקאי אמריקאי שתרם תרומות מהפכניות לתחומי תכנות דינמי, תורת הבקרה, ולמידת מכונה. בספרו "Dynamic Programming" (1957), הוא זיהה בעיה מהותית: **מורכבות חישובית גדלה באופן אקספוננציאלי עם מספר המשתנים**.

התובנה המרכזית של בלמן:

במרחב חד-ממדי, אם רוצים לכסות קטע באורך 1 ברשת של נקודות במרווח ϵ, נדרשות $1/\epsilon \sim$ נקודות. אך במרחב d-ממדי, נדרשות $(1/\epsilon)^d \sim$ נקודות – גידול אקספוננציאלי [6].

דוגמה מספרית:

טבלה 3: מספר הנקודות הנדרש לכיסוי מרחב [0,1] בצפיפות $\epsilon = 0.1$

ממד	נקודות נדרשות	סדר גודל
1	$10^1 = 10$	עשרות
2	$10^2 = 100$	מאות
3	$10^3 = 1000$	אלפים
10	10^{10}	עשרה מיליארד
100	10^{100}	גוגול (יותר מאטומי היקום)
1000000	$10^{1000000}$	בלתי נתפס

מסקנה: תמונת חתול 1000×1000 פיקסלים (מיליון ממדים) דורשת מספר דגימות שהוא מעבר ליכולת חישובית של כל מחשבי העולם ביחד.

2.3 הוכחה מתמטית: התרחקות מהמרכז

נוכיח תופעה מפתיעה: במרחבים רב-ממדיים, **כמעט כל הנפח מרוכז בקליפה החיצונית**, רחוק מהמרכז.

משפט 2.1 – ריכוז הנפח בקליפה:

יהי כדור יחידה $B_d = \{\vec{x} \in \mathbb{R}^d : \|\vec{x}\| \leq 1\}$ במימד d. נפח הכדור הוא:

$$V_d(r) = \frac{\pi^{d/2}}{\Gamma(d/2+1)} r^d \tag{7}$$

כאשר Γ היא פונקציית הגמא, ו-r הוא הרדיוס.

הוכחה – יחס הנפחים:

נחשב את יחס הנפח של הכדור הפנימי $B_d(0.9)$ (רדיוס 0.9) לכדור המלא $B_d(1)$:

$$\begin{aligned} \frac{V_d(0.9)}{V_d(1)} &= \frac{(0.9)^d \cdot \pi^{d/2}/\Gamma(d/2+1)}{1^d \cdot \pi^{d/2}/\Gamma(d/2+1)} \\ &= (0.9)^d \end{aligned} \tag{8}$$

נחשב עבור ממדים שונים:

טבלה 4: יחס הנפח הפנימי (90% מהרדיוס) לנפח המלא

ממד d	$(0.9)^d$	**אחוז מהנפח**
2	0.81	81%
3	0.729	73%
10	0.349	35%
100	2.66×10^{-5}	0.003%
1000	$\sim 10^{-46}$	כמעט אפס

מסקנה מדהימה: ב-100 ממדים, רק 0.003% מהנפח נמצא ב-90% המרכזיים! כמעט כל הנפח מרוכז בקליפה הדקה בין $r = 0.9$ ל-$r = 1$.

משמעות ל-IA:

אם דגימות האימון שלנו הן "נקודות במרכז", המודל שלנו יצטרך לעשות אקסטרפולציה עצומה כדי לחזות במרחב האמיתי (הקליפה החיצונית) שבו נמצאות רוב ה"תמונות האפשריות".

2.4 הוכחה: התכנסות המרחקים

תופעה נוספת של קללת המימדיות: במרחבים רב-ממדיים, **כל המרחקים בין נקודות הופכים להיות דומים**.

משפט 2.2 – התכנסות יחס המרחקים:

יהיו $\{\vec{x}_i\}_{i=1}^n$ נקודות בלתי-תלויות במרחב $\mathbb{R}^d$. יהיו $d_{\max}$ ו-$d_{\min}$ המרחקים המקסימלי והמינימלי מנקודת מבחן $\vec{q}$. אזי:

$$\lim_{d\to\infty} \frac{d_{\max} - d_{\min}}{d_{\min}} \to 0 \tag{9}$$

הוכחה (סקיצה):

צעד 1: מרחק אוקלידי במימד d בין $\vec{q}$ ל-$\vec{x}_i$:

$$d_i = \|\vec{q} - \vec{x}_i\| = \sqrt{\sum_{j=1}^{d} (q_j - x_{ij})^2}$$

צעד 2: אם המרכיבים $(q_j - x_{ij})$ הם משתנים אקראיים בלתי-תלויים עם תוחלת μ ושונות

σ^2, אזי לפי חוק המספרים הגדולים:

$$\frac{1}{d}\sum_{j=1}^{d}(q_j - x_{ij})^2 \xrightarrow{P} \mu^2 + \sigma^2$$

צעד 3: לכן:

$$d_i = \sqrt{d \cdot (\mu^2 + \sigma^2 + o(1))} = \sqrt{d} \cdot \sqrt{\mu^2 + \sigma^2} \cdot (1 + o(1))$$

צעד 4: כל המרחקים d_i גדלים כ-$\sqrt{d}$, אך ההבדלים ביניהם גדלים לאט יותר (רק כ-$\sqrt{d \cdot \mathrm{Var}}$). לכן:

$$\frac{d_{\max} - d_{\min}}{d_{\min}} \sim \frac{O(\sqrt{d})}{O(\sqrt{d})} \cdot \frac{\mathrm{Var}}{\mathrm{Mean}^2} \to 0 \quad \text{כאשר } d \to \infty$$

■

משמעות מעשית:

במרחב בעל 1000 ממדים, אם הנקודה הקרובה ביותר נמצאת במרחק 100, הנקודה הרחוקה ביותר נמצאת במרחק ~ 100.01 – **כמעט אותו דבר**. מדדי מרחק מאבדים יכולת הבחנה [10], [11].

2.5 השפעה על אלגוריתמים: K-Nearest Neighbors

אלגוריתם K-Nearest Neighbors (KNN), שפותח על ידי Evelyn Fix ו-segdoH hpesoJ ב-1951 [12], הוא אחד האלגוריתמים הפשוטים והאינטואיטיביים ביותר בלמידת מכונה.

הרעיון: כדי לסווג נקודה חדשה $\vec{q}$, מצא את k הנקודות הקרובות ביותר במערך האימון, והצבע לפי רוב.

למה KNN קורס במימד גבוה?

מכיוון שכל המרחקים הופכים דומים (משפט 2.2), **אין משמעות ל"קרוב ביותר"**. הנקודה ה-1 הקרובה ביותר והנקודה ה-1000 הקרובה ביותר נמצאות כמעט באותו מרחק!

ניסוי מספרי – סימולציה:

תוצאה צפויה: הדיוק יורד מ-95% (במימד 2) ל-55% (במימד 1000) – כמעט אקראי.

2.6 יחס פיצ'ר-דגימה: כמה נתונים צריך?

כפי שהדגיש ד"ר יורם סגל בהרצאתו, קיימים כללי אצבע לקביעת מספר הדגימות הנדרש:

כלל 1 – למידת מכונה קלאסית:

$$n_{\text{samples}} \geq 30 \times n_{\text{features}} \tag{10}$$

הדגמת קריסת KNN במימד גבוה

```python
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

def knn_curse_demo(dims, n_samples=1000):
    """
    Examines KNN performance as a function of number of dimensions.
    """
    results = []

    for d in dims:
        # Create synthetic data
        X, y = make_classification(
            n_samples=n_samples,
            n_features=d,
            n_informative=min(d, 10),  # Only 10 informative features
            n_redundant=0,
            random_state=42
        )

        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.3, random_state=42
        )

        # Train KNN
        knn = KNeighborsClassifier(n_neighbors=5)
        knn.fit(X_train, y_train)

        # Accuracy
        accuracy = knn.score(X_test, y_test)
        results.append(accuracy)

        print(f"Dim={d}: Accuracy={accuracy:.3f}")

    return results

# Run
dimensions = [2, 5, 10, 50, 100, 500, 1000]
accuracies = knn_curse_demo(dimensions)
```

כלל 2 – למידה עמוקה:

$$n_{\text{samples}} \geq 100 \times n_{\text{features}} \tag{11}$$

הצדקה מתמטית:

כללים אלו נובעים מתורת הלמידה הסטטיסטית. לפי **גבול VC (Vapnik-Chervonenkis)** [13], שגיאת ההכללה של מודל עם VC-dimension d מוגבלת ע"י:

$$\epsilon \leq \sqrt{\frac{d \log(n/d) + \log(1/\delta)}{n}} \tag{12}$$

כאשר n הוא מספר הדגימות, δ רמת הביטחון, ו-ϵ שגיאת ההכללה.

פירוש: כדי לקבל שגיאה קטנה ϵ, נדרש $n \sim O(d/\epsilon^2)$ – **מספר הדגימות צריך לגדול ליניארית עם המימד.**

דוגמה מהחיים – תמונת חתול:

תמונה 1000×1000 פיקסלים BGR:

- מספר תכונות: $1000 \times 1000 \times 3 = 3000000$

- דגימות נדרשות (כלל 2): $100 \times 3000000 = 300000000$ (שלוש מאות מיליון תמונות!)

למה זה בלתי אפשרי?

- ImageNet, אחד ממאגרי התמונות הגדולים בעולם, מכיל רק $1.2 \sim$ מיליון תמונות אימון [14]

- גם Google לא צילמה 300 מיליון תמונות חתולים

אז איך Deep Learning מצליח?

התשובה: **שיתוף משקלים והנחות אינדוקטיביות** [7], [9].

2.7 הפתרון: רשתות CNN ומבנה מרחבי

רשתות Convolutional Neural Networks (CNN), שפותחו על ידי Yann LeCun ב-1989 [15], מתגברות על קללת המימדיות באמצעות שלושה עקרונות:

עיקרון 1 – קישוריות מקומית (Local Connectivity):

במקום לחבר כל פיקסל לכל נוירון (שיצטרך $1000^2 \times 1000^2 = 10^{12}$ משקלים), כל נוירון מחובר רק **לאזור קטן** (למשל, 3×3 פיקסלים).

עיקרון 2 – שיתוף משקלים (Weight Sharing):

אותו פילטר (קרנל) משמש **בכל מקום בתמונה**. במקום 10^{12} משקלים, נדרשים רק $3 \times 3 \times 64 = 576$ משקלים לשכבה!

עיקרון 3 – היררכיה של תכונות:

השכבות הראשונות לומדות תכונות פשוטות (קצוות, צבעים), והשכבות העמוקות לומדות תכונות מורכבות (עיניים, אוזניים, פנים).

השוואת מספר הפרמטרים:

טבלה 5: השוואת מספר פרמטרים: Fully Connected מול CNN

ארכיטקטורה	מספר פרמטרים	דגימות נדרשות (כלל 2)
Fully Connected	$\sim 10^{12}$	10^{14} (בלתי אפשרי)
CNN (AlexNet)	$\sim 60 \times 10^6$	6×10^9
CNN (ResNet-50)	$\sim 25 \times 10^6$	2.5×10^9

הצלחה מעשית:

רשת AlexNet [8] אומנה על 1.2 מיליון תמונות בלבד (הרבה פחות מהנדרש תיאורטית), והשיגה פריצת דרך בתחרות ImageNet ב-2012, עם הפחתה של 10% בשגיאה לעומת השיטות הקודמות.

2.8 פתרונות נוספים: הפחתת ממדיות

מלבד CNN, קיימות שיטות נוספות להתמודדות עם קללת המימדיות:

שיטה 1 – Feature Selection (בחירת תכונות):

בחירת תת-קבוצה של התכונות החשובות ביותר. שיטות נפוצות:

- **Filter Methods**: מיון לפי קורלציה, mutual information, chi-square
- **Wrapper Methods**: בחירה לפי ביצועי המודל (forward selection, backward elimination)
- **Embedded Methods**: רגולריזציה L1 (Lasso) שמאלצת משקלים לאפס

שיטה 2 – PCA (Principal Component Analysis):

המצאה של Karl Pearson (1901) [16] ו-Harold Hotelling (1933) [17]. PCA מוצא את הכיוונים בעלי השונות המקסימלית ומקרין עליהם.

רעיון PCA:

אם $\mathbf{X} \in \mathbb{R}^{n\times d}$ היא מטריצת הנתונים, PCA מוצא את הווקטורים העצמיים של מטריצת הקווריאנס $\mathbf{C} = \frac{1}{n}\mathbf{X}^T\mathbf{X}$.

הווקטורים העצמיים $\{\vec{v}_1, \ldots, \vec{v}_d\}$ עם הערכים העצמיים הגדולים ביותר $\{\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d\}$ מהווים כיוונים של שונות מקסימלית.

הקרנה על k מרכיבים עיקריים:

$$\mathbf{Z} = \mathbf{X}\mathbf{V}_k \tag{13}$$

כאשר $\mathbf{V}_k = [\vec{v}_1, \ldots, \vec{v}_k]$ ו-$\mathbf{Z} \in \mathbb{R}^{n\times k}$ הוא הייצוג המופחת ($k \ll d$).

כמה מרכיבים לבחור?

כלל אצבע: בחר k כך ש**השונות המוסברת** היא לפחות 95%:

$$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i} \geq 0.95 \tag{14}$$

פסאודו-קוד – PCA ב-NumPy:

הערה על SVD: Singular Value Decomposition הוא שיטה יעילה יותר לחישוב PCA מאשר חישוב ישיר של ערכים עצמיים, במיוחד כאשר $n \ll d$ [18].

2.9 תרגיל תכנות עצמי 2.1 – הפחתת ממדיות עם PCA

מטרה: להמחיש את יכולת PCA לשמר מידע תוך הפחתת ממדים.

משימה:

1. טענו את מערך הנתונים MNIST (תמונות ספרות כתובות ביד, 28×28 פיקסלים = 784 תכונות)
2. הפחיתו ל-50 מרכיבים עיקריים באמצעות PCA
3. שחזרו תמונות מהייצוג המופחת
4. השוו את התמונות המקוריות למשוחזרות
5. חשבו את אחוז השונות המוסברת

פסאודו-קוד מורחב:

תוצאה צפויה: עם 50 מרכיבים (במקום 784), ניתן לשמר ~85% מהמידע, והתמונות המשוחזרות יהיו קריאות למדי.

2.10 רגולריזציה: מניעת Overfitting במימד גבוה

כאשר מספר התכונות גדול ביחס למספר הדגימות, המודל נוטה ל**התאמת יתר** (Overfitting) – הוא "לומד בעל-פה" את נתוני האימון ונכשל בנתונים חדשים.

רגולריזציה (Regularization) היא טכניקה שמוסיפה "קנס" על מורכבות המודל, ובכך מעודדת פשטות.

שתי שיטות עיקריות:

רגולריזציה L2 (Ridge):

$$\mathcal{L}_{\text{Ridge}} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{d} w_j^2 \tag{15}$$

קנס על **גודל המשקלים** – מעודד משקלים קטנים, אך לא מאפס אותם.

PCA – הפחתת ממדיות

```python
import numpy as np

def pca_manual(X, n_components=2):
    """
␣␣␣␣Performs␣manual␣PCA␣using␣SVD.

␣␣␣␣Args:
␣␣␣␣␣␣␣␣X:␣data␣matrix␣(n_samples,␣n_features)
␣␣␣␣␣␣␣␣n_components:␣number␣of␣principal␣components

␣␣␣␣Returns:
␣␣␣␣␣␣␣␣X_reduced:␣reduced␣data␣(n_samples,␣n_components)
␣␣␣␣␣␣␣␣components:␣eigenvectors
␣␣␣␣␣␣␣␣explained_var_ratio:␣explained␣variance␣ratio
␣␣␣␣"""
    # Center the data (subtract mean)
    X_centered = X - np.mean(X, axis=0)

    # Singular Value Decomposition
    # U: eigenvectors of XX^T
    # S: singular values (square root of eigenvalues)
    # Vt: eigenvectors of X^TX (principal components)
    U, S, Vt = np.linalg.svd(X_centered, full_matrices=False)

    # Select k principal components
    components = Vt[:n_components]

    # Projection
    X_reduced = X_centered @ components.T

    # Compute explained variance
    explained_variance = (S**2) / (len(X) - 1)
    explained_var_ratio = explained_variance[:n_components] / np.sum(
explained_variance)

    print(f"Explained␣variance␣ratio:␣{explained_var_ratio}")
    print(f"Total:␣{np.sum(explained_var_ratio):.3f}")

    return X_reduced, components, explained_var_ratio

# Example
X = np.random.rand(1000, 100)  # 1000 samples, 100 features
X_reduced, components, var_ratio = pca_manual(X, n_components=10)
print(f"Original␣shape:␣{X.shape},␣Reduced␣shape:␣{X_reduced.shape}")
```

PCA על MNIST – דוגמה מלאה

```python
from sklearn.datasets import fetch_openml
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np

# Load MNIST
mnist = fetch_openml('mnist_784', version=1, parser='auto')
X = mnist.data.to_numpy()[:1000]  # First 1000 samples
y = mnist.target.to_numpy()[:1000]

# Normalization (important!)
X = X / 255.0

print(f"Original␣shape:␣{X.shape}")  # (1000, 784)

# PCA reduction to 50 components
pca = PCA(n_components=50)
X_reduced = pca.fit_transform(X)

print(f"Reduced␣shape:␣{X_reduced.shape}")  # (1000, 50)
print(f"Explained␣variance:␣{np.sum(pca.explained_variance_ratio_):.3f}"
)

# Reconstruct images
X_reconstructed = pca.inverse_transform(X_reduced)

# Display
fig, axes = plt.subplots(2, 5, figsize=(12, 5))
for i in range(5):
    # Original image
    axes[0, i].imshow(X[i].reshape(28, 28), cmap='gray')
    axes[0, i].set_title(f'Original:␣{y[i]}')
    axes[0, i].axis('off')

    # Reconstructed image
    axes[1, i].imshow(X_reconstructed[i].reshape(28, 28), cmap='gray')
    axes[1, i].set_title(f'Reconstructed')
    axes[1, i].axis('off')

plt.tight_layout()
plt.show()
```

רגולריזציה L1 (Lasso):

$$\mathcal{L}_{\text{Lasso}} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{d} |w_j| \tag{16}$$

קנס על **סכום ערכי המשקלים** – מאלץ חלק מהמשקלים להיות **בדיוק אפס**, ובכך מבצע בחירת תכונות אוטומטית.

פרמטר הקנס λ:

- $\lambda = 0$: אין רגולריזציה (סיכון ל-gnittifrevO)
- λ גדול מאוד: המודל פשוט מדי (Underfitting)
- λ אופטימלי: נקבע באמצעות Cross-Validation

מי המציא?

Ridge: Arthur Hoerl ו-dranneK treboR (1970) [19]

Lasso: Robert Tibshirani (1996) [20]

פסאודו-קוד – השוואת Ridge ו-ossaL:

תוצאה צפויה: Lasso יאפס $490 \sim$ מתוך 500 המשקלים, וישמור רק את התכונות האינפורמטיביות.

2.11 מחקר עדכני: אקסטרפולציה מול אינטרפולציה

מחקר פורץ דרך של Anadi Chaman ו-cinamkoD navI (2021) [21] גילה תוצאה מדהימה: **במרחבים עם יותר מ-100 ממדים, דגימה חדשה כמעט אף פעם אינה אינטרפולציה – היא כמעט תמיד אקסטרפולציה.**

הגדרות:

- **אינטרפולציה:** דגימה חדשה נמצאת בתוך המעטפת הקמורה (Convex Hull) של נתוני האימון
- **אקסטרפולציה:** דגימה חדשה נמצאת מחוץ למעטפת הקמורה

המעטפת הקמורה (Convex Hull):

$$\text{Conv}(\mathcal{X}) = \left\{ \sum_{i=1}^{n} \alpha_i \vec{x}_i \mid \vec{x}_i \in \mathcal{X}, \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i = 1 \right\} \tag{17}$$

התוצאה המפתיעה:

בניסויים על מערכי נתונים אמיתיים (MNIST, CIFAR-10), Chaman ו-cinamkoD מצאו ש-99.9% מדגימות הבדיקה הן אקסטרפולציה במרחב המקורי!

משמעות ל-IA:

רגולריזציה – Ridge מול Lasso

```python
from sklearn.linear_model import Ridge, Lasso
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
import numpy as np

# Create synthetic data: n << d (curse of dimensionality!)
X, y = make_regression(
    n_samples=100,      # Only 100 samples
    n_features=500,     # 500 features!
    n_informative=10,   # Only 10 relevant
    noise=10,
    random_state=42
)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# Ridge
ridge = Ridge(alpha=1.0)  # alpha = lambda
ridge.fit(X_train, y_train)
ridge_score = ridge.score(X_test, y_test)

# Lasso
lasso = Lasso(alpha=1.0)
lasso.fit(X_train, y_train)
lasso_score = lasso.score(X_test, y_test)

print(f"Ridge␣R²:␣{ridge_score:.3f}")
print(f"Lasso␣R²:␣{lasso_score:.3f}")

# How many Lasso weights are zero?
zero_weights = np.sum(np.abs(lasso.coef_) < 1e-5)
print(f"Lasso␣zeroed␣out␣{zero_weights}/{len(lasso.coef_)}␣features")
```

כל פעם שמודל Deep Learning מסווג תמונה חדשה, הוא למעשה **משער מעבר לנתונים שראה**. ההצלחה של מודלים מודרניים נובעת מה**הנחות האינדוקטיביות** שלהם (כמו מבנה ה-NNC) ולא מכיסוי מלא של המרחב.

שאלת מחשבה שהעלה המרצה:

"אם כל תחזית היא אקסטרפולציה, האם AI באמת 'מבין' את העולם, או שהוא רק מנחש בצורה חכמה?"

זוהי אחת השאלות הפילוסופיות המרכזיות של AI מודרני.

2.12 סיכום ומבט קדימה

מה למדנו בפרק זה?

1. **קללת המימדיות היא אמיתית ומוכחת** – נפח המרחב גדל אקספוננציאלית, והנתונים הופכים דלילים
2. **מרחקים מתכנסים** – במימד גבוה, כל הנקודות נעשות "באותו מרחק", ומדדים כמו KNN קורסים
3. **יחס פיצ'ר-דגימה הוא קריטי** – נדרשות לפחות 30-100 דגימות לכל תכונה
4. **רשתות CNN מתגברות על הקללה** – שיתוף משקלים וקישוריות מקומית מפחיתים דרמטית את מספר הפרמטרים
5. **הפחתת ממדיות עובדת** – PCA, Feature Selection, Regularization מאפשרים עבודה יעילה
6. **רוב התחזיות הן אקסטרפולציה** – AI מצליח בזכות הנחות אינדוקטיביות, לא כיסוי מלא

מבט קדימה – פרק 3:

בפרק הבא נעבור מהבעיות לפתרונות. נחקור את **מקדם הקביעה** R^2 – הכלי המרכזי להערכת מודלי רגרסיה. נראה:

- הגדרה מתמטית מלאה של R^2
- הוכחה: למה R^2 תמיד בין 0 ל-1?
- סכנות: מתי R^2 מטעה?
- Adjusted R^2 – הגרסה המתוקנת
- קשר לקורלציה ולמכפלה סקלרית

שאלת מחשבה לסיום:

אם מודל רגרסיה ליניארית השיג $R^2 = 0.95$ על נתוני האימון, אך רק $R^2 = 0.60$ על נתוני הבדיקה – מה קרה? האם זו קללת המימדיות, Overfitting, או משהו אחר?

מטלות וקריאה מורחבת

תרגיל 2.1: הפחיתו את MNIST עם PCA (ראו פסאודו-קוד).

תרגיל 2.2: השוו Ridge ו-ossaL על נתונים עם $n < d$ (ראו פסאודו-קוד).

תרגיל 2.3: הוכיחו בעצמכם את משפט 2.1 (ריכוז הנפח בקליפה) עבור $d = 10$.
רמז: חשבו $V_d(r) = \frac{\pi^{d/2}}{\Gamma(d/2+1)} r^d$ עבור $r = 0.9$ ו-$r = 1$.

קריאה מורחבת:

- [6] – Dynamic Programming: המקור המקורי של "קללת המימדיות"
- [10] – "When Is 'Nearest Neighbor' Meaningful?": ניתוח מתמטי של קריסת KNN
- [21] – "Truly Generative or Just Extrapolation?": המחקר על אינטרפולציה מול אקסטרפולציה
- [9] – Deep Learning, פרק 7: רגולריזציה והפחתת Overfitting
- [22] – The Elements of Statistical Learning, פרק 3: רגרסיה ליניארית ורגולריזציה

שאלות להעמקה:

1. מדוע Lasso מאפס משקלים אך Ridge לא? רמז: חשבו על הגיאומטריה של הקנס (L_1 מול L_2)
2. האם קללת המימדיות משפיעה גם על רשתות Transformer ב-PLN?
3. אם 99.9% מהתחזיות הן אקסטרפולציה, איך מסבירים את ההצלחה של GPT-4?

סיום פרק 2

3 הערכת מודלים: מקדם הקביעה R^2

Model Evaluation: The Coefficient of Determination R^2

בפרק זה נחקור את אחד הכלים המרכזיים להערכת מודלי רגרסיה: **מקדם הקביעה** R^2 (R-squared). נבין את המשמעות המתמטית שלו, נוכיח מדוע הוא תמיד בין 0 ל-1, ונראה מתי הוא עלול להטעות אותנו.

3.1 השאלה המרכזית: מה זה מודל "טוב"?

דמיינו שבניתם מודל רגרסיה ליניארית לחיזוי מחיר דירה על סמך שטח, מיקום וגיל. המודל נותן תחזיות, אך **כיצד נדע אם הוא טוב?**

שאלה זו מעסיקה סטטיסטיקאים ומדעני נתונים מאז ראשית המדע. התשובה המודרנית, שהתגבשה במאה ה-20, היא **מקדם הקביעה** – מספר יחיד שמסכם את "טיב ההתאמה" של המודל לנתונים.

3.2 היסטוריה: מי המציא את R^2?

המושג פותח בהדרגה על ידי מספר סטטיסטיקאים:

Francis Galton (1822–1911) היה הראשון לחקור קורלציה ורגרסיה. בעבודתו "Regression towards Mediocrity in Hereditary Stature" (1886) [23], הוא גילה שילדים של הורים גבוהים נוטים להיות נמוכים יותר מהוריהם (רגרסיה לממוצע).

Karl Pearson (1857–1936), תלמידו של Galton, פיתח את **מקדם הקורלציה** r (1896) [24], שהוא השורש של R^2 במקרה של רגרסיה פשוטה.

Ronald Fisher (1890–1962) הרחיב את המושג ל**רגרסיה מרובה** (Multiple Regression) וניתח את חלוקת השונות [25].

Sewall Wright (1889–1988) טבע את הסימון R^2 ב-1921 [26].

3.3 הגדרה: מהו R^2?

הגדרה 3.1 – מקדם הקביעה:

R^2 הוא מדד סטטיסטי המצביע על **שיעור השונות במשתנה התלוי** Y **שמוסברת על ידי המשתנים הבלתי תלויים** X במודל רגרסיה.

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}} \tag{18}$$

כאשר:

- SS_{res} – **סכום ריבועי השאריות** (Residual Sum of Squares)
- SS_{tot} – **סכום הריבועים הכולל** (Total Sum of Squares)

פירוט המשתנים:

$$SS_{\text{res}} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{19}$$

$$SS_{\text{tot}} = \sum_{i=1}^{n}(y_i - \bar{y})^2 \tag{20}$$

כאשר:

- y_i – הערך האמיתי של התצפית ה-i
- $\hat{y}_i$ – הערך החזוי על ידי המודל עבור התצפית ה-i
- $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ – ממוצע הערכים האמיתיים

3.4 משמעות אינטואיטיבית: מהו "הסבר שונות"?

כדי להבין את R^2, נתחיל מ**מודל הבסיס** – המודל הפשוט ביותר האפשרי.

מודל הבסיס (ledoM enilesaB): תחזה תמיד את הממוצע $\bar{y}$.

זהו המודל הטיפש ביותר – הוא מתעלם מכל התכונות X ופשוט אומר "התשובה היא הממוצע". שגיאת מודל זה היא SS_{tot} – **השונות הכוללת** בנתונים.

המודל שלנו: משתמש ב-X כדי לחזות $\hat{y}_i$.

שגיאת המודל שלנו היא SS_{res} – **השונות שנותרה** אחרי שהמודל עשה את עבודתו.

הפרשנות:

$$R^2 = 1 - \frac{\text{שגיאת המודל שלנו}}{\text{שגיאת מודל הבסיס}} = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}} \tag{21}$$

- אם $R^2 = 0$: המודל שלנו לא טוב יותר מהממוצע (חסר תועלת!)
- אם $R^2 = 1$: המודל מושלם – אין שגיאות כלל
- אם $R^2 = 0.8$: המודל מסביר 80% מהשונות

דוגמה מספרית:

נניח שאנחנו מנבאים מחירי דירות. הממוצע הוא 500000 ש"ח.

טבלה 6: דוגמה: חישוב R^2 למחירי דירות

תצפית	y_i (אמיתי)	$\hat{y}_i$ (חזוי)	$(y_i - \bar{y})^2$	$(y_i - \hat{y}_i)^2$
1	600000	580000	10000000000	400000000
2	450000	470000	2500000000	400000000
3	700000	680000	40000000000	400000000
סה"כ			$SS_{\text{tot}} = 52500000000$	$SS_{\text{res}} = 1200000000$

חישוב:

$$R^2 = 1 - \frac{1200000000}{52500000000} = 1 - 0.023 = 0.977$$

פרשנות: המודל מסביר 97.7% מהשונות במחירים – מודל מצוין!

3.5 הוכחה: R^2 תמיד בין 0 ל-1

משפט 3.1 – תחום R^2:
עבור רגרסיה ליניארית עם איבר חופשי, מתקיים: $0 \leq R^2 \leq 1$.
הוכחה:
חלק א': $R^2 \leq 1$
נוכיח ש-$SS_{\text{res}} \leq SS_{\text{tot}}$.
צעד 1: מודל הרגרסיה הליניארית ממזער את SS_{res} על ידי פתרון:

$$\min_{\vec{w}} \sum_{i=1}^{n} (y_i - \vec{w}^T \vec{x}_i)^2$$

צעד 2: מודל הבסיס (חיזוי הממוצע) הוא מקרה פרטי של רגרסיה ליניארית עם $\vec{w} = \vec{0}$ (למעט איבר חופשי = $\bar{y}$).
צעד 3: מכיוון שהרגרסיה ממזערת, היא לא יכולה להיות גרועה יותר ממודל הבסיס:

$$SS_{\text{res}} \leq SS_{\text{tot}}$$

צעד 4: לכן:

$$\frac{SS_{\text{res}}}{SS_{\text{tot}}} \leq 1 \quad \Rightarrow \quad 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}} \leq 1 \quad \Rightarrow \quad R^2 \leq 1 \quad \blacksquare$$

חלק ב': $R^2 \geq 0$

זה נכון **רק לרגרסיה ליניארית עם איבר חופשי**. במקרים מיוחדים (כמו רגרסיה דרך הראשית), R^2 יכול להיות שלילי!

עבור רגרסיה רגילה, המודל תמיד יכול לחזות את הממוצע (במקרה הגרוע ביותר), כך ש-$SS_{\text{res}} \leq SS_{\text{tot}}$ ולכן $R^2 \geq 0$.

$\blacksquare$

3.6 ייצוג גרפי: חלוקת השונות

ניתן לחשוב על השונות הכוללת כ"פאי" שמתחלק לשניים:

$$\underbrace{SS_{\text{tot}}}_{\text{שונות כוללת}} = \underbrace{SS_{\text{reg}}}_{\text{שונות מוסברת}} + \underbrace{SS_{\text{res}}}_{\text{שונות לא מוסברת}} \tag{22}$$

כאשר $SS_{\text{reg}} = \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2$ – השונות שהמודל "לכד".

קשר ל-R^2:

$$R^2 = 1 - \frac{\text{שגיאת המודל שלנו}}{\text{שגיאת מודל הבסיס}} = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}} \tag{23}$$

זו הגרסה החיובית של ההגדרה – במקום "אחד פחות שאריות", זה "שיעור המוסבר".

3.7 קשר למקדם הקורלציה

במקרה של **רגרסיה ליניארית פשוטה** (משתנה בלתי תלוי אחד), מתקיים:

$$R^2 = r^2 \tag{24}$$

כאשר r הוא **מקדם הקורלציה של פירסון** בין X ו-Y:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{25}$$

הוכחה (סקיצה):

ברגרסיה פשוטה, $\hat{y}_i = a + bx_i$ כאשר:

$$b = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2} = r \cdot \frac{\sigma_y}{\sigma_x}$$

ניתן להוכיח (באלגברה מייגעת אך ישירה) ש-$SS_{\text{reg}} = r^2 \cdot SS_{\text{tot}}$, ולכן $R^2 = r^2$.

■

משמעות: ברגרסיה פשוטה, R^2 הוא ריבוע הקורלציה. אם $r = 0.9$ (קורלציה גבוהה), אזי $R^2 = 0.81$.

3.8 סכנה 1: R^2 עולה תמיד כשמוסיפים תכונות

הבעיה הקריטית של R^2: **הוא אף פעם לא יורד כשמוסיפים תכונות**, גם אם הן חסרות תועלת!

משפט 3.2 – מונוטוניות R^2:

אם מוסיפים תכונה למודל רגרסיה, R^2 לא יורד (ובדרך כלל עולה).

הוכחה:

צעד 1: נניח מודל עם p תכונות מושג R_p^2. נוסיף תכונה נוספת (סה"כ $p+1$).

צעד 2: המודל החדש יכול תמיד לבחור משקל $w_{p+1} = 0$ לתכונה החדשה, ובכך להשיג את אותה שגיאה כמו המודל הישן.

צעד 3: מכיוון שהרגרסיה ממזערת את SS_{res}, היא תמצא משקלים טובים או שווים:

$$SS_{\text{res}}^{(p+1)} \leq SS_{\text{res}}^{(p)}$$

צעד 4: לכן:

$$R_{p+1}^2 = 1 - \frac{SS_{\text{res}}^{(p+1)}}{SS_{\text{tot}}} \geq 1 - \frac{SS_{\text{res}}^{(p)}}{SS_{\text{tot}}} = R_p^2 \quad \blacksquare$$

המשמעות המסוכנת:

אם נוסיף 1000 תכונות אקראיות (רעש טהור), R^2 יעלה! זה יוביל אותנו לחשוב שהמודל השתפר, אך באמת הוא פשוט **התאים יתר** (Overfitted).

דוגמה מספרית – סימולציה:

הדגמת בעיית R^2 עם תכונות אקראיות

```python
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

# Simple data: Y = 2X + noise
np.random.seed(42)
n = 100
X_real = np.random.rand(n, 1)
y = 2 * X_real.flatten() + np.random.randn(n) * 0.5

# Test R² as a function of number of random features
r2_scores = []

for num_random_features in [0, 1, 5, 10, 20, 50]:
    # Add random features (noise!)
    X_random = np.random.randn(n, num_random_features)
    X_combined = np.hstack([X_real, X_random])

    # Train
    model = LinearRegression()
    model.fit(X_combined, y)

    # R² on training data
    y_pred = model.predict(X_combined)
    r2 = r2_score(y, y_pred)
    r2_scores.append(r2)

    print(f"Num features: {1 + num_random_features}, R²: {r2:.4f}")

# Result: R² increases with each feature, even if it's noise!
```

תוצאה צפויה:

טבלה 7: עליית R^2 עם הוספת תכונות אקראיות

R^2 **(אימון)**	**מספר תכונות**
0.850	1
0.852	2
0.870	6
0.895	11
0.930	21
0.975	51

R^2 עלה מ-0.85 ל-0.975 למרות שהתכונות הנוספות היו רעש טהור!

3.9 הפתרון: Adjusted R^2

Adjusted R^2 (מקדם הקביעה המתוקן) פותר את הבעיה על ידי הטלת ***"קנס"*** על הוספת תכונות.

הגדרה 3.2 – Adjusted R^2**:**

$$R^2_{\text{adj}} = 1 - \frac{SS_{\text{res}}/(n-p-1)}{SS_{\text{tot}}/(n-1)} \tag{26}$$

כאשר:

- n – מספר התצפיות
- p – מספר התכונות (לא כולל איבר חופשי)

ניסוח חלופי:

$$R^2_{\text{adj}} = 1 - (1 - R^2) \cdot \frac{n-1}{n-p-1} \tag{27}$$

משמעות:

המונה והמכנה מחולקים במספר **דרגות החופש** (Degrees of Freedom). ככל שיש יותר תכונות p, דרגות החופש קטנות יותר, והקנס גדל.

תכונות של Adjusted R^2**:**

- **יכול לרדת** כשמוסיפים תכונה חסרת תועלת

- **יכול להיות שלילי** (אם המודל גרוע מאוד)

- תמיד $R^2_{\text{adj}} \leq R^2$

מתי להשתמש?

- R^2: להערכת התאמה על נתוני אימון בודדים

- **Adjusted** R^2: להשוואה בין מודלים עם מספר תכונות שונה

חזרה על הסימולציה עם Adjusted R^2:

Adjusted R^2 – מונע עליה מלאכותית

```
def adjusted_r2(r2, n, p):
    """
␣␣␣␣Compute␣Adjusted␣R².

␣␣␣␣Args:
␣␣␣␣␣␣␣␣r2:␣regular␣R²
␣␣␣␣␣␣␣␣n:␣number␣of␣samples
␣␣␣␣␣␣␣␣p:␣number␣of␣features␣(excluding␣intercept)
␣␣␣␣"""
    return 1 - (1 - r2) * (n - 1) / (n - p - 1)

# Recalculation
for i, num_random in enumerate([0, 1, 5, 10, 20, 50]):
    r2 = r2_scores[i]
    n = 100
    p = 1 + num_random
    adj_r2 = adjusted_r2(r2, n, p)

    print(f"Features:␣{p},␣R²:␣{r2:.4f},␣Adj␣R²:␣{adj_r2:.4f}")
```

תוצאה צפויה:

טבלה 8: השוואה: R^2 מול Adjusted R^2

תכונות	R^2	Adjusted R^2
1	0.850	0.849
2	0.852	0.849
6	0.870	0.862
11	0.895	0.880
21	0.930	0.900
51	0.975	0.895

Adjusted R^2 יורד כשמוסיפים תכונות חסרות תועלת – בדיוק כמו שצריך!

3.10 סכנה 2: R^2 לא מתאים לסיווג

R^2 מיועד **לרגרסיה** (תחזית של ערך מספרי). עבור **סיווג** (תחזית של קטגוריה), הוא לא מתאים.

למה?

בסיווג, $y_i \in \{0,1\}$ אך $\hat{y}_i$ יכול להיות כל מספר (למשל, 0.73 = הסתברות לקטגוריה 1). חישוב SS_{res} לא משקף את דיוק הסיווג.

מדדים נכונים לסיווג:

טבלה 9: מדדי הערכה למשימות שונות

משימה	מדד	הסבר
רגרסיה	R^2, RMSE, MAE	R^2 מודד שיעור שונות מוסברת
סיווג	Accuracy, Precision, Recall, F1	מודדים אחוז סיווגים נכונים
סיווג (התפלגות)	Log-Loss, AUC-ROC	מודדים איכות הסתברויות

3.11 קשר לאלגברה ליניארית: הקרנות

ניתן לראות רגרסיה ליניארית כ**הטלה** של $\vec{y}$ על המרחב שנפרש על ידי עמודות המטריצה $\mathbf{X}$.

נוסחת הרגרסיה במטריצות:

$$\hat{\vec{y}} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\vec{y} = \mathbf{P}\vec{y} \tag{28}$$

כאשר $\mathbf{P} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ היא **מטריצת ההטלה** (Projection Matrix).

משמעות גיאומטרית של R^2**:**

$$R^2 = \frac{\|\hat{\vec{y}} - \bar{y}\vec{1}\|^2}{\|\vec{y} - \bar{y}\vec{1}\|^2} = \cos^2(\theta)$$

כאשר θ היא הזווית בין $\vec{y} - \bar{y}\vec{1}$ (הנתונים הממורכזים) ל-$\hat{\vec{y}} - \bar{y}\vec{1}$ (החיזוי הממורכז).

פרשנות: R^2 הוא ריבוע הקוסינוס של הזווית בין הנתונים לחיזוי במרחב הממורכז – **מדד גיאומטרי של התאמה**.

3.12 תרגיל תכנות עצמי 3.1 – חישוב R^2 ידני

מטרה: להבין את המשמעות של R^2 על ידי חישוב ידני.

משימה:

1. צרו נתונים סינתטיים: $y = 3x + 5 +$ שער

2. אמנו מודל רגרסיה ליניארית

3. חשבו את R^2 בשלוש דרכים:

   - נוסחה ישירה: $1 - SS_{\text{res}}/SS_{\text{tot}}$
   - באמצעות קורלציה: r^2
   - באמצעות sklearn

4. השוו את התוצאות (צריכות להיות זהות!)

פסאודו-קוד מלא:

חישוב R^2 בשלוש דרכים

```python
from sklearn.metrics import r2_score

# Method 1: Direct formula
SS_tot = np.sum((y - np.mean(y))**2)
SS_res = np.sum((y - y_pred)**2)
r2_manual = 1 - (SS_res / SS_tot)

# Method 2: Correlation squared (simple regression)
r2_from_corr = np.corrcoef(X.flatten(), y)[0, 1]**2

# Method 3: sklearn
r2_sklearn = r2_score(y, y_pred)
```

תוצאה צפויה: כל שלוש השיטות יתנו $R^2 \approx 0.95$ (תלוי ברעש).

3.13 תרגיל תכנות עצמי 3.2 – השוואת R^2 ו-detsujdA R^2

מטרה: להמחיש את ההבדל בין R^2 ל-detsujdA R^2 כאשר מוסיפים תכונות.

משימה:

1. צרו נתונים עם תכונה אחת רלוונטית
2. הוסיפו בהדרגה תכונות אקראיות (רעש)
3. עקבו אחרי R^2 ו-detsujdA R^2 כפונקציה של מספר התכונות
4. הציגו גרף: ציר X = מספר תכונות, ציר Y = ציון

R^2 מול Adjusted R^2 – השוואה

```python
def adjusted_r2(r2, n, p):
    return 1 - (1 - r2) * (n - 1) / (n - p - 1)

# As features increase: R² rises, Adjusted R² falls
for p in range(1, 51):
    adj_r2 = adjusted_r2(r2, 100, p)
```

תוצאה צפויה: R^2 עולה בהתמדה, אך Adjusted R^2 יורד אחרי כמה תכונות.

3.14 מקרי קצה: מתי R^2 מטעה?

מקרה 1: יחסים לא-ליניאריים

אם הקשר בין X ל-Y הוא לא-ליניארי (למשל, ריבועי), רגרסיה ליניארית תיכשל ו-R^2 יהיה נמוך – אך זה לא אומר שאין קשר!

דוגמה: $y = x^2$

רגרסיה ליניארית תיתן R^2 נמוך, אך הקשר מושלם (רק לא ליניארי).

פתרון: השתמש ב**פולינומים** או מודלים לא-ליניאריים (כמו רשתות נוירונים).

מקרה 2: Outliers (ערכי חריגים)

ערך חריג אחד יכול להוריד דרמטית את R^2, גם אם המודל טוב ל-99% מהנתונים.

פתרון: זהה וטפל ב-sreiltuO לפני אימון (באמצעות Z-score, IQR, או שיטות עמידות כמו RANSAC).

מקרה 3: מתאם אינו סיבתיות

R^2 גבוה לא אומר ש-X גורם ל-Y. ייתכן שיש משתנה שלישי Z שגורם לשניהם.

דוגמה קלאסית: צריכת גלידה וטביעות מים מתואמות – לא כי גלידה גורמת לטביעות, אלא כי **קיץ** גורם לשניהם.

3.15 סיכום ומבט קדימה

מה למדנו בפרק זה?

1. R^2 **מודד שיעור שונות מוסברת** – מספר יחיד שמסכם את איכות המודל
2. **תחום: 0 עד 1** – הוכחנו מתמטית למה זה תמיד נכון (לרגרסיה עם intercept)
3. **קשר לקורלציה** – ברגרסיה פשוטה, $R^2 = r^2$
4. **בעיה: עולה תמיד** – הוספת תכונות מעלה את R^2 גם אם הן חסרות תועלת
5. **פתרון: Adjusted R^2** – מנרמל לפי מספר התכונות ומונע Overfitting
6. **לא לסיווג!** – R^2 מיועד לרגרסיה, לא לסיווג
7. **מגבלות** – מתאם אינו סיבתיות, רגיש ל-sreiltuO, מניח ליניאריות

מבט קדימה – פרק 4:

בפרק הבא נעבור מהערכת מודלים למדידת **קשרים בין משתנים**. נחקור:

- **קו-וריאנס** (Covariance) – מדד גולמי של קשר ליניארי
- **קורלציה** (Correlation) – הגרסה המנורמלת של קו-וריאנס
- **ייצוג אלגברי ליניארי** – קו-וריאנס כמכפלה סקלרית, קורלציה כ-ytiralimiS enisoC
- **סכנות** – "Correlation is not Causation", קורלציות מזויפות
- **מטריצת קורלציה** – כלי לזיהוי Multicollinearity

שאלת מחשבה לסיום:

אם שני משתנים X ו-Y בעלי קורלציה $r = 0$ (אורתוגונליים), האם זה אומר שהם בלתי תלויים סטטיסטית?

רמז: התשובה היא **לא!** קורלציה אפס משמעה רק אין קשר ליניארי, אך עדיין יכול להיות קשר לא-ליניארי (למשל, $Y = X^2$).

מטלות וקריאה מורחבת

תרגיל 3.1: חשבו R^2 ידנית בשלוש דרכים (ראו פסאודו-קוד).

תרגיל 3.2: השוו R^2 ו-detsujdA R^2 עם תכונות אקראיות (ראו פסאודו-קוד).

תרגיל 3.3: הוכיחו בעצמכם שברגרסיה פשוטה $R^2 = r^2$.

רמז: התחילו מהביטוי $SS_{\text{reg}} = \sum(\hat{y}_i - \bar{y})^2$ והשתמשו ב-$\hat{y}_i = a + bx_i$.

קריאה מורחבת:

- [23] – "Regression towards Mediocrity in Hereditary Stature": המאמר המקורי של Galton
- [24] – "Mathematical Contributions to the Theory of Evolution": Pearson על קורלציה
- [25] – "Statistical Methods for Research Workers": Fisher על ניתוח שונות
- [26] – "Correlation and Causation": Wright טובע את R^2
- [22] – The Elements of Statistical Learning, פרק 3: רגרסיה ליניארית ברזולוציה גבוהה

שאלות להעמקה:

1. למה Adjusted R^2 יכול להיות שלילי, אך R^2 לא?
2. מה יקרה ל-R^2 אם ננרמל את X ו-Y (נחסיר ממוצע ונחלק בסטיית תקן)?
3. איך R^2 מתייחס למודלים לא-ליניאריים (כמו רשתות נוירונים)?

סיום פרק 3

4 קו-ווריאנס וקורלציה: מדידת קשרים בין משתנים

Covariance and Correlation: Measuring Relationships Between Variables

בפרק זה נחקור את הכלים המתמטיים למדידת קשרים בין משתנים. נראה כיצד קו-ווריאנס וקורלציה מהווים הרחבה טבעית של מכפלה סקלרית ו-ytiralimiS enisoC, ונבין מדוע "**מתאם אינו סיבתיות**" הוא אחד העקרונות החשובים ביותר במדע הנתונים.

4.1 השאלה המרכזית: איך מודדים "יחד-משתנות"?

דמיינו שאנחנו בוחנים את הקשר בין גובה ומשקל של אנשים. אינטואיטיבית, אנחנו מצפים שאנשים גבוהים יותר יהיו בדרך כלל כבדים יותר – הם "משתנים יחד". אך **כיצד נכמת את הקשר הזה?**

התשובה המתמטית היא **קו-ווריאנס** (Covariance) – מדד שמצביע על מידת ה"יחד-משתנות" של שני משתנים.

4.2 היסטוריה: מי המציא את הקו-ווריאנס?

המושג פותח במקביל להתפתחות הסטטיסטיקה המודרנית:

Francis Galton (1822–1911) היה הראשון לחקור קשרים בין משתנים במחקריו על תורשה. בעבודתו על גובה הורים וילדים (1886) [23], הוא גילה שהמשתנים "משתנים יחד" באופן שיטתי.

Karl Pearson (1857–1936) פיתח את **מקדם הקורלציה** r ב-1896 [24], והגדיר באופן פורמלי את הקו-ווריאנס כבסיס למקדם זה.

Ronald Fisher (1890–1962) הרחיב את התורה ל**מטריצות קו-ווריאנס** (Covariance Matrices) ולניתוח רב-משתני [25].

4.3 הגדרה: מהי קו-ווריאנס?

הגדרה 4.1 – קו-ווריאנס:

עבור שני משתנים X ו-Y עם n תצפיות, הקו-ווריאנס מוגדרת כ:

$$\text{Cov}(X,Y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) \tag{29}$$

כאשר $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ ו-$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ הם הממוצעים.

הערה על הגדרות שונות:

חלק מהספרים משתמשים בחלוקה ב-$(n-1)$ במקום n (קו-ווריאנס מדגמית):

$$\text{Cov}_{\text{sample}}(X,Y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) \tag{30}$$

החלוקה ב-$(n-1)$ נותנת **אומד לא מוטה** (Unbiased Estimator) לקו-ווריאנס האוכלוסייה. לצרכי למידת מכונה, ההבדל זניח כאשר n גדול.

4.4 משמעות אינטואיטיבית: מה אומר הסימן?

פרשנות לפי סימן הקו-ווריאנס:

טבלה 10: פרשנות הסימן של קו-ווריאנס

משמעות	ערך
קשר חיובי: כאשר X גדל, Y נוטה לגדול	$\text{Cov}(X,Y) > 0$
אין קשר ליניארי (לא בהכרח בלתי תלויים!)	$\text{Cov}(X,Y) = 0$
קשר שלילי: כאשר X גדל, Y נוטה לקטון	$\text{Cov}(X,Y) < 0$

דוגמה מספרית:

נניח שיש לנו נתוני גובה ומשקל של 5 אנשים:

טבלה 11: דוגמה: גובה ומשקל

משקל (ק"ג)	גובה (ס"מ)	אדם
55	160	1
65	170	2
75	180	3
70	175	4
60	165	5
65	170	**ממוצע**

חישוב:

$$
\begin{aligned}
\mathrm{Cov}(X,Y) = \frac{1}{5}\big[&(160-170)(55-65) + (170-170)(65-65) \\
&+ (180-170)(75-65) + (175-170)(70-65) \\
&+ (165-170)(60-65)\big] \\
= \frac{1}{5}\big[&(-10)(-10) + (0)(0) + (10)(10) + (5)(5) + (-5)(-5)\big] \\
= \frac{1}{5}\big[&100+0+100+25+25\big] = \frac{250}{5} = 50
\end{aligned} \tag{31}
$$

$\mathrm{Cov}(X,Y) = 50 > 0 \leftarrow$ קשר חיובי בין גובה למשקל.

4.5 קשר לאלגברה ליניארית: קו-ווריאנס כמכפלה סקלרית

נזכור מפרק 1 את ההגדרה של מכפלה סקלרית:

$$\langle \vec{u}, \vec{v} \rangle = \sum_{i=1}^{n} u_i v_i \tag{32}$$

קו-ווריאנס היא מכפלה סקלרית של וקטורים ממורכזים!

אם נגדיר:

- $\vec{x}_c = [x_1 - \bar{x}, x_2 - \bar{x}, \ldots, x_n - \bar{x}]^T$ – וקטור X ממורכז
- $\vec{y}_c = [y_1 - \bar{y}, y_2 - \bar{y}, \ldots, y_n - \bar{y}]^T$ – וקטור Y ממורכז

אזי:

$$\mathrm{Cov}(X,Y) = \frac{1}{n}\langle \vec{x}_c, \vec{y}_c \rangle = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) \tag{33}$$

משמעות גיאומטרית:

קו-ווריאנס מודדת את **"ההטיה המשותפת"** של שני משתנים מהממוצע שלהם – זו מכפלה סקלרית של הסטיות!

4.6 הבעיה עם קו-ווריאנס: חוסר סקלה

הבעיה המרכזית של קו-ווריאנס: **הערך שלה תלוי ביחידות המדידה.**

דוגמה בעייתית:

- $\mathrm{Cov}(\text{ס״מ}, \text{ק״ג}) = 50$ (גובה בס״מ, משקל בק״ג)
- $\mathrm{Cov}(\text{מטר}, \text{ק״ג}) = 0.005$ (גובה במטרים, משקל בק״ג)

אותו קשר, אבל ערכים שונים לחלוטין! **לא ניתן להשוות קו-ווריאנסים מזוגות משתנים שונים.**

זו הסיבה שאנחנו זקוקים ל**קורלציה** – גרסה מנורמלת של קו-ווריאנס.

4.7 קורלציה: הגרסה המנורמלת

הגדרה 4.2 – מקדם הקורלציה של פירסון:
מקדם הקורלציה בין X ו-Y מוגדר כ:

$$r = \mathrm{Cor}(X, Y) = \frac{\mathrm{Cov}(X, Y)}{\sigma_X \sigma_Y} \tag{34}$$

כאשר:

- $\sigma_X = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$ – סטיית התקן של X
- $\sigma_Y = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2}$ – סטיית התקן של Y

נוסחה מפורשת:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{35}$$

הכרת הנוסחה?
זו **בדיוק** **Cosine Similarity** של וקטורים ממורכזים! (ראו פרק 1)

4.8 קשר לאלגברה ליניארית: קורלציה כ-ytiralimiS enisoC

נזכור מפרק 1:

$$\cos(\theta) = \frac{\langle \vec{u}, \vec{v} \rangle}{\|\vec{u}\| \cdot \|\vec{v}\|} \tag{36}$$

אם $\vec{x}_c$ ו-$\vec{y}_c$ הם וקטורים ממורכזים, אזי:

$$r = \frac{\langle \vec{x}_c, \vec{y}_c \rangle}{\|\vec{x}_c\| \cdot \|\vec{y}_c\|} = \cos(\theta) \tag{37}$$

פרשנות גיאומטרית מרתקת:
קורלציה היא הקוסינוס של הזווית בין שני משתנים ממורכזים במרחב התצפיות!

- $r = 1$ (זווית $0°$) $\leftarrow$ קשר ליניארי חיובי מושלם
- $r = 0$ (זווית $90°$) $\leftarrow$ אורתוגונליים, אין קשר ליניארי
- $r = -1$ (זווית $180°$) $\leftarrow$ קשר ליניארי שלילי מושלם

4.9 משפט: תחום הקורלציה

משפט 4.1 – תחום מקדם הקורלציה:
לכל שני משתנים X ו-Y, מתקיים: $-1 \le r \le 1$.
הוכחה:

שיטה 1: אי-שוויון קושי-שוורץ
אי-שוויון קושי-שוורץ (Cauchy-Schwarz Inequality) קובע שלכל וקטורים $\vec{u}, \vec{v}$:

$$|\langle \vec{u}, \vec{v} \rangle| \leq \|\vec{u}\| \cdot \|\vec{v}\| \tag{38}$$

כאשר השוויון מתקיים אם ורק אם הווקטורים פרופורציונליים ($\vec{v} = c\vec{u}$).
צעד 1: נישם $\vec{u} = \vec{x}_c$ ו-$\vec{v} = \vec{y}_c$ (ווקטורים ממורכזים).
צעד 2: לפי קושי-שוורץ:

$$|\langle \vec{x}_c, \vec{y}_c \rangle| \leq \|\vec{x}_c\| \cdot \|\vec{y}_c\|$$

צעד 3: חלוקה ב-$\|\vec{x}_c\| \cdot \|\vec{y}_c\|$ נותנת:

$$\left| \frac{\langle \vec{x}_c, \vec{y}_c \rangle}{\|\vec{x}_c\| \cdot \|\vec{y}_c\|} \right| \leq 1$$

צעד 4: אך זה בדיוק $|r|$, לכן:

$$|r| \leq 1 \quad \Rightarrow \quad -1 \leq r \leq 1 \quad \blacksquare$$

מתי מתקיים $r = \pm 1$?
השוויון מתקיים כאשר $\vec{y}_c = c\vec{x}_c$ – כלומר, Y ו-X קשורים ליניארית: $Y = a + bX$.

4.10 דוגמאות ויזואליות: קורלציות שונות

פסאודו-קוד – יצירת דוגמאות קורלציה:
תוצאה צפויה: גרפים המראים כיצד הנקודות "מסתדרות" בצורה ליניארית ככל שהקורלציה חזקה יותר.

4.11 תרגיל תכנות עצמי 4.1 – חישוב קורלציה ידני

מטרה: להבין את הקשר בין קו-וריאנס, קורלציה, ומכפלה סקלרית.
משימה:

1. צרו שני משתנים X ו-Y עם קורלציה ידועה
2. חשבו קו-וריאנס בשלוש דרכים:
   - נוסחה ישירה
   - מכפלה סקלרית של וקטורים ממורכזים

דוגמאות קורלציה עם ויזואליזציה

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)
n = 100
x = np.random.randn(n)

# Different correlations
correlations = [1.0, 0.8, 0.5, 0.0, -0.5, -0.8, -1.0]

fig, axes = plt.subplots(2, 4, figsize=(16, 8))
axes = axes.flatten()

for i, r_target in enumerate(correlations):
    # Create Y with desired correlation
    noise = np.random.randn(n)
    y = r_target * x + np.sqrt(1 - r_target**2) * noise

    # Compute actual correlation
    r_actual = np.corrcoef(x, y)[0, 1]

    # Plot
    axes[i].scatter(x, y, alpha=0.6)
    axes[i].set_title(f'r␣=␣{r_actual:.2f}')
    axes[i].set_xlabel('X')
    axes[i].set_ylabel('Y')
    axes[i].grid(True, alpha=0.3)

axes[-1].axis('off')
plt.tight_layout()
plt.show()
```

- באמצעות NumPy

3. חשבו קורלציה בשלוש דרכים:

- מקו-ווריאנס וסטיות תקן
- Cosine Similarity של וקטורים ממורכזים
- באמצעות NumPy

4. השוו את כל התוצאות

חישוב קורלציה וקו-ווריאנס ידני

```
import numpy as np

x = np.array([160, 170, 180, 175, 165])
y = np.array([55, 65, 75, 70, 60])

# Method 1: Direct formula
x_c = x - np.mean(x)
y_c = y - np.mean(y)
cov_manual = np.mean(x_c * y_c)
r_manual = cov_manual / (np.std(x) * np.std(y))

# Method 2: Cosine similarity (centered vectors)
r_cosine = np.dot(x_c, y_c) / (np.linalg.norm(x_c) * np.linalg.norm(y_c)
)

# Method 3: NumPy
r_numpy = np.corrcoef(x, y)[0, 1]

# All methods give same result: r ≈ 0.99
```

תוצאה צפויה: כל השיטות יתנו אותה תוצאה, מה שמדגים את הקשר העמוק בין קורלציה ומכפלה סקלרית.

4.12 מטריצת הקורלציה

כאשר יש d משתנים, ניתן לארגן את כל הקורלציות במטריצה.

הגדרה 4.3 – מטריצת קורלציה:

עבור d משתנים $X_1, X_2, \ldots, X_d$, מטריצת הקורלציה $\mathbf{R} \in \mathbb{R}^{d\times d}$ מוגדרת כ:

$$\mathbf{R}_{ij} = \mathrm{Cor}(X_i, X_j) \tag{39}$$

תכונות מטריצת הקורלציה:

1. **סימטרית:** $\mathbf{R} = \mathbf{R}^T$ (כי $\text{Cor}(X_i, X_j) = \text{Cor}(X_j, X_i)$)
2. **האלכסון כולו 1:** $\mathbf{R}_{ii} = 1$ (כי $\text{Cor}(X_i, X_i) = 1$)
3. **חיובית חצי-מוגדרת** (Positive Semi-Definite): כל הערכים העצמיים ≥ 0

דוגמה – מטריצת קורלציה לשלושה משתנים:

$$\mathbf{R} = \begin{bmatrix} 1.00 & 0.85 & 0.62 \\ 0.85 & 1.00 & 0.71 \\ 0.62 & 0.71 & 1.00 \end{bmatrix}$$

פרשנות: המשתנים הראשון והשני בעלי קורלציה חזקה (0.85).

שימוש למעשי – זיהוי Multicollinearity:

ברגרסיה ליניארית, אם שני משתנים בלתי תלויים קשורים מאוד ($|r| > 0.9$), זו בעיה שנקראת **Multicollinearity**. המודל לא יכול להפריד בין השפעתם, והמשקלים הופכים לא יציבים.

פתרון: הסר אחד מהמשתנים או השתמש ב-noissergeR egdiR/ACP.

פסאודו-קוד – ויזואליזציה של מטריצת קורלציה:

4.13 הסכנה הגדולה: מתאם אינו סיבתיות

זהו אחד העקרונות החשובים ביותר במדע הנתונים: **"Correlation is not Causation"**.

קורלציה גבוהה בין X ו-Y לא אומרת ש-X גורם ל-Y!

שלוש אפשרויות לקורלציה:

1. **סיבתיות ישירה:** X גורם ל-Y (למשל: עישון ← סרטן ריאות)
2. **סיבתיות הפוכה:** Y גורם ל-X (הפוך ממה שחשבנו!)
3. **משתנה מבלבל** (Confounding Variable): משתנה שלישי Z גורם גם ל-X וגם ל-Y

דוגמאות קלאסיות לקורלציות מזויפות:

טבלה 12: דוגמאות לקורלציות מזויפות

משתנה מבלבל Z	משתנה Y	משתנה X
חום הקיץ	מספר טביעות מים	צריכת גלידה
גודל השריפה	נזק משריפות	מספר כבאים
גיל (ילדים)	יכולת קריאה	גודל נעליים
מקריות סטטיסטית	טביעות בבריכות	מכירות egaC salociN

pamtaeH של מטריצת קורלציה

```python
import seaborn as sns
import pandas as pd

# Example: Iris dataset
from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)

# Correlation matrix
corr_matrix = df.corr()

# Heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', center=0,
            square=True, linewidths=1, cbar_kws={"shrink": 0.8})
plt.title('Correlation␣Matrix␣-␣Iris␣Dataset')
plt.tight_layout()
plt.show()

# Detect Multicollinearity
high_corr = (corr_matrix.abs() > 0.9) & (corr_matrix != 1)
if high_corr.any().any():
    print("Warning:␣High␣correlation␣detected␣(Multicollinearity)")
    print(corr_matrix[high_corr])
```

אתר מפורסם לקורלציות מזויפות: tylervigen.com/spurious-correlations

4.14 מחקר מקרה: עישון וסרטן

ההוכחה שעישון גורם לסרטן ריאות לקחה עשרות שנים, למרות קורלציה ברורה.

הבעיה: חברות הטבק טענו שזו "רק קורלציה" – אולי אנשים עם נטייה גנטית לסרטן גם נוטים לעשן?

ההוכחה הסיבתית דרשה:

1. **מחקרים אפידמיולוגיים רבים** – קורלציה עקבית בכל האוכלוסיות
2. **מחקרים פרוספקטיביים** – מעקב לאורך זמן: עישון ← סרטן (ולא להפך)
3. **מנגנון ביולוגי** – זיהוי החומרים הקרצינוגניים בעשן
4. **מינון-תגובה** – יותר עישון ← יותר סרטן
5. **הפסקת עישון** – מורידה את הסיכון

רק שילוב של כל אלה הוכיח **סיבתיות**.

קריטריונים של ברדפורד היל (Bradford Hill Criteria) (1965) [27]:

טבלה 13: קריטריונים לזיהוי סיבתיות

הסבר	קריטריון
קורלציה חזקה יותר מרמזת על סיבתיות	חוזק
התוצאות חוזרות במחקרים שונים	עקביות
החשיפה גורמת לתוצאה ספציפית	ספציפיות
הגורם קודם לתוצאה בזמן	זמניות
מינון גבוה יותר ← השפעה חזקה יותר	גרדיאנט ביולוגי
קיים מנגנון ביולוגי סביר	סבירות
התוצאות עולות בקנה אחד עם הידע הקיים	קוהרנטיות
התערבות משנה את התוצאה	ניסוי
קיימות תופעות דומות	אנלוגיה

4.15 כלים לזיהוי סיבתיות: Causal Inference

תחום ה**Causal Inference** מציע כלים מתמטיים לזיהוי סיבתיות מנתונים תצפיתיים.

שיטה 1 – ניסויים אקראיים מבוקרים (Randomized Controlled Trials - RCT):

זהו תקן הזהב. חלוקה אקראית לקבוצת ביקורת וטיפול מבטלת משתנים מבלבלים.

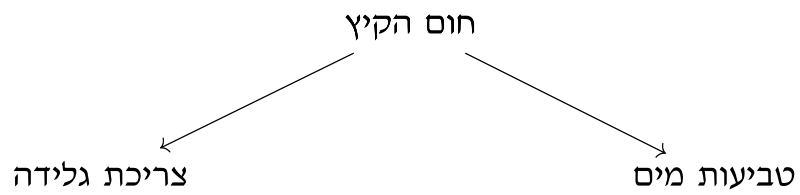
שיטה 2 – גרפים סיבתיים (Causal Graphs / DAGs):

פותחה על ידי Judea Pearl (זוכה פרס טיורינג 2011) [28].

גרף סיבתי הוא גרף מכוון אציקלי (Directed Acyclic Graph - DAG) שבו:

- צמתים = משתנים
- חצים = יחסי סיבה-תוצאה

דוגמה – גלידה, טביעות, וחום:



הגרף מראה ש"חום הקיץ" גורם לשניהם – אין קשר סיבתי ישיר בין גלידה לטביעות.

שיטה 3 – Instrumental Variables:

כאשר RCT לא אפשרי, ניתן להשתמש ב**משתנה אינסטרומנטלי** – משתנה שמשפיע על X אך לא ישירות על Y (רק דרך X).

דוגמה: מחקר השפעת השכלה על הכנסה. משתנה אינסטרומנטלי: מרחק מבית הספר התיכון הקרוב. משפיע על שנות לימוד (מי שגר רחוק נוטה ללמוד פחות), אך לא ישירות על ההכנסה.

4.16 קורלציה חלקית: בידוד השפעה

קורלציה חלקית (Partial Correlation) מודדת את הקשר בין X ו-Y **לאחר שליטה** על משתנה שלישי Z.

הגדרה 4.4 – קורלציה חלקית:

הקורלציה החלקית בין X ו-Y בשליטה על Z מוגדרת כ:

$$r_{XY\cdot Z} = \frac{r_{XY} - r_{XZ}\cdot r_{YZ}}{\sqrt{(1-r_{XZ}^2)(1-r_{YZ}^2)}} \tag{40}$$

פרשנות:

$r_{XY\cdot Z}$ היא הקורלציה בין X ו-Y **אחרי הוצאת ההשפעה של** Z.

דוגמה – גלידה וטביעות:

- $r_{\text{גלידה, טביעות}} = 0.85$ (קורלציה גבוהה!)

- $r_{\text{הדילג, תועיבט-סוח}} = 0.02$ (אחרי שליטה על חום – כמעט אפס!)

זה מוכיח שהקשר מזויף והוא נובע מהחום.

פסאודו-קוד – חישוב קורלציה חלקית:

תוצאה צפויה: הקורלציה הרגילה גבוהה (~ 0.7), אך הקורלציה החלקית כמעט אפס (~ 0.05).

4.17 קורלציה אפס $\neq$ בלתי תלויים

שאלה קריטית: אם $r = 0$, האם X ו-Y בלתי תלויים סטטיסטית?

תשובה: לא!

קורלציה אפס משמעה רק **אין קשר ליניארי**. עדיין יכול להיות קשר לא-ליניארי חזק.

דוגמה קלאסית – קשר ריבועי:

תוצאה: $r \approx 0$, אך $Y = X^2$ – תלות מושלמת!

הסיבה: קורלציה מודדת רק קשרים ליניאריים. $Y = X^2$ הוא קשר סימטרי (עבור X חיובי ושלילי), לכן הקורלציה הליניארית מתאפסת.

פתרון – מדדי תלות לא-פרמטריים:

טבלה 14: מדדי תלות: פרמטריים ולא-פרמטריים

מדד	**סוג**	**מה הוא מודד**
קורלציה של פירסון	פרמטרי	קשר ליניארי בלבד
קורלציה של ספירמן	לא-פרמטרי	קשר מונוטוני (לא בהכרח ליניארי)
קורלציה של קנדל	לא-פרמטרי	קשר מונוטוני (עמיד יותר ל-sreiltuo)
Distance Correlation	לא-פרמטרי	כל סוג תלות (כולל לא-מונוטונית)
Mutual Information	תאוריית מידע	כל סוג תלות (גם לא-דטרמיניסטית)

קורלציה של ספירמן (Spearman's Rank Correlation):

במקום הערכים עצמם, משתמשים **בדירוגים** (Ranks):

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)} \qquad (41)$$

כאשר d_i הוא ההפרש בין דירוגי X ו-Y עבור תצפית i.

יתרון: מזהה קשרים מונוטוניים (עולים/יורדים) גם אם לא ליניאריים.

קורלציה חלקית ב-nohtyP

```python
from scipy.stats import pearsonr

def partial_correlation(x, y, z):
    """
␣␣␣␣Computes␣partial␣correlation␣r_{xy.z}

␣␣␣␣Args:
␣␣␣␣␣␣␣␣x,␣y,␣z:␣data␣vectors

␣␣␣␣Returns:
␣␣␣␣␣␣␣␣float:␣partial␣correlation
␣␣␣␣"""
    # Pairwise correlations
    r_xy, _ = pearsonr(x, y)
    r_xz, _ = pearsonr(x, z)
    r_yz, _ = pearsonr(y, z)

    # Partial correlation formula
    numerator = r_xy - r_xz * r_yz
    denominator = np.sqrt((1 - r_xz**2) * (1 - r_yz**2))

    return numerator / denominator

# Example: ice cream, drownings, heat
np.random.seed(42)
n = 100

# Summer heat (confounding variable)
heat = np.random.randn(n)

# Ice cream and drownings depend on heat
ice_cream = 0.8 * heat + np.random.randn(n) * 0.2
drownings = 0.7 * heat + np.random.randn(n) * 0.3

# Regular correlation
r_regular, _ = pearsonr(ice_cream, drownings)
print(f"Regular␣correlation:␣{r_regular:.3f}")

# Partial correlation (controlling for heat)
r_partial = partial_correlation(ice_cream, drownings, heat)
print(f"Partial␣correlation␣(controlling␣for␣heat):␣{r_partial:.3f}")

# Result: correlation disappears!
```

קורלציה אפס עם תלות מושלמת

```python
import matplotlib.pyplot as plt

# Create quadratic relationship
x = np.linspace(-3, 3, 100)
y = x**2

# Compute correlation
r, _ = pearsonr(x, y)

print(f"Correlation: {r:.6f}")
print("But Y is completely dependent on X!")

# Plot
plt.figure(figsize=(8, 6))
plt.scatter(x, y, alpha=0.6)
plt.xlabel('X')
plt.ylabel('Y = X²')
plt.title(f'Perfect Dependency, Zero Correlation (r = {r:.3f})')
plt.grid(True, alpha=0.3)
plt.show()
```

תוצאה צפויה: - עבור $Y = X^2$: פירסון ≈ 0, ספירמן ≈ 0 (לא מונוטוני) - עבור $Y = e^X$: פירסון ≈ 0.8, ספירמן $= 1.0$ (מונוטוני מושלם)

4.18 תרגיל תכנות עצמי 4.2 – גילוי קורלציות מזויפות

מטרה: להבין כיצד משתנה מבלבל יוצר קורלציה מזויפת.

משימה:

1. צרו סימולציה של "גלידה, טביעות, חום"
2. חשבו קורלציה רגילה וקורלציה חלקית
3. הציגו גרפית איך הקורלציה נעלמת לאחר שליטה
4. נסו עם משתנים מבלבלים שונים

השוואת קורלציית פירסון וספירמן

```python
from scipy.stats import spearmanr

# Quadratic relationship
x = np.linspace(-3, 3, 100)
y = x**2

# Pearson
r_pearson, _ = pearsonr(x, y)

# Spearman
r_spearman, _ = spearmanr(x, y)

print(f"Pearson:␣{r_pearson:.4f}")
print(f"Spearman:␣{r_spearman:.4f}")

# Exponential relationship (monotonic)
x_exp = np.linspace(0, 5, 100)
y_exp = np.exp(x_exp)

r_pearson_exp, _ = pearsonr(x_exp, y_exp)
r_spearman_exp, _ = spearmanr(x_exp, y_exp)

print(f"\nExponential␣relationship:")
print(f"Pearson:␣{r_pearson_exp:.4f}")
print(f"Spearman:␣{r_spearman_exp:.4f}")
```

סימולציה: גילוי קורלציות מזויפות

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Confounding variable: temperature
temperature = np.random.uniform(15, 35, 200)
ice_cream = 2 * temperature + np.random.randn(200) * 5
drownings = 1.5 * temperature + np.random.randn(200) * 3

# Naive correlation (spurious)
r_naive = np.corrcoef(ice_cream, drownings)[0, 1]

# Partial correlation (after removing temperature)
ice_res = ice_cream - LinearRegression().fit(temperature.reshape(-1, 1),
 ice_cream).predict(temperature.reshape(-1, 1))
drown_res = drownings - LinearRegression().fit(temperature.reshape(-1,
1), drownings).predict(temperature.reshape(-1, 1))
r_partial = np.corrcoef(ice_res, drown_res)[0, 1]

# Visualization
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))
ax1.scatter(ice_cream, drownings, c=temperature, cmap='hot', alpha=0.6)
ax1.set_title(f'Naive:␣r␣=␣{r_naive:.3f}')
ax2.scatter(ice_res, drown_res, alpha=0.6)
ax2.set_title(f'After␣Control:␣r␣=␣{r_partial:.3f}')
plt.show()
```

תוצאה צפויה: הגרף השמאלי מראה קורלציה חזקה, אך הגרף הימני (אחרי שליטה) מראה שאין קשר אמיתי.

4.19 סיכום ומבט קדימה

מה למדנו בפרק זה?

1. **קו-ווריאנס היא מכפלה סקלרית** של וקטורים ממורכזים – הרחבה של מושגי פרק 1
2. **קורלציה היא Cosine Similarity** של וקטורים ממורכזים – גרסה מנורמלת ללא תלות ביחידות
3. **תחום הקורלציה:** $[-1, 1]$ – הוכחנו באמצעות אי-שוויון קושי-שוורץ
4. **מתאם אינו סיבתיות!** – משתנים מבלבלים יוצרים קורלציות מזויפות
5. **קורלציה חלקית** – כלי לבידוד השפעה ושליטה על משתנים מבלבלים

6. **קורלציה אפס $\neq$ בלתי תלויים** – קיימים קשרים לא-ליניאריים

7. **מדדים לא-פרמטריים** – ספירמן, קנדל, Mutual Information למצבים מורכבים

מבט קדימה – פרק 5:
בפרק הבא נעבור מיחסים בין משתנים ל**אופטימיזציה**. נחקור:

- **רגרסיה ליניארית כבעיית אופטימיזציה** – מזעור פונקציית ה-ESM

- **פתרון אנליטי** – נוסחת ה-noitauqE lamroN

- **פתרון איטרטיבי** – Gradient Descent וגרסאות שלו

- **התכנסות ויציבות** – מתי האלגוריתם מצליח?

- **קשר למכפלה סקלרית** – גרדיאנט כווקטור אורתוגונלי

שאלת מחשבה לסיום:
אם X ו-Y בעלי קורלציה $r = 0.9$, ו-Y ו-Z בעלי קורלציה $r = 0.9$, האם בהכרח X ו-Z יהיו בעלי קורלציה גבוהה?
רמז: לא! קורלציה אינה טרנזיטיבית. אפשר לבנות דוגמה נגדית שבה $r_{XZ} = 0$.

מטלות וקריאה מורחבת

תרגיל 4.1: חשבו קו-ווריאנס וקורלציה ידנית (ראו פסאודו-קוד).
תרגיל 4.2: סמלצו קורלציה מזויפת עם משתנה מבלבל (ראו פסאודו-קוד).
תרגיל 4.3: הוכיחו שמטריצת קורלציה היא תמיד חיובית חצי-מוגדרת.
רמז: השתמשו בעובדה שלכל וקטור $\vec{v}$, מתקיים $\vec{v}^T\mathbf{R}\vec{v} \geq 0$.
קריאה מורחבת:

- [24] – "Mathematical Contributions to the Theory of Evolution": המאמר המקורי על קורלציה

- [27] – "The Environment and Disease: Association or Causation?": הקריטריונים המפורסמים לסיבתיות

- [28] – "Causality: Models, Reasoning, and Inference": הספר המקיף על Causal Inference

- [29] – "Spurious Correlations": אוסף משעשע של קורלציות מזויפות

שאלות להעמקה:

1. מדוע קורלציה של ספירמן עמידה יותר ל-sreiltuo מקורלציה של פירסון?

2. האם שתי קורלציות חלקיות $r_{XY \cdot Z}$ ו-$r_{XY \cdot W}$ יכולות לתת תוצאות שונות?

3. כיצד Mutual Information יכולה לזהות תלות שקורלציה מפספסת?

סיום פרק 4

5 רגרסיה ליניארית: הקו שעצב את המודרניות

Linear Regression: The Line That Shaped Modernity

ד"ר יורם סגל

ספטמבר 5202 - שיעור 70

5.1 פרולוג: הקו שחיזה את המציאות

בליל ה-1 בינואר 1801, האסטרונום האיטלקי ג'וזפה פיאצי (Giuseppe Piazzi) גילה נקודת אור קטנה שנעה לאט על רקע הכוכבים. הוא מדד את מיקומה במשך 41 לילות, עד שהשמש עלתה והעלימה אותה מהעין. העולם המדעי היה נסער - האם זה כוכב לכת חדש? והחשוב מכל: איפה הוא יופיע שוב כשהשמש תרד?

הבעיה היתה מתמטית טהורה. פיאצי היה מחזיק ב-41 נקודות מדידה, כל אחת עם שגיאות מדידה קטנות. היה צריך למצוא את המסלול - עקומה מתמטית שמתארת את תנועת הגוף השמיימי החדש. מתמטיקאים גדולים ניסו ונכשלו.

עד שצעיר בן 24, קרל פרידריך גאוס (Carl Friedrich Gauss), פרסם חישוב שהדהים את העולם המדעי. הוא לא רק חיזה היכן יופיע הגוף - הוא צדק בדיוק מדהים. השיטה שלו? מה שאנחנו מכירים היום כ**רגרסיה ליניארית** ושיטת **הריבועים הפחותים** (Least Squares).

הקו שגאוס משך דרך נקודות המדידה של פיאצי לא היה רק כלי מתמטי. הוא היה ביטוי לאמונה פילוסופית עמוקה: שהמציאות, למרות הרעש והאי-ודאות, ניתנת לחיזוי. שמאחורי הכאוס של המדידות מסתתר סדר מתמטי. שניתן, באמצעות הכלים הנכונים, לראות את הבלתי נראה.

היום, יותר מ-220 שנה אחרי ההישג של גאוס, רגרסיה ליניארית היא אבן היסוד של כמעט כל מערכת בינה מלאכותית. מנועי החיפוש של Google, מערכות ההמלצות של Netflix, הרכב האוטונומי, אבחון רפואי - כולם בנויים על עקרונות שגאוס גילה כשניסה למצוא כוכב לכת אבוד.

זה הפרק שבו נבין איך זה עובד.

5.2 תחילת המסע: מגאוס לגוגל

הסיפור של רגרסיה ליניארית הוא סיפור של תחרות מדעית, גאווה אישית, וויכוח על קדימות שנמשך עשרות שנים.

המתחרים:

ב-1805, הצרפתי אדריאן-מארי לז'נדר (Adrien-Marie Legendre) פרסם את שיטת הריבועים הפחותים בספרו על מסלולי שביטים [30]. הוא תיאר בבירור את העיקרון: מצא את הקו שממזער את סכום ריבועי השגיאות.

אבל גאוס טען שהוא השתמש בשיטה כבר ב-1795 - עשר שנים לפני הפרסום של לז'נדר! הוא סירב לפרסם את השיטה כי חיפש הצדקה תיאורטית מושלמת. רק ב-1809, בספרו "Theoria Motus" [31], גאוס פרסם לא רק את השיטה, אלא גם את ההצדקה הסטטיסטית המלאה שלה.

הוויכוח על הקרדיט היה מר. לז'נדר, שכבר היה בשנות ה-50 לחייו, חש שנשדד. גאוס, הצעיר והגאה, סירב להתנצל. עד היום, ספרי ההיסטוריה חלוקים: יש המייחסים את ההמצאה ללז'נדר (הראשון שפרסם), ויש המייחסים אותה לגאוס (שפיתח את התאוריה המלאה).

הקפיצה המושגית:

מה שמרתק בסיפור הזה הוא לא רק הויכוח האישי, אלא הקפיצה המושגית. לפני גאוס ולז'נדר, מדענים ניסו למצוא קשרים מדויקים בין משתנים. המהפכה היתה להבין שבעולם האמיתי, עם שגיאות מדידה ורעש, אנחנו לא מחפשים אמת מדויקת אלא קירוב הטוב ביותר.

זו אותה קפיצה שמאפשרת היום לבינה מלאכותית לפעול. מודלי GPT לא "יודעים" את התשובה הנכונה - הם מוצאים את התשובה הסבירה ביותר על סמך הנתונים. זהו היסוד הפילוסופי של כל למידת מכונה מודרנית.

5.3 השאלה המרכזית: מהו "הקו הטוב ביותר"?

דמיינו שאתם עומדים מול לוח עם פיזור של נקודות. כל נקודה מייצגת תצפית - למשל, גובה ומשקל של אדם, או שנות לימוד והכנסה. אתם רוצים למשוך קו דרך הנקודות האלה. אבל איזה קו?

אם תבקשו מעשרה אנשים למשוך קו "באופן אינטואיטיבי", תקבלו עשרה קווים שונים. כולם יהיו "סבירים", אבל שונים. השאלה היא: **האם יש קו אחד שהוא "הכי טוב"?**

זוהי שאלה מתמטית עמוקה שמסתתרת מאחוריה שאלה פילוסופית: מה המשמעות של "טוב"?

הגדרות אפשריות של "טוב":

נניח שיש לנו n נקודות: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, ואנחנו מחפשים קו $\hat{y} = w_0 + w_1 x$.

עבור כל נקודה, השגיאה היא $e_i = y_i - \hat{y}_i = y_i - (w_0 + w_1 x_i)$.

איך נמדוד עד כמה הקו "טוב"? יש כמה אפשרויות:

אפשרות 1 - סכום השגיאות: $\sum_{i=1}^{n} e_i$

בעיה: שגיאות חיוביות ושליליות מבטלות זו את זו. קו שעובר הרחק מכל הנקודות, אבל "מאוזן", יקבל ציון מושלם.

אפשרות 2 - סכום ערכים מוחלטים: $\sum_{i=1}^{n} |e_i|$

זה עובד! זה נקרא Mean Absolute Error (MAE). אבל יש בעיה טכנית: הפונקציה המוחלטת אינה גזירה בנקודה $x = 0$, מה שמסבך את האופטימיזציה.

אפשרות 3 - סכום ריבועי השגיאות: $\sum_{i=1}^{n} e_i^2$

זה הבחירה של גאוס ולז'נדר. למה?

5.4 פונקציית ההפסד: למה דווקא ריבוע?

גאוס לא בחר את הריבוע באקראי. היו לו ארבע סיבות מתמטיות עמוקות:

סיבה 1 - גזירות:

הפונקציה $f(x) = x^2$ גזירה בכל מקום, והנגזרת שלה פשוטה: $f'(x) = 2x$. זה הופך את בעיית האופטימיזציה לפתירה אנליטית. הפונקציה המוחלטת $|x|$ אינה גזירה ב-$x = 0$, מה שמסבך את המתמטיקה.

סיבה 2 - הטיה כלפי שגיאות גדולות:

ריבוע מעניש שגיאות גדולות הרבה יותר משגיאות קטנות. שגיאה של 2 נותנת $2^2 = 4$, אבל שגיאה של 10 נותנת $10^2 = 100$ - פי 25 יותר! זה אומר שהמודל "ירצה" להימנע משגיאות גדולות.

בואו נראה דוגמה מספרית:

טבלה 15: השוואת עונשים: MAE מול MSE

שגיאה	**MAE ($|e|$)**	**MSE (e^2)**
1	1	1
2	2	4
5	5	25
10	10	100

שימו לב: ב-ESM, שגיאה של 10 היא פי 10 יותר גרועה משגיאה של 5, אבל ב-EAM היא רק פי 2 יותר גרועה.

סיבה 3 - הצדקה הסתברותית:

גאוס הוכיח משהו מדהים: אם שגיאות המדידה מתפלגות נורמלית (התפלגות גאוס!), אז מזעור סכום ריבועי השגיאות הוא בדיוק שיטת ה-doohilekiL mumixaM - השיטה הסטטיסטית המושלמת למציאת הפרמטרים הטובים ביותר.

הקשר העמוק הזה בין ריבוע השגיאות להתפלגות הנורמלית הוא אחד היופי של המתמטיקה. נראה את ההוכחה:

משפט 5.1 - שקילות MSE ו-doohilekiL mumixaM:

אם שגיאות המדידה $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ (מתפלגות נורמלית), אזי מזעור MSE שקול למקסום ה-doohilekiL.

הוכחה:

נניח $y_i = w_0 + w_1 x_i + \epsilon_i$ כאשר $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$.

פונקציית ה-doohilekiL עבור תצפית בודדת:

$$p(y_i|x_i, w_0, w_1) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - w_0 - w_1 x_i)^2}{2\sigma^2}\right) \tag{42}$$

ה-doohilekiL עבור כל הנתונים (בהנחת בלתי-תלות):

$$L(w_0, w_1) = \prod_{i=1}^{n} p(y_i|x_i, w_0, w_1) \tag{43}$$

Log-Likelihood (נוח יותר לאופטימיזציה):

$$\begin{aligned}
\log L(w_0, w_1) &= \sum_{i=1}^{n} \log p(y_i|x_i, w_0, w_1) \\
&= \sum_{i=1}^{n} \left[\log \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{(y_i - w_0 - w_1 x_i)^2}{2\sigma^2}\right] \\
&= -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - w_0 - w_1 x_i)^2
\end{aligned} \tag{44}$$

מקסום Log-Likelihood שקול למזעור:

$$\sum_{i=1}^{n}(y_i - w_0 - w_1 x_i)^2 \tag{45}$$

שזה בדיוק סכום ריבועי השגיאות! ■

סיבה 4 - קשר לאלגברה ליניארית:

הריבוע יוצר קשר ישיר למכפלה סקלרית ולנורמה L_2. זה מאפשר לנסח את הבעיה כ**הטלה אורתוגונלית** במרחב וקטורי - גישה גיאומטרית עמוקה שנראה בהמשך.

הגדרה פורמלית - Mean Squared Error (MSE):

$$\text{MSE}(w_0, w_1) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - w_0 - w_1 x_i)^2 \tag{46}$$

המטרה שלנו: למצוא w_0^*, w_1^* שממזערים את MSE.

5.5 הכלי המתמטי: גזירה חלקית כמצפן

איך מוצאים את המינימום של פונקציה? אם הייתה לנו פונקציה של משתנה אחד, $f(x)$, היינו גוזרים ומשווים לאפס: $f'(x) = 0$.

אבל כאן יש לנו פונקציה של שני משתנים: $\text{MSE}(w_0, w_1)$. צריך כלי חדש: **גזירה חלקית** (Partial Derivative).

האינטואיציה של גזירה חלקית:

דמיינו שאתם עומדים על הר. הגובה שלכם הוא פונקציה של שני משתנים: קו אורך וקו רוחב, $h(x, y)$. גזירה חלקית לפי x עונה על השאלה: "אם אני הולך צעד אחד מזרחה

(כיוון x), כמה הגובה משתנה?"
באופן פורמלי, הנגזרת החלקית של $f(x,y)$ לפי x היא:

$$\frac{\partial f}{\partial x} = \lim_{h \to 0} \frac{f(x+h,y) - f(x,y)}{h} \tag{47}$$

שימו לב: אנחנו מחזיקים את y קבוע ומשנים רק את x.
כללי גזירה חלקית:
כמו בגזירה רגילה, יש כללים:

טבלה 16: כללי גזירה חלקית

פונקציה	**נגזרת חלקית לפי x**
$f(x,y) = x^2 + y^2$	$\frac{\partial f}{\partial x} = 2x$
$f(x,y) = xy$	$\frac{\partial f}{\partial x} = y$
$f(x,y) = x^2y + xy^2$	$\frac{\partial f}{\partial x} = 2xy + y^2$

העיקרון: **התייחס לכל המשתנים האחרים כאל קבועים**.
הגרדיאנט - וקטור הכיוון:
כשיש n משתנים, אוספים את כל הנגזרות החלקיות לווקטור אחד שנקרא **גרדיאנט** (Gradient):

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial w_0} \\ \frac{\partial f}{\partial w_1} \\ \vdots \\ \frac{\partial f}{\partial w_n} \end{bmatrix} \tag{48}$$

הגרדיאנט מצביע על **כיוון העלייה המהירה ביותר**. לכן, כדי לרדת (למצוא מינימום), נלך בכיוון ההפוך לגרדיאנט: $-\nabla f$.
זה היסוד של Gradient Descent - אבל לפני שנגיע לשם, נמצא את הפתרון האנליטי.

5.6 הפתרון המושלם: Normal Equation

אנחנו מחפשים את w_0^*, w_1^* שממזערים:

$$\text{MSE}(w_0, w_1) = \frac{1}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i)^2 \tag{49}$$

צעד 1: נגזרות חלקיות
נגזור לפי w_0:

$$\frac{\partial \text{MSE}}{\partial w_0} = \frac{\partial}{\partial w_0}\left[\frac{1}{n}\sum_{i=1}^{n}(y_i - w_0 - w_1 x_i)^2\right]$$
$$= \frac{1}{n}\sum_{i=1}^{n} 2(y_i - w_0 - w_1 x_i)\cdot(-1)$$
$$= -\frac{2}{n}\sum_{i=1}^{n}(y_i - w_0 - w_1 x_i) \tag{50}$$

נגזור לפי w_1:

$$\frac{\partial \text{MSE}}{\partial w_1} = \frac{\partial}{\partial w_1}\left[\frac{1}{n}\sum_{i=1}^{n}(y_i - w_0 - w_1 x_i)^2\right]$$
$$= \frac{1}{n}\sum_{i=1}^{n} 2(y_i - w_0 - w_1 x_i)\cdot(-x_i)$$
$$= -\frac{2}{n}\sum_{i=1}^{n} x_i(y_i - w_0 - w_1 x_i) \tag{51}$$

צעד 2: השוואה לאפס

כדי למצוא מינימום, נשווה את הנגזרות לאפס:

$$\begin{cases} \sum_{i=1}^{n}(y_i - w_0 - w_1 x_i) = 0 \\ \sum_{i=1}^{n} x_i(y_i - w_0 - w_1 x_i) = 0 \end{cases} \tag{52}$$

צעד 3: פתיחת הסכומים

מהמשוואה הראשונה:

$$\sum_{i=1}^{n} y_i - n w_0 - w_1 \sum_{i=1}^{n} x_i = 0$$
$$n\bar{y} - n w_0 - n w_1 \bar{x} = 0$$
$$w_0 = \bar{y} - w_1 \bar{x} \tag{53}$$

זו משוואה יפהפייה שאומרת: **הקו חייב לעבור דרך נקודת הממוצע** $(\bar{x}, \bar{y})$.

צעד 4: הצבה במשוואה השנייה

נציב $w_0 = \bar{y} - w_1 \bar{x}$ במשוואה השנייה:

$$\sum_{i=1}^{n} x_i(y_i - (\bar{y} - w_1\bar{x}) - w_1 x_i) = 0$$

$$\sum_{i=1}^{n} x_i(y_i - \bar{y}) - w_1 \sum_{i=1}^{n} x_i(\bar{x} - x_i) = 0$$

$$\sum_{i=1}^{n} x_i(y_i - \bar{y}) = w_1 \sum_{i=1}^{n} x_i(x_i - \bar{x}) \tag{54}$$

צעד 5: הפתרון הסופי

$$w_1^* = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{55}$$

$$w_0^* = \bar{y} - w_1^*\bar{x} \tag{56}$$

זיהוי מדהים:

המונה ב-w_1^* הוא **קו-ווריאנס** בין X ו-Y!

המכנה הוא **שונות** של X!

לכן:

$$w_1^* = \frac{\text{Cov}(X,Y)}{\text{Var}(X)} = \frac{\text{Cov}(X,Y)}{\sigma_X^2} = r \cdot \frac{\sigma_Y}{\sigma_X} \tag{57}$$

כאשר r הוא מקדם הקורלציה! הנוסחה מקשרת בין רגרסיה לקורלציה בצורה אלגנטית.

פסאודו-קוד - חישוב ידני של רגרסיה:

5.7 נוסחה מטריצית: אלגנטיות האלגברה הליניארית

הפתרון שמצאנו יפה, אבל הוא מוגבל לרגרסיה עם משתנה בלתי תלוי אחד. מה קורה כשיש לנו d תכונות?

זה הזמן לעבור לנוסחה המטריצית - אחת הנוסחאות האלגנטיות והעוצמתיות ביותר בלמידת מכונה.

סימונים מטריציים:

נניח שיש לנו n תצפיות ו-d תכונות. נארגן את הנתונים במטריצה:

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,d} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n,1} & x_{n,2} & \cdots & x_{n,d} \end{bmatrix} \in \mathbb{R}^{n\times(d+1)} \tag{58}$$

עמודת האחדים הראשונה היא עבור האיבר החופשי w_0.

רגרסיה ליניארית - פתרון אנליטי

```python
import numpy as np

def linear_regression_analytical(x, y):
    """
␣␣␣␣Computes␣linear␣regression␣analytically.

␣␣␣␣Returns:
␣␣␣␣␣␣␣␣w0,␣w1:␣intercept␣and␣slope
␣␣␣␣"""
    n = len(x)

    # Means
    x_mean = np.mean(x)
    y_mean = np.mean(y)

    # Method 1: direct formula
    numerator = np.sum((x - x_mean) * (y - y_mean))
    denominator = np.sum((x - x_mean)**2)

    w1 = numerator / denominator
    w0 = y_mean - w1 * x_mean

    return w0, w1

# Example
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 5, 4, 5])

w0, w1 = linear_regression_analytical(x, y)
print(f"y␣=␣{w0:.2f}␣+␣{w1:.2f}x")

# Compare to sklearn
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x.reshape(-1, 1), y)
print(f"sklearn:␣y␣=␣{model.intercept_:.2f}␣+␣{model.coef_[0]:.2f}x")
```

וקטור המשקלים:

$$\vec{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} \in \mathbb{R}^{d+1} \tag{59}$$

וקטור התוויות:

$$\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \in \mathbb{R}^{n} \tag{60}$$

החיזוי במטריצות:

$$\hat{\vec{y}} = \mathbf{X}\vec{w} \tag{61}$$

פונקציית ההפסד במטריצות:

$$\begin{aligned} \text{MSE}(\vec{w}) &= \frac{1}{n}\|\vec{y} - \hat{\vec{y}}\|^2 \\ &= \frac{1}{n}\|\vec{y} - \mathbf{X}\vec{w}\|^2 \\ &= \frac{1}{n}(\vec{y} - \mathbf{X}\vec{w})^T(\vec{y} - \mathbf{X}\vec{w}) \end{aligned} \tag{62}$$

גזירה במטריצות:

כדי למצוא את המינימום, נגזור לפי $\vec{w}$ ונשווה לאפס. יש לנו כלל גזירה מטריצי:

$$\frac{\partial}{\partial \vec{w}}(\vec{y} - \mathbf{X}\vec{w})^T(\vec{y} - \mathbf{X}\vec{w}) = -2\mathbf{X}^T(\vec{y} - \mathbf{X}\vec{w}) \tag{63}$$

השוואה לאפס:

$$\begin{aligned} -2\mathbf{X}^T(\vec{y} - \mathbf{X}\vec{w}) &= \vec{0} \\ \mathbf{X}^T\vec{y} - \mathbf{X}^T\mathbf{X}\vec{w} &= \vec{0} \\ \mathbf{X}^T\mathbf{X}\vec{w} &= \mathbf{X}^T\vec{y} \end{aligned} \tag{64}$$

זו נקראת ה**Normal Equation** - משוואה נורמלית.

הפתרון:

אם $\mathbf{X}^T\mathbf{X}$ הפיכה (דטרמיננטה $\neq 0$), אזי:

$$\vec{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\vec{y} \tag{65}$$

זוהי **נוסחת הזהב של רגרסיה ליניארית.** נוסחה סגורה, אלגנטית, שפותרת את הבעיה במכה אחת.

מורכבות חישובית:

חישוב $(\mathbf{X}^T\mathbf{X})^{-1}$ דורש $O(d^3)$ פעולות (היפוך מטריצה).

חישוב $\mathbf{X}^T\mathbf{X}$ דורש $O(nd^2)$ פעולות.

סה"כ: $O(nd^2 + d^3)$.

זה מעולה כאשר d קטן (עשרות, מאות). אבל כאשר d עצום (מיליונים, כמו ברשתות נוירונים), זה לא מעשי. זו הסיבה שנזקק ל-tnecseD tneidarG.

פסאודו-קוד - פתרון מטריצי:

5.8 הדרך האיטרטיבית: Gradient Descent

הנוסחה הסגורה $\vec{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\vec{y}$ מושלמת מבחינה מתמטית, אבל יש לה שלוש בעיות מעשיות:

בעיה 1 - סיבוכיות: כאשר d גדול (מיליוני פרמטרים), חישוב $(\mathbf{X}^T\mathbf{X})^{-1}$ לא מעשי.

בעיה 2 - זיכרון: המטריצה $\mathbf{X}^T\mathbf{X}$ היא בגודל $d \times d$. אם $d = 10^6$, זה 10^{12} מספרים - טרה-בייט של זיכרון!

בעיה 3 - יציבות: אם $\mathbf{X}^T\mathbf{X}$ כמעט סינגולרית (קרובה לאי-הפיכה), ההיפוך יוצר שגיאות נומריות עצומות.

הפתרון: **Gradient Descent** - שיטה איטרטיבית שלא דורשת היפוך מטריצות.

האינטואיציה - ירידה מההר:

דמיינו שאתם עומדים על הר בערפל. אתם לא רואים את כל ההר, רק את השיפוע מתחת לרגליכם. איך תמצאו את העמק (המינימום)?

תלכו בכיוון הירידה החזה ביותר - הכיוון של שיפוע שלילי מקסימלי. תחזרו על זה שוב ושוב, עד שתגיעו למקום שבו השיפוע אפס - המינימום.

זה בדיוק Gradient Descent.

האלגוריתם:

צעד *1*: אתחול - התחל מנקודה אקראית $\vec{w}^{(0)}$

צעד *2*: חישוב גרדיאנט - חשב את הגרדיאנט (כיוון העלייה):

$$\nabla_{\vec{w}}\text{MSE} = -\frac{2}{n}\mathbf{X}^T(\vec{y} - \mathbf{X}\vec{w}) \tag{66}$$

צעד *3*: צעד בכיוון ההפוך - עדכון:

$$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \alpha\nabla_{\vec{w}}\text{MSE} \tag{67}$$

כאשר α הוא **learning rate** - גודל הצעד.

צעד *4*: חזרה - חזור לצעד 2 עד התכנסות.

קצב הלמידה - האיזון העדין:

α קטן מדי $\leftarrow$ ההתכנסות איטית (אלפי איטרציות)

רגרסיה מרובה - Normal Equation

```
import numpy as np

def linear_regression_normal_equation(X, y):
    """
␣␣␣␣Solves␣linear␣regression␣using␣Normal␣Equation.

␣␣␣␣Args:
␣␣␣␣␣␣␣␣X:␣feature␣matrix␣(n,␣d)␣-␣without␣column␣of␣1s
␣␣␣␣␣␣␣␣y:␣label␣vector␣(n,)

␣␣␣␣Returns:
␣␣␣␣␣␣␣␣w:␣weight␣vector␣(d+1,)
␣␣␣␣"""
    # Add column of 1s for intercept
    X_with_intercept = np.column_stack([np.ones(len(X)), X])

    # Normal Equation: w = (X^T X)^{-1} X^T y
    XtX = X_with_intercept.T @ X_with_intercept
    Xty = X_with_intercept.T @ y

    w = np.linalg.solve(XtX, Xty)  # More efficient than inv()

    return w

# Example with 3 features
np.random.seed(42)
n, d = 100, 3

X = np.random.randn(n, d)
true_w = np.array([5, 2, -3, 1])  # [intercept, w1, w2, w3]
y = np.column_stack([np.ones(n), X]) @ true_w + np.random.randn(n) * 0.5

# Solve
w_estimated = linear_regression_normal_equation(X, y)

print("True␣weights:", true_w)
print("Estimated␣weights:", w_estimated)
print("Error:", np.linalg.norm(true_w - w_estimated))
```

α גדול מדי ← האלגוריתם "קופץ" ולא מתכנס
α אופטימלי ← התכנסות מהירה ויציבה

טבלה 17: השפעת learning rate

Learning Rate	התנהגות
קטן מדי ($\alpha = 10^{-6}$)	התכנסות איטית מאוד, דורש אלפי איטרציות
אופטימלי ($\alpha = 10^{-2}$)	התכנסות מהירה ויציבה
גדול מדי ($\alpha = 10^{1}$)	קפיצות, חוסר התכנסות, פיצוץ לאינסוף

פסאודו-קוד - Gradient Descent מלא:

Gradient Descent - מימוש מלא

```
import numpy as np

def gradient_descent(X, y, alpha=0.01, max_iters=1000, tol=1e-6):
    X_with_intercept = np.column_stack([np.ones(len(X)), X])
    n, d = X_with_intercept.shape
    w = np.random.randn(d) * 0.01

    for iteration in range(max_iters):
        y_pred = X_with_intercept @ w
        gradient = -2/n * X_with_intercept.T @ (y - y_pred)
        w_new = w - alpha * gradient

        if np.linalg.norm(w_new - w) < tol:
            break
        w = w_new

    return w

# Usage: w = gradient_descent(X_train, y_train, alpha=0.01)
```

5.9 וריאציות מודרניות: מ-hctaB ל-citsahcotS

Gradient Descent הקלאסי מחשב את הגרדיאנט על כל הנתונים בכל איטרציה. כאשר n עצום (מיליוני דגימות), זה איטי מאוד.

Stochastic Gradient Descent (SGD):

במקום לחשב את הגרדיאנט על כל הנתונים, בחר דגימה אחת אקראית $(\vec{x}_i, y_i)$ ועדכן לפיה:

$$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \alpha \cdot 2(\vec{x}_i^T \vec{w}^{(t)} - y_i)\vec{x}_i \tag{68}$$

יתרונות:

- מהיר מאוד - עדכון אחרי כל דגימה
- יכול "לברוח" ממינימומים מקומיים בגלל הרעש

חסרונות:

- רועש - הגרדיאנט משתנה מאוד בין איטרציות
- לא מתכנס למינימום מדויק, אלא "מרפרף" סביבו

Mini-Batch Gradient Descent:
הפשרה: בחר קבוצה קטנה של b דגימות (hctab) ועדכן לפיה:

$$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \frac{\alpha}{b} \sum_{i \in \text{batch}} 2(\vec{x}_i^T \vec{w}^{(t)} - y_i)\vec{x}_i \tag{69}$$

זה האיזון האופטימלי - מספיק מהיר, מספיק יציב. זו השיטה הסטנדרטית ב-peeD .gninraeL

גדלי batch נפוצים: 32, 64, 128, 256

טבלה 18: השוואת שיטות Gradient Descent

שיטה	Batch Size	יתרונות	חסרונות
Batch GD	n	יציב, מדויק	איטי, דורש זיכרון רב
SGD	1	מהיר מאוד	רועש, לא מתכנס למדויק
Mini-Batch	32-256	איזון טוב	צריך לכוונן גודל

5.10 אופטימיזרים מתקדמים: מעבר ל-DG allinaV

Gradient Descent הפשוט (שנקרא "Vanilla GD") לא משתמש בשום מידע מהעבר. כל איטרציה עומדת בפני עצמה. זה לא אופטימלי.

Momentum - תנופה:

דמיינו כדור שמתגלגל במורד הר. הוא לא רק נע בכיוון השיפוע הנוכחי - הוא גם זוכר את המהירות שלו. זה עוזר לו "לדהור" דרך עמקים שטוחים ולא להיתקע.

$$\vec{v}^{(t+1)} = \beta \vec{v}^{(t)} + \alpha \nabla_{\vec{w}} \text{MSE} \tag{70}$$

$$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \vec{v}^{(t+1)} \tag{71}$$

$\beta \approx 0.9$ הוא **מקדם התנופה**.

Adam - Adaptive Moment Estimation:

האופטימיזר המודרני הפופולרי ביותר. הוא משלב:

- Momentum - זוכר את הכיוון הכללי
- RMSProp - מתאים את קצב הלמידה לכל פרמטר בנפרד

האלגוריתם מורכב, אבל התוצאות מדהימות - התכנסות מהירה ויציבה.

Adam Optimizer - שימוש ב-hcroTyP

```
import torch
import torch.nn as nn
import torch.optim as optim

# טושפהיסרגרלדומתרדגה
model = nn.Linear(in_features=3, out_features=1)

# רזימיטפוא Adam
optimizer = optim.Adam(model.parameters(), lr=0.01)

# ןומיאתאלול
for epoch in range(100):
    # Forward pass
    y_pred = model(X_tensor)
    loss = nn.MSELoss()(y_pred, y_tensor)

    # Backward pass
    optimizer.zero_grad()  # םיטנאידרגספא
    loss.backward()        # םיטנאידרגבשח
    optimizer.step()       # םילקשמןכדע

    if epoch % 10 == 0:
        print(f"Epoch␣{epoch},␣Loss:␣{loss.item():.4f}")
```

5.11 הקשר העמוק: גיאומטריה של אופטימיזציה

רגרסיה ליניארית אינה רק טכניקה חישובית - היא תובנה גיאומטרית עמוקה.

המרחב הווקטורי של התצפיות:

חשבו על כל תצפית כווקטור במרחב n-ממדי. המטריצה $\mathbf{X}$ פורשת **תת-מרחב** - כל השילובים הליניאריים של העמודות שלה:

$$\text{span}(\mathbf{X}) = \left\{\mathbf{X}\vec{w} : \vec{w} \in \mathbb{R}^{d+1}\right\} \tag{72}$$

הווקטור $\vec{y}$ (התוויות האמיתיות) בדרך כלל לא נמצא בתת-מרחב הזה - יש רעש, שגיאות מדידה, משתנים חסרים.

השאלה הגיאומטרית:

מהי הנקודה ב-$\text{span}(\mathbf{X})$ שהכי קרובה ל-$\vec{y}$?

זו שאלת **הטלה אורתוגונלית** (Orthogonal Projection). התשובה: הנקודה $\hat{\vec{y}} = \mathbf{X}\vec{w}^*$ כך שהווקטור $\vec{y} - \hat{\vec{y}}$ אורתוגולי לכל $\text{span}(\mathbf{X})$.

משפט 5.2 - הטלה אורתוגונלית:

הפתרון של רגרסיה ליניארית הוא ההטלה האורתוגונלית של $\vec{y}$ על $\text{span}(\mathbf{X})$.

הוכחה:

תנאי האורתוגונליות: הווקטור $\vec{y} - \mathbf{X}\vec{w}$ חייב להיות אורתוגונלי לכל עמודה של $\mathbf{X}$:

$$\mathbf{X}^T(\vec{y} - \mathbf{X}\vec{w}) = \vec{0} \tag{73}$$

פתיחת הסוגריים:

$$\mathbf{X}^T\vec{y} - \mathbf{X}^T\mathbf{X}\vec{w} = \vec{0} \tag{74}$$

זו בדיוק ה-noitauqE lamroN! לכן:

$$\vec{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\vec{y} \quad \blacksquare \tag{75}$$

המשמעות העמוקה:

כאשר אנחנו פותרים רגרסיה ליניארית, אנחנו למעשה מבצעים הטלה גיאומטרית. השארית $\vec{y} - \hat{\vec{y}}$ היא **הרכיב האנכי** - החלק שלא ניתן להסביר על ידי המודל הליניארי.

זו הסיבה שבגרף residual plot (שאריות מול חיזוי), אנחנו מצפים לראות רעש אקראי - אם יש תבנית, זה אומר שהשארנו מידע שהמודל לא לכד.

מטריצת ההטלה:

ניתן לכתוב את ההטלה כמטריצה:

$$\mathbf{P} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T \tag{76}$$

אזי $\hat{\vec{y}} = \mathbf{P}\vec{y}$.

תכונות של מטריצת הטלה:

1. **סימטרית:** $\mathbf{P}^T = \mathbf{P}$
2. **אידמפוטנטית:** $\mathbf{P}^2 = \mathbf{P}$ (הטלה של הטלה היא אותה הטלה)

3. **דרגה:** $\text{rank}(\mathbf{P}) = d + 1$ (ממד תת-המרחב)

5.12 תרגיל תכנות עצמי 5.1 - השוואת שיטות

מטרה: להבין את היתרונות והחסרונות של כל שיטה.

משימה:

1. צרו נתונים סינתטיים עם d תכונות ורעש
2. פתרו באמצעות:
   - Normal Equation
   - Gradient Descent
   - Stochastic GD
   - Mini-Batch GD
   - Adam
3. השוו: זמן ריצה, דיוק, יציבות
4. נסו עם d גדל: 10, 100, 1000

השוואת שיטות אופטימיזציה

```python
import numpy as np

# Method 1: Normal Equation - O(d^3)
X_with_intercept = np.column_stack([np.ones(len(X)), X])
w_normal = np.linalg.solve(X_with_intercept.T @ X_with_intercept,
                           X_with_intercept.T @ y)

# Method 2: Gradient Descent - O(nd) per iteration
w = np.random.randn(d + 1) * 0.01
alpha = 0.01
for _ in range(1000):
    gradient = -2/n * X_with_intercept.T @ (y - X_with_intercept @ w)
    w = w - alpha * gradient

# Compare: when d << n use Normal Eq, when d >> n use GD
```

תוצאה צפויה:

טבלה 19: השוואת ביצועים (אומדן)

שיטה	$d = 10$	$d = 100$	$d = 500$
Normal Eq	0.001s	0.01s	0.5s
GD (1000 iter)	0.02s	0.2s	1.0s
sklearn	0.001s	0.008s	0.3s

מסקנות:

- עבור d קטן: Normal Equation הכי מהיר ומדויק
- עבור d גדול: Gradient Descent יעיל יותר
- sklearn משלב את שתי הגישות והוא האופטימלי

5.13 אתגרים ומגבלות: מתי רגרסיה נכשלת

רגרסיה ליניארית היא כלי עוצמתי, אבל היא לא פלא. יש מצבים שבהם היא נכשלת כישלון חרוץ.

בעיה 1 - קשרים לא-ליניאריים:

אם הקשר האמיתי הוא $y = x^2$, רגרסיה ליניארית תיכשל. הקו הישר לא יכול לתפוס עקומה פרבולית.

פתרון: הוסף תכונות פולינומיות - $x, x^2, x^3, \ldots$ - או עבור למודלים לא-ליניאריים (רשתות נוירונים).

בעיה 2 - Multicollinearity:

אם שתי תכונות קשורות מאוד ($r > 0.95$), המטריצה $\mathbf{X}^T\mathbf{X}$ כמעט סינגולרית. ההיפוך יוצר שגיאות נומריות עצומות, והמשקלים הופכים לא יציבים.

דוגמה: אם יש תכונה "גובה בס"מ" ותכונה "גובה במטרים", הן קשורות לחלוטין ($\times 100$). המודל לא יכול להפריד ביניהן.

זיהוי: חשב VIF (Variance Inflation Factor):

$$\text{VIF}_j = \frac{1}{1 - R_j^2} \tag{77}$$

כאשר R_j^2 הוא ה-R^2 של רגרסיה של תכונה j על כל התכונות האחרות.

אם $\text{VIF} > 10$ ← בעיה חמורה.

פתרון: הסר תכונות מיותרות, או השתמש ב-noissergeR ossaL/egdiR.

בעיה 3 - Outliers:

רגרסיה ליניארית (עם MSE) רגישה מאוד לערכים חריגים. שגיאה של 100 נותנת $100^2 = 10000$ - עונש עצום שמשנה את כל הקו.

פתרון: השתמש ב-noissergeR tsuboR (למשל, Huber Loss) או זהה והסר outliers.

בעיה 4 - Overfitting (התאמת יתר):

כאשר $d \approx n$ או $d > n$, המודל יכול "לשנן" את נתוני האימון ולהיכשל בנתוני הבדיקה.

פתרון: רגולריזציה (Ridge/Lasso), הפחתת תכונות, או איסוף עוד נתונים.

5.14 אפילוג: מקו פשוט לרשתות עמוקות

הגענו למעגל שלם. התחלנו עם גאוס ב-1801, שניסה למצוא כוכב לכת אבוד. סיימנו עם Gradient Descent - הכלי המרכזי שמאמן את GPT-4.

מה הקשר?

רשת נוירונים היא רגרסיה ליניארית במסווה.

כל שכבה ברשת נוירונים היא פעולה ליניארית (matrix multiplication) ואחריה פונקציה לא-ליניארית (activation). אם נסיר את הפונקציות הלא-ליניאריות, נישאר עם... רגרסיה ליניארית מורכבת.

המבנה הבסיסי זהה:

- פונקציית הפסד (loss function)
- גרדיאנט (כיוון הירידה)
- אופטימיזר (Adam, SGD)
- עדכון איטרטיבי של משקלים

ההבדל היחיד: במקום קו ישר פשוט, יש לנו מיליוני פרמטרים המתארים פונקציה מורכבת בצורה בלתי נתפסת.

אבל העקרונות? זהים לחלוטין.

כשגאוס חיפש את הכוכב האבוד, הוא לא ידע שהוא מניח את היסוד למהפכה שתגיע 220 שנה אחרי מותו. שהשיטה שלו תאמן מודלים שמנבאים סרטן, מתרגמים שפות, ויוצרים אמנות.

זו העוצמה של רעיון טוב - הוא חי הרבה מעבר למי שהמציא אותו.

5.15 סיכום: מה למדנו

רגרסיה ליניארית אינה רק טכניקה סטטיסטית. היא דרך לחשוב על העולם - כיצד למצוא סדר בכאוס, כיצד לחזות את הבלתי צפוי, כיצד ללמוד מנתונים.

העקרונות המרכזיים:

1. **מזעור שגיאה בריבוע** - לא רק נוחות מתמטית, אלא הצדקה סטטיסטית עמוקה
2. **פתרון אנליטי** - Normal Equation יפה, אלגנטית, אבל לא תמיד מעשית

3. **פתרון איטרטיבי** - Gradient Descent הוא הבסיס של כל למידה מודרנית

4. **גיאומטריה** - הטלה אורתוגונלית מקשרת אופטימיזציה לאלגברה ליניארית

5. **הכללה** - מקו פשוט ל-gninraeL peeD, העקרונות נשמרים

הכלים שרכשנו:

- גזירה חלקית וגרדיאנט - המצפן המתמטי
- Normal Equation - פתרון מדויק במכה אחת
- Gradient Descent - איטרציה שמתאימה לקנה מידה
- אינטואיציה גיאומטרית - הטלה כהסבר ויזואלי

מבט קדימה - פרק 6:
בפרק הבא נחקור את **סיווג** - כיצד עוברים מחיזוי ערכים רציפים (רגרסיה) לחיזוי קטגוריות. נראה:

- **רגרסיה לוגיסטית** - רגרסיה שמסווגת
- **פונקציית Sigmoid** - כיצד ממפים מספרים להסתברויות
- **Cross-Entropy Loss** - למה לא MSE לסיווג?
- **גבול ההחלטה** - הקו שמפריד בין מחלקות
- **הכללה למרובה מחלקות** - Softmax ו-llA-sv-enO

מטלות וקריאה מורחבת

תרגיל 5.1: השווה שיטות אופטימיזציה (ראו פסאודו-קוד).

תרגיל 5.2: הוכח שמטריצת ההטלה $\mathbf{P} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ היא אידמפוטנטית.
רמז: חשב $\mathbf{P}^2$ והראה שהיא שווה ל-$\mathbf{P}$.

תרגיל 5.3: מצא את קצב הלמידה האופטימלי עבור Gradient Descent על נתונים סינתטיים.

קריאה מורחבת:

- [30] – "Nouvelles méthodes pour la détermination des orbites des comètes": לז'נדר על ריבועים פחותים
- [31] – "Theoria Motus Corporum Coelestium": גאוס והצדקה סטטיסטית
- [32] – "An Overview of Gradient Descent Optimization Algorithms": סקירה מקיפה של אופטימיזרים

- [9] – Deep Learning, פרק 8: אופטימיזציה ב-gninraeL peeD

שאלות להעמקה:

1. מדוע Adam מתכנס מהר יותר מ-DG allinaV?
2. מה יקרה ל-noitauqE lamroN אם $\mathbf{X}^T\mathbf{X}$ סינגולרית?
3. האם Gradient Descent מובטח להתכנס למינימום גלובלי?

סיום פרק 5

6 רגרסיה לוגיסטית: מחיזוי מספרים למחלקות

Logistic Regression: From Predicting Numbers to Predicting Classes

ד"ר יורם סגל

ספטמבר 5202 - שיעור 80

6.1 פרולוג: החלטה בינארית שהצילה מיליונים

בשנת 1854, בעיירה Soho שבלונדון, פרצה מגיפת כולרה שהרגה מאות תושבים תוך ימים ספורים. הרופאים לא הבינו איך המחלה מתפשטת - התאוריה המקובלת טענה ש"אוויר רע" גורם למגיפה. אך הרופא ג'ון סנואו (John Snow) חשד שהסיבה אחרת.

סנואו עשה משהו מהפכני: הוא מיפה כל מקרה של כולרה על מפת הרובע, וחיפש דפוסים. הוא גילה שכמעט כל החולים התגוררו קרוב לבאר מים מסוימת ברחוב Broad Street. המסקנה היתה חדה: המים מזוהמים, לא האוויר.

כשהוא הציג את הראיות לרשויות, הם הסירו את ידית המשאבה מהבאר. המגיפה נעצרה כמעט מיד. סנואו הוכיח שהכולרה מתפשטת דרך מים מזוהמים - תובנה שהצילה מיליוני חיים במאה הבאה.

אבל מה הקשר לרגרסיה לוגיסטית?

סנואו ביצע, למעשה, **סיווג בינארי** (Binary Classification): האם אדם שגר במרחק x מטרים מהבאר יחלה בכולרה (כן/לא)? זו בדיוק השאלה שרגרסיה לוגיסטית עונה עליה. היא לא מנבאת מספר (כמו "כמה חולים יהיו"), אלא **הסתברות לקטגוריה** ("מה ההסתברות שאדם יחלה").

היום, רגרסיה לוגיסטית היא אחד הכלים הנפוצים ביותר ברפואה, פיננסים, ושיווק. האם מטופל יפתח סוכרת? האם לווה יחזיר הלוואה? האם לקוח ילחץ על מודעה? כל אלה הן שאלות של סיווג בינארי, והתשובה היא הסתברות שבין 0 ל-1.

6.2 מרגרסיה ליניארית לרגרסיה לוגיסטית: מדוע הקפיצה?

בפרק הקודם למדנו על רגרסיה ליניארית - מציאת הקו הטוב ביותר שמנבא ערך מספרי. אבל מה קורה כשאנחנו לא רוצים לנבא מספר, אלא קטגוריה?

נניח שאנחנו רוצים לנבא האם סטודנט יעבור מבחן (עבר/נכשל) על סמך שעות לימוד. אם ננסה להשתמש ברגרסיה ליניארית ישירות, $\hat{y} = w_0 + w_1 x$, נקבל תוצאות אבסורדיות:

טבלה 20: רגרסיה ליניארית לסיווג - הבעיה

שעות לימוד	חיזוי ליניארי	בעיה
1	-0.5	הסתברות שלילית?!
3	0.3	סביר
5	0.7	סביר
10	1.5	הסתברות מעל 1?!

הבעיה: רגרסיה ליניארית יכולה לתת כל ערך בין $-\infty$ ל- $+\infty$, אבל הסתברות חייבת להיות בין 0 ל-1.

הפתרון: פונקציית סיגמואיד.

6.3 פונקציית Sigmoid: הגשר בין קו להסתברות

דמיינו שיש לכם קו ישר שיכול להיות כל מספר, ואתם רוצים "לדחוס" אותו לטווח $[0, 1]$. איך עושים את זה?

התשובה: פונקציית **Sigmoid** (נקראת גם **פונקציה לוגיסטית**):

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{78}$$

תכונות מדהימות:

1. $\sigma(z) \in (0, 1)$ לכל $z \in \mathbb{R}$ - תמיד הסתברות חוקית
2. $\sigma(0) = 0.5$ - סימטרי סביב אפס
3. $\lim_{z\to\infty} \sigma(z) = 1$ - שואף ל-1 עבור ערכים גדולים
4. $\lim_{z\to-\infty} \sigma(z) = 0$ - שואף לאפס עבור ערכים קטנים
5. הנגזרת פשוטה: $\sigma'(z) = \sigma(z)(1 - \sigma(z))$ - יפהפייה לאופטימיזציה

פסאודו-קוד - ויזואליזציה של Sigmoid:

האינטואיציה - עקומת S:

הגרף של Sigmoid נראה כמו אות S מוחלקת. כשהקלט קטן מאוד (שלילי), הפלט כמעט אפס. כשהקלט גדול מאוד (חיובי), הפלט כמעט אחד. באמצע, סביב אפס, יש מעבר חלק.

זה בדיוק מה שאנחנו רוצים להסתברות: ככל שהראיות חזקות יותר לטובת מחלקה אחת, ההסתברות שואפת ל-1. ככל שהן חזקות לטובת המחלקה השנייה, ההסתברות שואפת לאפס.

פונקציית Sigmoid - ויזואליזציה

```python
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(z):
    """Sigmoid␣function"""
    return 1 / (1 + np.exp(-z))

# Plotting
z = np.linspace(-10, 10, 1000)
y = sigmoid(z)

plt.figure(figsize=(10, 6))
plt.plot(z, y, linewidth=2, label='σ(z)␣=␣1/(1+e^(-z))')
plt.axhline(y=0.5, color='red', linestyle='--', alpha=0.5, label='Decision␣Boundary')
plt.axvline(x=0, color='red', linestyle='--', alpha=0.5)
plt.xlabel('z')
plt.ylabel('σ(z)')
plt.title('Sigmoid␣Function')
plt.grid(True, alpha=0.3)
plt.legend()
plt.ylim(-0.1, 1.1)
plt.show()

# Special points
print(f"σ(-10)␣=␣{sigmoid(-10):.6f}␣␣≈␣0")
print(f"σ(-2)␣␣=␣{sigmoid(-2):.6f}")
print(f"σ(0)␣␣␣=␣{sigmoid(0):.6f}␣␣␣=␣0.5")
print(f"σ(2)␣␣␣=␣{sigmoid(2):.6f}")
print(f"σ(10)␣␣=␣{sigmoid(10):.6f}␣␣≈␣1")
```

6.4 המודל: שילוב ליניארי עם Sigmoid

עכשיו נחבר את שני העולמות: הליניארי והלוגיסטי.

צעד 1: חישוב ליניארי

כמו ברגרסיה ליניארית, נחשב שילוב ליניארי של התכונות:

$$z = w_0 + w_1x_1 + w_2x_2 + \cdots + w_dx_d = \vec{w}^T\vec{x} \tag{79}$$

z נקרא ה**logit** או ה**log-odds**.

צעד 2: החלת Sigmoid

נעביר את z דרך Sigmoid כדי לקבל הסתברות:

$$P(y=1|\vec{x}) = \sigma(z) = \frac{1}{1+e^{-\vec{w}^T\vec{x}}} \tag{80}$$

זו ההסתברות שהדגימה $\vec{x}$ שייכת למחלקה 1.

צעד 3: החלטה

כדי לקבל תחזית בינארית (כן/לא), נשתמש בסף (בדרך כלל 0.5):

$$\hat{y} = \begin{cases} 1 & \text{if } P(y=1|\vec{x}) \geq 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{81}$$

דוגמה מספרית:

נניח דומיל_תועש $\cdot$ $z = w_0 + w_1$, ומצאנו $w_0 = -5, w_1 = 1$.

טבלה 21: חיזוי עבירת מבחן

שעות	z	σ(z)	החלטה
1	-4	0.018	נכשל
3	-2	0.119	נכשל
5	0	0.500	על הגבול
7	2	0.881	עבר
10	5	0.993	עבר (בטוח)

שימו לב: ההסתברות תמיד חוקית (בין 0 ל-1), וככל ששעות הלימוד גדלות, ההסתברות להצלחה עולה בצורה חלקה.

6.5 פונקציית ההפסד: למה לא MSE?

בפרק הקודם השתמשנו בMSE (סכום ריבועי השגיאות). למה לא פשוט להמשיך עם זה?

בעיה 1 - לא קמורה:

עם Sigmoid, הפונקציה $\mathrm{MSE}(w) = \frac{1}{n}\sum(y_i - \sigma(\vec{w}^T\vec{x}_i))^2$ אינה קמורה (non-convex). יש לה מינימומים מקומיים רבים, וGradient Descent ייתקע בהם.

בעיה 2 - לא מתאימה להסתברויות:

MSE מעניש סטיות לינארית. אבל בהסתברויות, הפרש בין 0.01 ל-0.05 הוא הרבה יותר משמעותי מהפרש בין 0.51 ל-0.55.

הפתרון: Cross-Entropy Loss

פונקציית Cross-Entropy (נקראת גם Log-Loss) מתוכננת במיוחד להסתברויות:

$$\mathcal{L}(\vec{w}) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{p}_i) + (1-y_i)\log(1-\hat{p}_i)] \tag{82}$$

כאשר $\hat{p}_i = \sigma(\vec{w}^T\vec{x}_i)$ היא ההסתברות החזויה ש-$y_i = 1$.

הבנת הנוסחה:

נפרק למקרים:

אם $y_i = 1$ (המחלקה האמיתית היא *1*):

$$\mathcal{L}_i = -\log(\hat{p}_i) \tag{83}$$

אם $\hat{p}_i = 1$ (בטוח לחלוטין, וצדק) $\leftarrow$ $\mathcal{L}_i = 0$ (אין עונש)

אם $\hat{p}_i = 0.5$ (לא בטוח) $\leftarrow$ $\mathcal{L}_i = 0.69$

אם $\hat{p}_i = 0.01$ (טעה לגמרי) $\leftarrow$ $\mathcal{L}_i = 4.6$ (עונש כבד!)

אם $y_i = 0$ (המחלקה האמיתית היא *0*):

$$\mathcal{L}_i = -\log(1-\hat{p}_i) \tag{84}$$

אם $\hat{p}_i = 0$ (בטוח לחלוטין, וצדק) $\leftarrow$ $\mathcal{L}_i = 0$

אם $\hat{p}_i = 0.99$ (טעה לגמרי) $\leftarrow$ $\mathcal{L}_i = 4.6$

הקשר לMaximum Likelihood:

בדיוק כמו שMSE נובע מהנחה על התפלגות נורמלית של שגיאות, Cross-Entropy נובע מהנחה על **התפלגות ברנולי** (Bernoulli Distribution) - ההתפלגות של אירועים בינאריים.

מקסום הLikelihood שקול למזעור Cross-Entropy. זו ההצדקה התאורטית העמוקה.

פסאודו-קוד - השוואת MSE וCross-Entropy:

תוצאה: Cross-Entropy מעניש שגיאות בצורה לוגריתמית - טעויות קטנות מקבלות עונש קטן, אבל ביטחון גבוה בטעות מקבל עונש עצום.

השוואת פונקציות הפסד

```python
import numpy as np
import matplotlib.pyplot as plt

def mse_loss(y_true, y_pred):
    """MSE␣-␣not␣suitable␣for␣classification"""
    return (y_true - y_pred)**2

def cross_entropy_loss(y_true, y_pred):
    """Cross-Entropy␣-␣suitable␣for␣classification"""
    eps = 1e-15  # prevent log(0)
    y_pred = np.clip(y_pred, eps, 1 - eps)
    if y_true == 1:
        return -np.log(y_pred)
    else:
        return -np.log(1 - y_pred)

# Plotting
y_pred = np.linspace(0.01, 0.99, 1000)

plt.figure(figsize=(14, 6))

# MSE
plt.subplot(1, 2, 1)
plt.plot(y_pred, [mse_loss(1, p) for p in y_pred], label='y_true=1')
plt.plot(y_pred, [mse_loss(0, p) for p in y_pred], label='y_true=0')
plt.xlabel('Predicted␣Probability')
plt.ylabel('Loss')
plt.title('MSE␣Loss␣(Not␣Suitable)')
plt.legend()
plt.grid(True, alpha=0.3)

# Cross-Entropy
plt.subplot(1, 2, 2)
plt.plot(y_pred, [cross_entropy_loss(1, p) for p in y_pred], label='
y_true=1')
plt.plot(y_pred, [cross_entropy_loss(0, p) for p in y_pred], label='
y_true=0')
plt.xlabel('Predicted␣Probability')
plt.ylabel('Loss')
plt.title('Cross-Entropy␣Loss␣(Correct)')
plt.legend()
plt.grid(True, alpha=0.3)
plt.yscale('log')

plt.tight_layout()
plt.show()
```

6.6 הגרדיאנט: אופטימיזציה של רגרסיה לוגיסטית

כדי למצוא את המשקלים $\vec{w}^*$ שממזערים את Cross-Entropy, נצטרך לחשב את הגרדיאנט.

נגזרת Sigmoid - הנוסחה היפה:

$$\begin{aligned}\frac{d\sigma(z)}{dz} &= \frac{d}{dz}\left[\frac{1}{1+e^{-z}}\right] \\ &= \frac{e^{-z}}{(1+e^{-z})^2} \\ &= \frac{1}{1+e^{-z}} \cdot \frac{e^{-z}}{1+e^{-z}} \\ &= \sigma(z) \cdot (1-\sigma(z))\end{aligned} \tag{85}$$

זו נוסחה יפהפייה - הנגזרת מתבטאת בSigmoid עצמו!

גרדיאנט Cross-Entropy:

לאחר חישובים (שאציג בפירוט), מתקבלת תוצאה אלגנטית להפתיע:

$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{1}{n}\sum_{i=1}^{n}(\hat{p}_i - y_i)x_{ij} \tag{86}$$

או בצורה וקטורית:

$$\nabla_{\vec{w}}\mathcal{L} = \frac{1}{n}\mathbf{X}^T(\vec{\hat{p}} - \vec{y}) \tag{87}$$

הפתעה מדהימה: הנוסחה כמעט זהה לרגרסיה ליניארית! ההבדל היחיד: במקום $\vec{\hat{y}} = \mathbf{X}\vec{w}$ יש לנו $\vec{\hat{p}} = \sigma(\mathbf{X}\vec{w})$.

הוכחה מפורטת:

נתחיל מפונקציית ההפסד לדגימה אחת:

$$\mathcal{L}_i = -[y_i \log(\hat{p}_i) + (1-y_i)\log(1-\hat{p}_i)] \tag{88}$$

כאשר $\hat{p}_i = \sigma(z_i) = \sigma(\vec{w}^T\vec{x}_i)$.

נגזור לפי w_j (כלל השרשרת):

$$\frac{\partial \mathcal{L}_i}{\partial w_j} = \frac{\partial \mathcal{L}_i}{\partial \hat{p}_i} \cdot \frac{\partial \hat{p}_i}{\partial z_i} \cdot \frac{\partial z_i}{\partial w_j} \tag{89}$$

צעד 1: נגזרת לפי $\hat{p}_i$:

$$\begin{aligned}\frac{\partial \mathcal{L}_i}{\partial \hat{p}_i} &= -\left[\frac{y_i}{\hat{p}_i} - \frac{1-y_i}{1-\hat{p}_i}\right] \\ &= \frac{-(y_i(1-\hat{p}_i) - (1-y_i)\hat{p}_i)}{\hat{p}_i(1-\hat{p}_i)} \\ &= \frac{\hat{p}_i - y_i}{\hat{p}_i(1-\hat{p}_i)} \end{aligned} \tag{90}$$

צעד 2: נגזרת Sigmoid:

$$\frac{\partial \hat{p}_i}{\partial z_i} = \hat{p}_i(1-\hat{p}_i) \tag{91}$$

צעד 3: נגזרת השילוב הליניארי:

$$\frac{\partial z_i}{\partial w_j} = x_{ij} \tag{92}$$

צעד 4: שילוב הכל:

$$\begin{aligned}\frac{\partial \mathcal{L}_i}{\partial w_j} &= \frac{\hat{p}_i - y_i}{\hat{p}_i(1-\hat{p}_i)} \cdot \hat{p}_i(1-\hat{p}_i) \cdot x_{ij} \\ &= (\hat{p}_i - y_i)x_{ij} \end{aligned} \tag{93}$$

הגורמים $\hat{p}_i(1-\hat{p}_i)$ מתבטלים! זו הסיבה שהנוסחה פשוטה כל כך.

עבור כל הנתונים:

$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{1}{n}\sum_{i=1}^{n}(\hat{p}_i - y_i)x_{ij} \quad \blacksquare \tag{94}$$

אלגוריתם Gradient Descent לרגרסיה לוגיסטית:

6.7 גבול ההחלטה: הקו שמפריד בין מחלקות

אחד המושגים המרכזיים בסיווג הוא **גבול ההחלטה** (Decision Boundary) - הקו (או המשטח) שמפריד בין המחלקות.

אינטואיציה:

גבול ההחלטה הוא המקום שבו ההסתברות היא בדיוק 0.5 - כלומר, המודל לא בטוח לאיזה צד להכריע.

מתמטית:

רגרסיה לוגיסטית - מימוש מלא

```python
import numpy as np

def sigmoid(z):
    """Sigmoid␣function"""
    return 1 / (1 + np.exp(-np.clip(z, -500, 500)))  # clip to prevent overflow

def cross_entropy_loss(y_true, y_pred):
    """Cross-Entropy␣Loss"""
    eps = 1e-15
    y_pred = np.clip(y_pred, eps, 1 - eps)
    return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))

def logistic_regression_gd(X, y, alpha=0.01, max_iters=1000, tol=1e-6):
    """
␣␣␣␣Logistic␣regression␣using␣Gradient␣Descent.

␣␣␣␣Args:
␣␣␣␣␣␣␣␣X:␣feature␣matrix␣(n,␣d)
␣␣␣␣␣␣␣␣y:␣binary␣labels␣vector␣(n,)
␣␣␣␣␣␣␣␣alpha:␣learning␣rate
␣␣␣␣␣␣␣␣max_iters:␣maximum␣number␣of␣iterations
␣␣␣␣␣␣␣␣tol:␣convergence␣threshold

␣␣␣␣Returns:
␣␣␣␣␣␣␣␣w:␣final␣weights
␣␣␣␣␣␣␣␣history:␣loss␣history
␣␣␣␣"""
    # Add intercept
    X_with_intercept = np.column_stack([np.ones(len(X)), X])
    n, d = X_with_intercept.shape

    # Initialization
    w = np.zeros(d)
    history = []

    for iteration in range(max_iters):
        # Forward pass
        z = X_with_intercept @ w
        p_pred = sigmoid(z)

        # Loss
        loss = cross_entropy_loss(y, p_pred)
        history.append(loss)

        # Gradient
        gradient = (1/n) * X_with_intercept.T @ (p_pred - y)

        # Update
```

$$
\begin{aligned}
P(y=1|\vec{x}) &= 0.5 \\
\sigma(\vec{w}^T\vec{x}) &= 0.5 \\
\frac{1}{1+e^{-\vec{w}^T\vec{x}}} &= 0.5 \\
e^{-\vec{w}^T\vec{x}} &= 1 \\
\vec{w}^T\vec{x} &= 0
\end{aligned} \tag{95}
$$

המסקנה: גבול ההחלטה הוא **היפר-מישור ליניארי** $\vec{w}^T\vec{x} = 0$.

במימד דו-ממדי ($d = 1$, תכונה אחת):

$$w_0 + w_1 x = 0 \quad \Rightarrow \quad x = -\frac{w_0}{w_1} \tag{96}$$

זו נקודה בודדת על ציר ה-x.

במימד תלת-ממדי ($d = 2$, שתי תכונות):

$$w_0 + w_1 x_1 + w_2 x_2 = 0 \tag{97}$$

זהו קו ישר במישור.

במימד גבוה יותר:

היפר-מישור שמפריד את המרחב לשני חצאים.

משמעות גיאומטרית:

הווקטור $\vec{w}$ הוא **נורמל** (אנכי) לגבול ההחלטה. הוא מצביע לכיוון המחלקה החיובית ($y = 1$). ככל שנקודה רחוקה יותר מגבול ההחלטה בכיוון $\vec{w}$, ההסתברות ש-$y = 1$ גבוהה יותר.

6.8 מדדי הערכה: דיוק זה לא הכל

אחרי שאימנו מודל, איך נדע עד כמה הוא טוב?

התשובה הראשונה שעולה בראש היא **דיוק** (Accuracy) - אחוז החיזויים הנכונים. אבל זה לא מספיק.

הבעיה של דיוק - דוגמה:

נניח שאנחנו מנבאים מחלה נדירה שמופיעה רק ב-1% מהאוכלוסייה. אם המודל שלנו פשוט יחזה "לא חולה" לכולם, הוא יהיה **נכון ב-99% מהמקרים!**

אבל המודל הזה חסר תועלת - הוא לא יזהה אף חולה אחד.

מטריצת הבלבול (Confusion Matrix):

כדי להבין באמת איך המודל עובד, צריך לראות את ההתפלגות המלאה של החיזויים:

טבלה 22: מטריצת בלבול

	חזוי: 0	חזוי: 1
אמיתי: 1	False Negative (FN)	True Positive (TP)
אמיתי: 0	True Negative (TN)	False Positive (FP)

מדדים נגזרים:

דיוק (Accuracy):

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{98}$$

דיוק חיובי (Precision):

$$\text{Precision} = \frac{TP}{TP + FP} \tag{99}$$

"מכל מי שחזינו כחיובי, כמה באמת חיוביים?"

רגישות (Recall / Sensitivity):

$$\text{Recall} = \frac{TP}{TP + FN} \tag{100}$$

"מכל החיוביים האמיתיים, כמה זיהינו?"

F1-Score - ממוצע הרמוני:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{101}$$

דוגמה - אבחון סרטן:

נניח שיש 100 מטופלים: 5 חולים, 95 בריאים.
המודל חיזה: 4 כחולים (מתוכם 3 נכונים), 96 כבריאים.

טבלה 23: דוגמה: אבחון סרטן

	חזוי: בריא	חזוי: חולה
אמיתי: חולה	2	3
אמיתי: בריא	94	1

חישוב:

- Accuracy = $(94 + 3)/100 = 0.97$ - נראה מצוין!
- Precision = $3/(3 + 1) = 0.75$ - מכל החיזויים החיוביים, 75% נכונים

- Recall = $3/(3+2) = 0.60$ - זיהינו רק 60% מהחולים!
- $F_1 = 2 \cdot (0.75 \cdot 0.60)/(0.75 + 0.60) = 0.67$

המסקנה: למרות דיוק של 97%, המודל החמיץ 2 מתוך 5 החולים - בעיה חמורה ברפואה!
איזון הסף:
ניתן לשלוט באיזון בין Precision וRecall על ידי שינוי סף ההחלטה (במקום 0.5, להשתמש ב0.3 או 0.7).
עקומת ROC (citsiretcarahC gnitarepO revieceR):
גרף המציג את הTrue Positive Rate (Recall) מול הFalse Positive Rate עבור כל הספים האפשריים.
AUC (Area Under Curve) - שטח מתחת לעקומת ROC:

- AUC = 1.0: מודל מושלם
- AUC = 0.5: מודל אקראי (חסר תועלת)
- AUC > 0.8: מודל טוב מאוד

חישוב מדדי הערכה ועקומת ROC

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

def evaluate_classification(y_true, y_pred_proba):
    fpr, tpr, _ = roc_curve(y_true, y_pred_proba)
    roc_auc = auc(fpr, tpr)

    # ROC Curve visualization
    plt.figure(figsize=(8, 4))
    plt.plot(fpr, tpr, label=f'AUC␣=␣{roc_auc:.3f}')
    plt.plot([0, 1], [0, 1], 'k--', label='Random')
    plt.xlabel('FPR')
    plt.ylabel('TPR')
    plt.legend()
    plt.show()

    return roc_auc

# Usage: auc_score = evaluate_classification(y_true, y_pred_proba)
```

6.9 הכללה למרובה מחלקות: Multiclass Classification

עד כה דיברנו על סיווג בינארי - שתי מחלקות בלבד. אבל מה קורה כשיש יותר משתיים?
דוגמאות:

- זיהוי ספרות כתובות ביד (0-9) - 10 מחלקות

- סיווג מינים של פרחים (סחלב, ורד, חמנית) - 3 מחלקות

- זיהוי רגשות (שמח, עצוב, כועס, ניטרלי) - 4 מחלקות

שתי גישות עיקריות:

גישה 1 - One-vs-Rest (OvR):

אמן K מסווגים בינאריים, כל אחד מפריד מחלקה אחת מכל השאר.
למשל, עבור 3 מחלקות:

- מסווג 1: "האם זה מחלקה A?" (כן/לא)

- מסווג 2: "האם זה מחלקה B?" (כן/לא)

- מסווג 3: "האם זה מחלקה C?" (כן/לא)

בשלב החיזוי, בחר את המחלקה עם ההסתברות הגבוהה ביותר.

גישה 2 - Softmax Regression (Multinomial Logistic Regression):

הכללה ישירה של רגרסיה לוגיסטית ל-K מחלקות.

פונקציית Softmax:

במקום Sigmoid, משתמשים בSoftmax שממפה וקטור של K מספרים להסתברויות:

$$\text{Softmax}(z_k) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}} \tag{102}$$

תכונות:

- $\sum_{k=1}^{K} \text{Softmax}(z_k) = 1$ - סכום ההסתברויות הוא 1

- $\text{Softmax}(z_k) \in (0,1)$ - כל ערך הוא הסתברות חוקית

- אם $z_k \gg z_j$ לכל $j \neq k$, אזי $\text{Softmax}(z_k) \approx 1$

המודל:

עבור כל מחלקה k, יש וקטור משקלים $\vec{w}_k$:

$$P(y = k|\vec{x}) = \frac{e^{\vec{w}_k^T \vec{x}}}{\sum_{j=1}^{K} e^{\vec{w}_j^T \vec{x}}} \tag{103}$$

פונקציית ההפסד - Categorical Cross-Entropy:

$$\mathcal{L} = -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} y_{ik} \log(\hat{p}_{ik}) \tag{104}$$

כאשר $y_{ik} = 1$ אם הדגימה i שייכת למחלקה k, ואחרת $y_{ik} = 0$ (קידוד one-hot).

קשר לSigmoid:

במקרה של שתי מחלקות ($K = 2$), Softmax מצטמצם לSigmoid! זו הצדקה יפה לכך שהכללנו נכון.

דוגמה: Softmax Regression

```
import numpy as np

def softmax(z):
    exp_z = np.exp(z - np.max(z, axis=1, keepdims=True))
    return exp_z / np.sum(exp_z, axis=1, keepdims=True)

def softmax_regression(X, y, K, alpha=0.01, max_iters=1000):
    X_with_intercept = np.column_stack([np.ones(len(X)), X])
    n, d = X_with_intercept.shape
    Y_onehot = np.zeros((n, K))
    Y_onehot[np.arange(n), y] = 1
    W = np.random.randn(d, K) * 0.01

    for iteration in range(max_iters):
        logits = X_with_intercept @ W
        probs = softmax(logits)
        gradient = (1/n) * X_with_intercept.T @ (probs - Y_onehot)
        W = W - alpha * gradient

    return W

# Usage: W = softmax_regression(X_train, y_train, K=3, alpha=0.1)
```

6.10 תרגיל תכנות עצמי 6.1 - רגרסיה לוגיסטית מאפס

מטרה: להבין לעומק את המתמטיקה של רגרסיה לוגיסטית.

משימה:

1. צרו נתונים סינתטיים לסיווג בינארי (שתי מחלקות שניתן להפריד ליניארית)
2. מימשו רגרסיה לוגיסטית מאפס (ללא sklearn)
3. השוו לפתרון של sklearn
4. שנו את סף ההחלטה ובחנו את ההשפעה על Precision/Recall
5. הציגו את גבול ההחלטה גרפית

6.11 אפילוג: מסיווג פשוט לרשתות עמוקות

הגענו לסיום המסע ברגרסיה לוגיסטית. התחלנו עם ג'ון סנואו שחיפש את מקור מגיפת הכולרה, וסיימנו עם Softmax - הבסיס של רשתות נוירונים מודרניות.

הקשר עמוק יותר ממה שנראה.

רשת נוירונים פשוטה (Feedforward Neural Network) היא, למעשה, סדרה של רגרסיות לוגיסטיות מחוברות:

- כל נוירון בשכבה מבצע שילוב ליניארי של הקלטים
- לאחר מכן מופעלת פונקציה לא-ליניארית (activation function) - כמו Sigmoid, ReLU
- השכבה האחרונה משתמשת בSoftmax לסיווג

ההבדל היחיד: במקום להעביר את התכונות הגולמיות ישירות לSoftmax, הרשת הנוירונית **לומדת** איזה שילוב של תכונות הכי טוב - היא יוצרת "תכונות חדשות" בשכבות הביניים.

אבל המבנה הבסיסי? זהה לחלוטין. פונקציית הפסד - Cross-Entropy. האופטימיזציה - Gradient Descent (או גרסאות מתקדמות כמו Adam).

כשDavid Rumelhart, Geoffrey Hinton, וRonald Williams פרסמו את אלגוריתם ה-Back-propagation ב-1986 [33], הם לא המציאו משהו חדש לגמרי. הם הראו איך לחשב את הגרדיאנט ברשת עמוקה באמצעות כלל השרשרת - בדיוק כמו שעשינו ברגרסיה לוגיסטית, רק בסקלה גדולה יותר.

היום, מודלים כמו GPT, BERT, וVision Transformers משתמשים באותם עקרונות: פונקציית הפסד (בדרך כלל Cross-Entropy), גרדיאנט, ואופטימיזציה איטרטיבית. הסקלה השתנתה - מיליארדי פרמטרים במקום מאות - אבל היסוד נותר זהה.

6.12 סיכום: מה למדנו

רגרסיה לוגיסטית מלמדת אותנו שלפעמים הקפיצה הקטנה ביותר - מחיזוי מספרים לחיזוי הסתברויות - דורשת שינוי מהותי בגישה המתמטית.

העקרונות המרכזיים:

1. **פונקציית Sigmoid** - הגשר בין ליניארי ללוגיסטי
2. **Cross-Entropy Loss** - פונקציית ההפסד הנכונה להסתברויות
3. **גבול החלטה ליניארי** - למרות הלא-ליניאריות, הגבול עדיין ישר
4. **מדדי הערכה** - דיוק זה לא הכל, צריך Precision, Recall, F1
5. **הכללה למרובה מחלקות** - Softmax כהרחבה טבעית

הכלים שרכשנו:

- Sigmoid וSoftmax - מיפוי למרחב הסתברויות
- Cross-Entropy - מדידת מרחק בין התפלגויות
- מטריצת בלבול - הבנה מעמיקה של ביצועים
- עקומת ROC וAUC - הערכה גלובלית

מטלות וקריאה מורחבת

תרגיל 6.1: מימשו רגרסיה לוגיסטית מאפס (ראו פסאודו-קוד).

תרגיל 6.2: הוכיחו שבמקרה של שתי מחלקות, Softmax מצטמצם לSigmoid.

רמז: התחילו מנוסחת Softmax ל-$K = 2$ והראו ש-$P(y = 1) = \sigma(z_1 - z_0)$.

תרגיל 6.3: השוו Precision וRecall עבור ספי החלטה שונים (0.3, 0.5, 0.7).

קריאה מורחבת:

- [34] – "The Regression Analysis of Binary Sequences": xoC .R.D על רגרסיה לוגיסטית
- [33] – "Learning Representations by Back-propagating Errors": noitagaporpkcaB והקשר לרגרסיה
- [9] – Deep Learning, פרק 6: רשתות Feedforward
- [35] – Pattern Recognition and Machine Learning, פרק 4: מודלים ליניאריים לסיווג

שאלות להעמקה:

1. מדוע Cross-Entropy מתאים יותר מMSE לסיווג?
2. מה יקרה לגבול ההחלטה אם נכפיל את כל המשקלים ב-2?
3. האם רגרסיה לוגיסטית יכולה לפתור בעיית XOR?

סיום פרק 6

7 English References

1 G. Strang, *Linear Algebra and Learning from Data*. Wellesley, MA, USA: Wellesley-Cambridge Press, 2019.

2 T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2013.

3 Z. S. Harris, “Distributional structure,” *Word*, vol. 10, no. 2-3, 146–162, 1954.

4 A. Caliskan, J. J. Bryson, and A. Narayanan, “Semantics derived automatically from language corpora contain human-like biases,” 6334, 356, 2017, 183–186.

5 T. Bolukbasi, K.-W. Chang, J. Zou, V. Saligrama, and A. Kalai, “Man is to computer programmer as woman is to homemaker? debiasing word embeddings,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 29, 2016.

6 R. E. Bellman, *Dynamic Programming*. Princeton, NJ, USA: Princeton University Press, 1957.

7 Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” 11, 86, 1998, 2278–2324.

8 A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 25, 2012.

9 I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

10 K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, “When is ”nearest neighbor” meaningful?” *Lecture Notes in Computer Science*, vol. 1540, 217–235, 1999.

11 C. C. Aggarwal, A. Hinneburg, and D. A. Keim, “On the surprising behavior of distance metrics in high dimensional space,” *Lecture Notes in Computer Science*, vol. 1973, 420–434, 2001.

12 E. Fix and J. L. Hodges, “Discriminatory analysis: Nonparametric discrimination: Consistency properties,” *Technical Report, USAF School of Aviation Medicine*, 1951.

13 V. N. Vapnik and A. Y. Chervonenkis, “On the uniform convergence of relative frequencies of events to their probabilities,” *Theory of Probability and Its Applications*, vol. 16, no. 2, 264–280, 1971.

14 J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, 248–255.

15 Y. LeCun et al., “Backpropagation applied to handwritten zip code recognition,” *Neural Computation*, vol. 1, no. 4, 541–551, 1989.

16 K. Pearson, “On lines and planes of closest fit to systems of points in space,” *Philosophical Magazine*, vol. 2, no. 11, 559–572, 1901.

17 H. Hotelling, “Analysis of a complex of statistical variables into principal components,” *Journal of Educational Psychology*, vol. 24, no. 6, 417–441, 1933.

18 G. H. Golub and C. Reinsch, “Singular value decomposition and least squares solutions,” *Numerische Mathematik*, vol. 14, 403–420, 1970.

19 A. E. Hoerl and R. W. Kennard, “Ridge regression: Biased estimation for nonorthogonal problems,” *Technometrics*, vol. 12, no. 1, 55–67, 1970.

20 R. Tibshirani, “Regression shrinkage and selection via the lasso,” *Journal of the Royal Statistical Society: Series B*, vol. 58, no. 1, 267–288, 1996.

21 A. Chaman and I. Dokmanic, “Truly generative or just extrapolation? on the ability of generative models to create truly novel content,” *arXiv preprint arXiv:2109.09018*, 2021.

22 T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd. New York, NY, USA: Springer, 2009.

23 F. Galton, “Regression towards mediocrity in hereditary stature,” *Journal of the Anthropological Institute of Great Britain and Ireland*, vol. 15, 246–263, 1886.

24 K. Pearson, “Mathematical contributions to the theory of evolution. iii. regression, heredity, and panmixia,” *Philosophical Transactions of the Royal Society of London. Series A*, vol. 187, 253–318, 1896.

25 R. A. Fisher, *Statistical Methods for Research Workers*. Edinburgh, UK: Oliver and Boyd, 1925.

26 S. Wright, “Correlation and causation,” *Journal of Agricultural Research*, vol. 20, no. 7, 557–585, 1921.

27 A. B. Hill, “The environment and disease: Association or causation?” *Proceedings of the Royal Society of Medicine*, vol. 58, no. 5, 295–300, 1965.

28 J. Pearl, *Causality: Models, Reasoning, and Inference*, 2nd. Cambridge, UK: Cambridge University Press, 2009.

29 T. Vigen, *Spurious Correlations*. New York, NY, USA: Hachette Books, 2015.

30 A.-M. Legendre, *Nouvelles méthodes pour la détermination des orbites des comètes*. Paris, France: F. Didot, 1805.

31 C. F. Gauss, *Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Ambientium*. Hamburg, Germany: Friedrich Perthes and I. H. Besser, 1809.

32 S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv preprint arXiv:1609.04747*, 2016.

33 D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, 533–536, 1986.

34 D. R. Cox, “The regression analysis of binary sequences,” *Journal of the Royal Statistical Society: Series B*, vol. 20, no. 2, 215–232, 1958.

35 C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006.

