Contents

מערכות לומדות תרגיל 3

עמית בסקין 312259013

1. לכל $x \in \mathcal{X}$:

$$h_{\mathcal{D}}(\mathrm{x}) = \begin{cases} +1 & \Pr(y=1 \mid \mathrm{x}) \geq \frac{1}{2} \\ -1 & \text{else} \end{cases}$$

צ.להוכיח:

$$h_{\mathcal{D}}(\mathrm{x}) = \underset{y\in\{\pm1\}}{\operatorname{argmax}} \Pr(\mathrm{x} \mid y)\Pr(y)$$

הוכחה:

ראשית נשים לב כי:

$$\Pr(\mathrm{x} \mid y)\Pr(y) = \frac{\Pr(\mathrm{x} \wedge y)}{\Pr(y)} \cdot \Pr(y) = \Pr(\mathrm{x} \wedge y)$$

כלומר בעצם צריך להוכיח ש־

$$h_{\mathcal{D}}(\mathrm{x}) = \underset{y\in\{\pm1\}}{\operatorname{argmax}} \Pr(\mathrm{x} \wedge y)$$

יהא $\mathrm{x} \in \mathcal{X}$. נחלק למקרים:

- $\Pr(y=1 \mid \mathrm{x}) \geq \frac{1}{2}$, קרי:

$$\frac{\Pr(\mathrm{x} \wedge y=1)}{\Pr(\mathrm{x})} \geq \frac{1}{2}$$

$$\Pr(\mathrm{x} \wedge y=1) \geq \frac{1}{2}\Pr(\mathrm{x})$$

ובפרט:

$$\frac{\Pr(\mathrm{x} \wedge y=-1)}{\Pr(\mathrm{x})} \leq \frac{1}{2}$$

$$\Pr(\mathrm{x} \wedge y=-1) \leq \frac{1}{2}\Pr(\mathrm{x})$$

כלומר:

$$\Pr(\mathrm{x} \wedge y=1) \geq \Pr(\mathrm{x} \wedge y=-1)$$

ולכן:

$$\underset{y\in\{\pm1\}}{\operatorname{argmax}} \Pr\left(\mathrm{x}\wedge y\right)=1$$

ומאחר שכל המעברים הם אסם אזי ש־ $\Pr\left(y=1\mid \mathrm{x}\right)\geq\frac{1}{2}$ אסם $\underset{y\in\{\pm1\}}{\operatorname{argmax}} \Pr\left(\mathrm{x}\wedge y\right)=1$.

מש"ל מקרה ראשון.

- $\Pr\left(y=1\mid \mathrm{x}\right)\leq\frac{1}{2}$, קרי:

$$\frac{\Pr\left(\mathrm{x}\wedge y=1\right)}{\Pr\left(\mathrm{x}\right)}\leq\frac{1}{2}$$

$$\Pr\left(\mathrm{x}\wedge y=1\right)\leq\frac{1}{2}\Pr\left(\mathrm{x}\right)$$

ובפרט:

$$\frac{\Pr\left(\mathrm{x}\wedge y=-1\right)}{\Pr\left(\mathrm{x}\right)}\geq\frac{1}{2}$$

$$\Pr\left(\mathrm{x}\wedge y=-1\right)\geq\frac{1}{2}\Pr\left(\mathrm{x}\right)$$

כלומר:

$$\Pr\left(\mathrm{x}\wedge y=1\right)\leq\Pr\left(\mathrm{x}\wedge y=-1\right)$$

ולכן:

$$\underset{y\in\{\pm1\}}{\operatorname{argmax}} \Pr\left(\mathrm{x}\wedge y\right)=-1$$

ומאחר שכל המעברים הם אסם אזי ש־ $\Pr\left(y=1\mid \mathrm{x}\right)\leq\frac{1}{2}$ אסם $\underset{y\in\{\pm1\}}{\operatorname{argmax}} \Pr\left(\mathrm{x}\wedge y\right)=-1$.

מש"ל מקרה שני.

מש"ל

2. נניח ש־ $\mathcal{X}=\mathbb{R}^d$ וש־ $\mathrm{x}\mid y\sim\mathcal{N}\left(\mu_y,\Sigma\right)$ עבור $\mu_y\in\mathbb{R}^d$ ו־ $\Sigma\in\mathbb{R}^d$, קרי פונקציית הצפיפות היא:

$$f\left(\mathrm{x}\mid y\right)=\frac{1}{\sqrt{\left(2\pi\right)^d\det\left(\Sigma\right)}}\exp\left(-\frac{1}{2}\left(\mathrm{x}-\mu_y\right)^t\Sigma^{-1}\left(\mathrm{x}-\mu_y\right)\right)$$

צ.להראות ש־

$$h_{\mathcal{D}}\left(\mathrm{x}\right)=\operatorname{argmax}\left\{-\frac{1}{2}\mathrm{x}^t\Sigma^{-1}\mathrm{x}-\frac{1}{2}\mu_y^t\Sigma^{-1}\mu_y+\ln\left(\Pr\left(y\right)\right)\mid y\in\{\pm1\}\right\}$$

הוכחה:

לפי נוסחת בייס:

$$\Pr\left(y \mid \mathrm{x}\right) \cdot \Pr\left(\mathrm{x}\right) = f\left(\mathrm{x} \mid y\right) \cdot \Pr\left(y\right)$$

קרי:

$$\Pr\left(y \mid \mathrm{x}\right) = \frac{f\left(\mathrm{x} \mid y\right) \cdot \Pr\left(y\right)}{\Pr\left(\mathrm{x}\right)}$$

נסמן $c = \Pr\left(\mathrm{x}\right)$ כקבוע כי מספר זה לא מושפע מ־ y. אז:

$$\Pr\left(y \mid \mathrm{x}\right) = c \cdot f\left(\mathrm{x} \mid y\right) \cdot \Pr\left(y\right)$$

ונציב את $f\left(\mathrm{x} \mid y\right)$:

$$\Pr\left(y \mid \mathrm{x}\right) = \frac{c}{\sqrt{\left(2\pi\right)^d \det\left(\Sigma\right)}} \Pr\left(y\right) \exp\left(-\frac{1}{2}\left(\mathrm{x} - \mu_y\right)^t \Sigma^{-1}\left(\mathrm{x} - \mu_y\right)\right)$$

ונסמן $c' = \frac{c}{\sqrt{\left(2\pi\right)^d \det\left(\Sigma\right)}}$ כקבוע כי לא תלוי ב־ y:

$$\Pr\left(y \mid \mathrm{x}\right) = c' \Pr\left(y\right) \exp\left(-\frac{1}{2}\left(\mathrm{x} - \mu_y\right)^t \Sigma^{-1}\left(\mathrm{x} - \mu_y\right)\right)$$

$$\ln\left(\Pr\left(y \mid \mathrm{x}\right)\right) = \ln\left(c'\right) + \ln\left(\Pr\left(y\right)\right) - \frac{1}{2}\left(\mathrm{x} - \mu_y\right)^t \Sigma^{-1}\left(\mathrm{x} - \mu_y\right) =$$

$$= \ln\left(c'\right) + \ln\left(\Pr\left(y\right)\right) - \frac{1}{2}\mathrm{x}^t \Sigma^{-1}\mathrm{x} - \frac{1}{2}\mu_y^t \Sigma^{-1}\mu_y + x^t \Sigma^{-1}\mu_y$$

ונסמן:

$$c'' = \ln\left(c'\right) - \frac{1}{2}\mathrm{x}^t \Sigma^{-1}\mathrm{x}$$

כי לא תלויים ב־ y:

$$\Pr\left(y \mid \mathrm{x}\right) = x^t \Sigma^{-1}\mu_y - \frac{1}{2}\mu_y^t \Sigma^{-1}\mu_y + \ln\left(\Pr\left(y\right)\right) + c''$$

עתה, בסעיף הקודם ראינו ש־

$$h_{\mathcal{D}}\left(\mathrm{x}\right) = \underset{y \in \{\pm 1\}}{\operatorname{argmax}} \Pr\left(\mathrm{x} \mid y\right) \Pr\left(y\right)$$

אבל למקסם את $\Pr\left(\mathrm{x} \mid y\right) \Pr\left(y\right)$ זה כמו למקסם את $\ln\left(\Pr\left(\mathrm{x} \mid y\right) \Pr\left(y\right)\right)$ קרי:

$$\ln\left(\Pr\left(\mathrm{x} \mid y\right)\right) + \ln\left(\Pr\left(y\right)\right)$$

קרי את:

$$x^t \Sigma^{-1}\mu_y - \frac{1}{2}\mu_y^t \Sigma^{-1}\mu_y + 2\ln\left(\Pr\left(y\right)\right) + c''$$

ומאחר ש־ c'' לא תלוי ב־ y אזי שזה כמו למקסם את:

$$x^t \Sigma^{-1}\mu_y - \frac{1}{2}\mu_y^t \Sigma^{-1}\mu_y + 2\ln\left(\Pr\left(y\right)\right)$$

מש"ל

3. נתונה קבוצת דגימות $S=\{(\mathrm{x}_1,y_1),\dots,(\mathrm{x}_m,y_m)\}$. צ.להעריך את $\mu_{+1},\mu_{-1},\Sigma,\Pr(y)$ בהתבסס על S.

פיתרון:

נניח בה"כ שקיים $l\le m$ כך שלכל $i\in[l]$ מתקיים $y_i=1$.

נסמן:

$$S_{+1}=\{\mathrm{x}_i\mid i\in[l]\}$$

$$S_{-1}=\{\mathrm{x}_i\mid i\in[m]\setminus[l]\}$$

נסמן ב־ X_{+1} ו־ X_{-1} את המטריצות שהשורות שלהן הן הוקטורים ב־ S_{+1} ו־ S_{-1} בהתאמה.

נסמן ב־ μ_{+1} ו־ μ_{-1} את וקטורי הממוצעים של הוקטורים ב־ S_{+1} ו־ S_{-1} בהתאמה.

יהא μ וקטור הממוצעים של X, ונסמן ב־ $\widehat{X}$ את X ממורכזת לפי μ.

נסמן ב־ X^0_{+1} ו־ X^0_{-1} את מטריצות הוקטורים X_{+1} ו־ X_{-1} בהתאמה, ממורכזים לפי μ_{+1} ו־ μ_{-1}, בהתאמה.

נסמן ב־ Σ_{+1} ו־ Σ_{-1} את מטריצות השונות המשותפת של S_{+1} ו־ S_{-1} בהתאמה, קרי:

$$\Sigma_{+1}=\frac{1}{n_{+1}}\left(X^0_{+1}\right)^tX^0_{+1}$$

$$\Sigma_{-1}=\frac{1}{n_{-1}}\left(X^0_{-1}\right)^tX^0_{-1}$$

ואז מטריצת השונות המשותפת הכוללת מחושבת על ידי:

$$\Sigma_{ij}=\frac{1}{m}\left(n_{+1}\left(\Sigma_{+1}\right)_{ij}+n_{-1}\left(\Sigma_{-1}\right)_{ij}\right)$$

נסמן ב־ n_{+1} ו־ n_{-1} את $|S_{+1}|$ ו־ $|S_{-1}|$ בהתאמה.

נסמן $p_{+1}=\Pr(+1)$ ו־ $p_{-1}=\Pr(-1)$ ונעריך:

$$p_{+1}=\frac{n_{+1}}{m}$$

$$p_{-1}=\frac{n_{-1}}{m}$$

מש"ל

4. ישנן שתי טעויות אפשריות בסיווג ספאם:

- לסווג הודעה כספאם למרות שהיא לא.

- לסווג כלא ספאם הודעה שהיא כן ספאם.

הטעות שאנחנו הכי לא רוצים לעשות זה לסווג הודעה כספאם למרות שהיא לא. זו טעות חמורה יותר מאשר לסווג כלא ספאם הודעה שהיא כן ספאם. אכן, אם נסווג הודעה כלא ספאם למרות שהיא ספאם, אז המשתמש ייאלץ למרבה הצער להיתקל בספאם למרות שהיה מעדיף שלא. לעומת זאת, אם ישנה הודעה שהיא לא ספאם אבל סווגה כספאם, כנראה שהמשתמש יפספס את ההודעה, וזאת על אף שפוטנציאלית מדובר בהודעה חשובה שהמשתמש צריך לקרוא, והנזק שעלול להיגרם למשתמש כתוצאה מפספוס ההודעה, עשוי להיות גדול בהרבה מלקרוא הודעת ספאם שהיה מעדיף שלא לקרוא.

אם כן, אנחנו רוצים שהשגיאה החמורה תהיה שגיאה מסוג ראשון, קרי FP, ולכן ניתן לסיווג הודעה כספאם את הלייבל positive, כלומר אם נסווג הודעה שהיא לא ספאם ב־ positive, אבל טעינו False, אז קיבלנו שגיאה מסוג ראשון, כפי שרצינו.

5. הצורה הקאנונית של תכנית ריבועית היא:

$$\underset{\mathrm{v}\in\mathbb{R}^n}{\operatorname{argmin}}\ \left(\frac{1}{2}\mathrm{v}^tQ\mathrm{v}+\mathrm{a}^t\mathrm{v}\right)$$

כך ש־

$$A\mathrm{v}\leq \mathrm{d}$$

עבור:

$$Q\in\mathbb{R}^{n\times n}, A\in\mathbb{R}^{m\times n}, \mathrm{a}\in\mathbb{R}^n, \mathrm{d}\in\mathbb{R}^m$$

נתבונן בבעיית ה־ Hard-SVM:

$$\underset{(\mathrm{w},b)}{\operatorname{argmin}}\ \|\mathrm{w}\|^2$$

כך שלכל i:

$$y_i\left(\langle\mathrm{w},\mathrm{x}_i\rangle+b\right)\geq 1$$

צ.לכתוב אותה כתכנית ריבועית קאנונית.

פיתרון:

נניח שיש לנו m דגימות (x_i, y_i) כאשר כל x_i הוא מסדר $n-1$.

נסמן ב־ A' מטריצה $m\times n-1$ שהשורות שלה הם הוקטורים $y_i\mathrm{x}_i$.

נסמן ב־ A'' את המטריצה שמתקבלת מ־ A' על ידי הוספת עמודה שהכניסות בה הן y_i, וניקח $A=-A''$.

נסמן $Q'=I_{(n-1)\times(n-1)}$, ונתאר את Q כמטריצת בלוקים שנתונים באלכסון הראשי: הבלוק הראשון הוא Q' והבלוק השני הוא הסקלר 0.

את a ניקח להיות 0_n ואת d ניקח להיות $(-1)_m$

אז התכנית הריבועית שלנו היא:

$$\underset{\mathrm{v}\in\mathbb{R}^n}{\operatorname{argmin}}\ \left(\frac{1}{2}\mathrm{v}^tQ\mathrm{v}+\mathrm{a}^t\mathrm{v}\right)$$

קרי:

$$Q=2\begin{pmatrix}1&&&\\&\ddots&&\\&&1&\\&&&0\end{pmatrix},\mathrm{v}=\mathrm{w}=\begin{pmatrix}w_1\\\vdots\\w_n\\b\end{pmatrix},\mathrm{a}=\begin{pmatrix}0\\\vdots\\0\end{pmatrix},A=-\begin{pmatrix}y_1&y_1x_{11}&\cdots&y_1x_{1(n-1)}\\\vdots&\vdots&\ddots&\vdots\\y_m&y_mx_{m1}&\cdots&y_mx_{m(n-1)}\end{pmatrix},\mathrm{d}=-\begin{pmatrix}1\\\vdots\\1\end{pmatrix}$$

כלומר:

$$\frac{1}{2}\mathrm{v}^tQ\mathrm{v}+\mathrm{a}^t\mathrm{v}=$$

$$=\frac{1}{2}\begin{pmatrix}w_1&\cdots&w_n\end{pmatrix}2\begin{pmatrix}1&&&\\&\ddots&&\\&&1&\\&&&0\end{pmatrix}\begin{pmatrix}b\\w_1\\\vdots\\w_n\end{pmatrix}+\begin{pmatrix}0&\cdots&0\end{pmatrix}\begin{pmatrix}w_1\\\vdots\\w_n\\b\end{pmatrix}=$$

$$=\begin{pmatrix}w_1&\cdots&w_n\end{pmatrix}\begin{pmatrix}w_1\\\vdots\\w_n\end{pmatrix}=\|\mathrm{w}\|^2$$

כאשר:

$$A\mathrm{v}\leq\mathrm{d}$$

קרי:

$$-\begin{pmatrix}y_1&y_1x_{11}&\cdots&y_1x_{1(n-1)}\\\vdots&\vdots&\ddots&\vdots\\y_m&y_mx_{m1}&\cdots&y_mx_{m(n-1)}\end{pmatrix}\begin{pmatrix}b\\w_1\\\vdots\\w_n\end{pmatrix}\leq-\begin{pmatrix}1\\\vdots\\1\end{pmatrix}$$

$$\begin{pmatrix}\langle y_1\mathrm{x}_1,\mathrm{w}\rangle+y_1b\\\vdots\\\langle y_m\mathrm{x}_m,\mathrm{w}\rangle+y_mb\end{pmatrix}\geq\begin{pmatrix}1\\\vdots\\1\end{pmatrix}$$

$$\begin{pmatrix}y_1\langle\mathrm{w},\mathrm{x}_1\rangle+y_1b\\\vdots\\y_m\langle\mathrm{w},\mathrm{x}_m\rangle+y_mb\end{pmatrix}\geq\begin{pmatrix}1\\\vdots\\1\end{pmatrix}$$

$$\begin{pmatrix}y_1\left(\langle\mathrm{w},\mathrm{x}_1\rangle+b\right)\\\vdots\\y_m\left(\langle\mathrm{w},\mathrm{x}_m\rangle+b\right)\end{pmatrix}\geq\begin{pmatrix}1\\\vdots\\1\end{pmatrix}$$

וקיבלנו שלכל i צריך להתקיים:

$$y_i\left(\langle\mathrm{w},\mathrm{x}_i\rangle+b\right)\geq1$$

כפי שרצינו.

מש"ל.

6. נתבונן בבעיית ה־ Soft-SVM:

$$\underset{\mathrm{w},\{\xi_i\}}{\operatorname{argmin}}\left(\frac{1}{2}\lambda\left\|\mathrm{w}\right\|^2+\frac{1}{m}\sum_{i=1}^{m}\xi_i\right)$$

כך שלכל i:

$$y_i\left\langle \mathrm{w},\mathrm{x}_i\right\rangle \geq 1-\xi_i$$

$$\xi_i\geq 0$$

צ.להוכיח שהיא שקולה לבעייה:

$$\underset{\mathrm{w}}{\operatorname{argmin}}\left(\frac{1}{2}\lambda\left\|\mathrm{w}\right\|^2+\frac{1}{m}\sum_{i=1}^{m}l^{hinge}\left(y_i\left\langle \mathrm{w},\mathrm{x}_i\right\rangle\right)\right)$$

כאשר:

$$l^{hinge}\left(y_i\left\langle \mathrm{w},\mathrm{x}_i\right\rangle\right)=\max\left\{0,1-y_i\left\langle \mathrm{w},\mathrm{x}_i\right\rangle\right\}$$

הוכחה:

מתקיים:

$$\sum_{i=1}^{m}l^{hinge}\left(y_i\left\langle \mathrm{w},\mathrm{x}_i\right\rangle\right)=\sum_{i,y_i\left\langle \mathrm{w},\mathrm{x}_i\right\rangle<1}\left(1-y_i\left\langle \mathrm{w},\mathrm{x}_i\right\rangle\right)$$

כלומר הבעיה הנ"ל שקולה ל־

$$\underset{\mathrm{w}}{\operatorname{argmin}}\left(\frac{1}{2}\lambda\left\|\mathrm{w}\right\|^2+\frac{1}{m}\sum_{i,y_i\left\langle \mathrm{w},\mathrm{x}_i\right\rangle<1}\left(1-y_i\left\langle \mathrm{w},\mathrm{x}_i\right\rangle\right)\right)$$

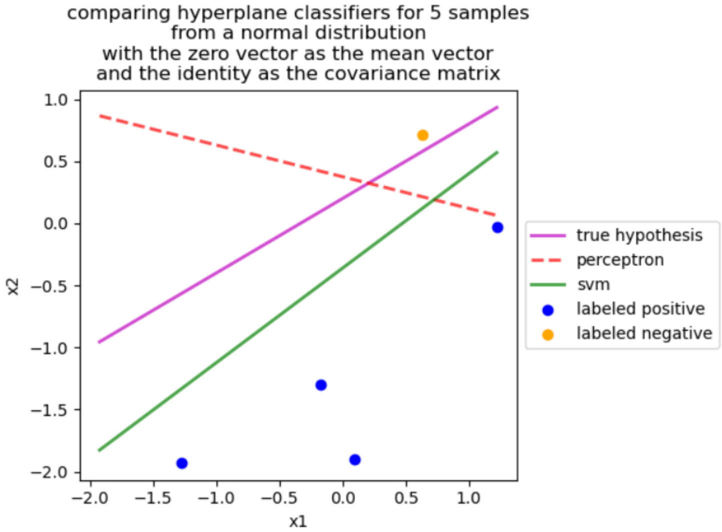
עתה:

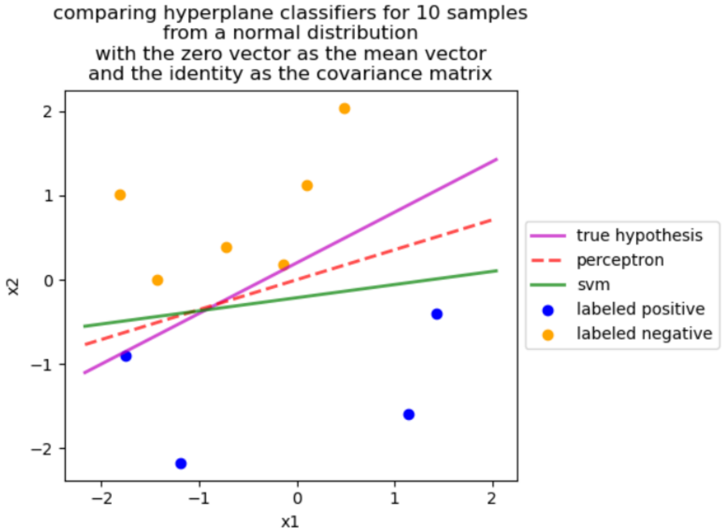
$$y_i\left\langle \mathrm{w},\mathrm{x}_i\right\rangle\geq 1-\xi_i\implies \xi_i\geq 1-y_i\left\langle \mathrm{w},\mathrm{x}_i\right\rangle$$

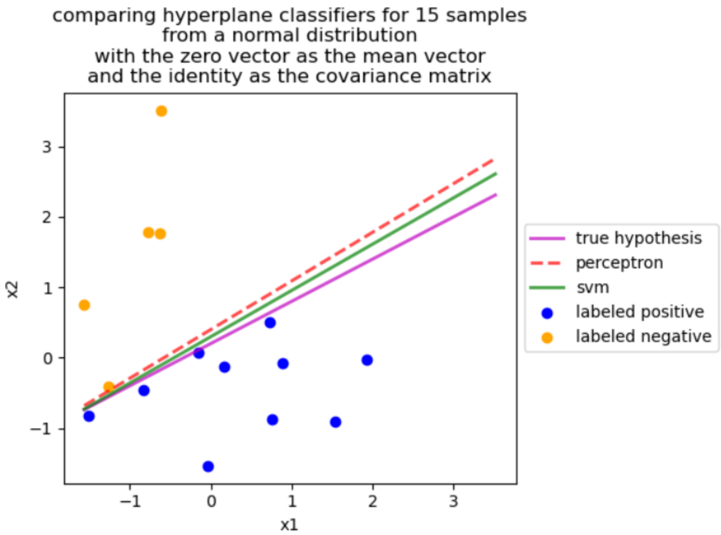
ולכן כשאנחנו ממזערים $\sum_{i=1}^{m}\xi_i$ אנחנו למעשה גם ממזערים את $1-y_i\left\langle \mathrm{w},\mathrm{x}_i\right\rangle$ ולכן המינימומים של שתי הבעיות מתקבלים באותן נקודות.

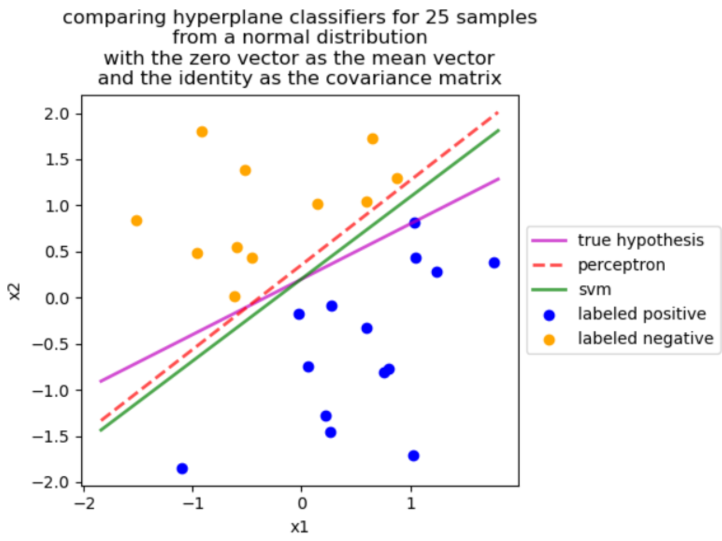
מש"ל

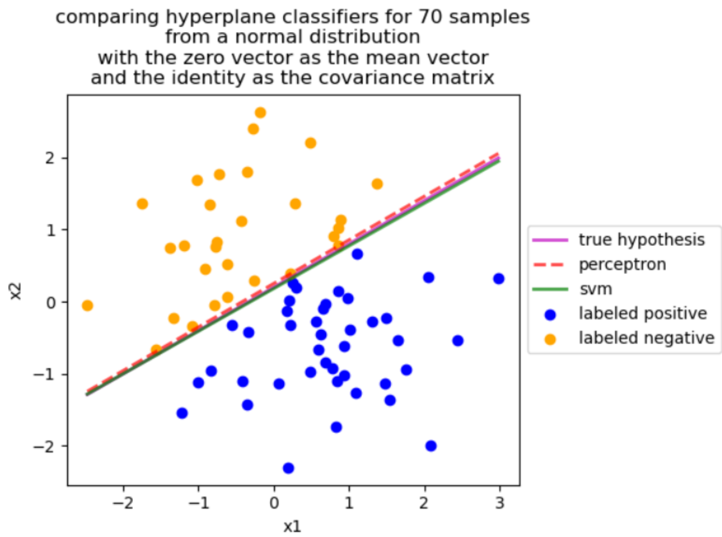
10. הגרפים להלן:



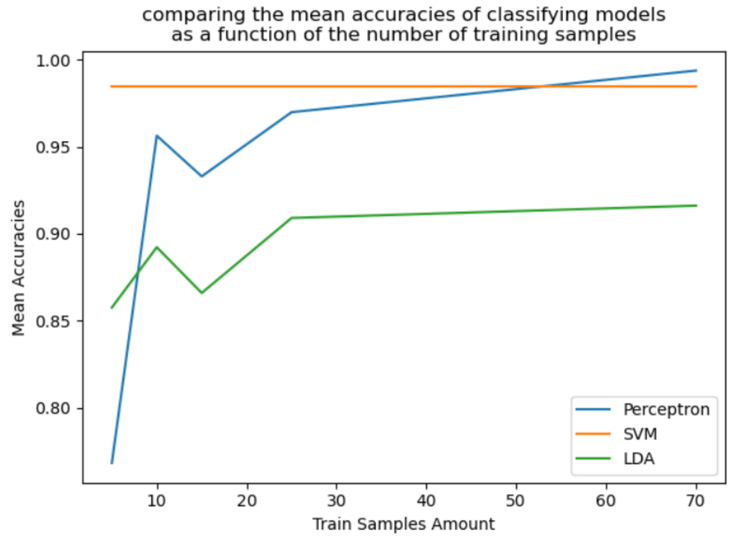





comparing hyperplane classifiers for 25 samples
from a normal distribution
with the zero vector as the mean vector
and the identity as the covariance matrix
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
x2
−2
−1
0
1
x1
true hypothesis
perceptron
svm
labeled positive
labeled negative


comparing hyperplane classifiers for 70 samples
from a normal distribution
with the zero vector as the mean vector
and the identity as the covariance matrix
2
1
0
−1
−2
x2
−2
−1
0
1
2
3
x1
true hypothesis
perceptron
svm
labeled positive
labeled negative

10. הגרף להלן:



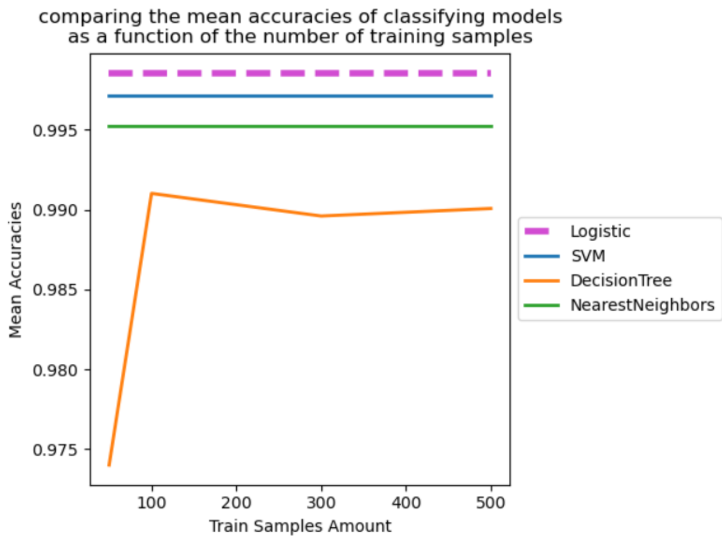
11. בממוצע, ל- SVM יש את הביצוע הטוב ביותר, אף על פי שבאזור ה- 55 דגימות, הפרספטרון עוקף אותו.

הסיבה לכך היא ש- SVM מוצא את הקו שממקסם את את רוחב השוליים, מה שנותן לו מעין "טווח ביטחון" עבור הדגימות הבאות.

אכן, ככל שהשוליים צרים יותר כך גדלה ההסתברות לשגיאה, ולכן שוליים מקסימליים ממזערים את ההסתברות לכך (קרי שטח גדול יותר שמאפשר תנודות של הדגימות).

14. הגרף להלן:

עבור המימוש של Logistic, SVM, DecisionTree לקחתי את אותו המימוש מהקובץ models.py כאשר עבור עומק מקסימלי של DesicionTree לקחתי עומק של 5. כמו כן, עבור NearestNeighbors לקחתי את המימוש של NearestCentroid שעבור כל דגימה לוקח את הלייבל של קבוצת הדגימות שהצנטרואיד הממוצע שלהם הוא הקרוב ביותר לדגימה שמסתכלים עליה.



זמני הריצה להלן:

```
Logistic, 50 training samples: 0.17 seconds
Logistic, 100 training samples: 0.19 seconds
Logistic, 300 training samples: 0.19 seconds
Logistic, 500 training samples: 0.17 seconds
SVM, 50 training samples: 3.58 seconds
SVM, 100 training samples: 3.56 seconds
SVM, 300 training samples: 3.53 seconds
SVM, 500 training samples: 3.53 seconds
DecisionTree, 50 training samples: 0.37 seconds
DecisionTree, 100 training samples: 0.39 seconds
DecisionTree, 300 training samples: 0.37 seconds
DecisionTree, 500 training samples: 0.36 seconds
NearestNeighbors, 50 training samples: 0.47 seconds
NearestNeighbors, 100 training samples: 0.48 seconds
NearestNeighbors, 300 training samples: 0.47 seconds
NearestNeighbors, 500 training samples: 0.47 seconds
```

ניתן לראות שבאחוזים אלגוריתם ה־ SVM לוקח משמעותית יותר זמן משאר האלגוריתמים.

מכאן ניתן לשער שייתכן כי הדבר טמון בכך שמקסום השוליים יותר קשה לחישוב משאר החישובים שהאלגוריתמים האחרים עושים.

2 comparison.py

```
import numpy as np
from pandas import DataFrame
from models import Model, Perceptron, SVM, LDA
import matplotlib.pyplot as plt
import time


WEIGHTS = [0.3, -0.5]
WEIGHTS_VEC = np.array([WEIGHTS])
INTERCEPT = 0.1
SAMPLES_AMOUNTS = [5, 10, 15, 25, 70]
TEST_AMOUNT = 10000
RUNS_AMOUNT = 500
MODELS = [Perceptron, SVM, LDA]


def draw_points_helper(samples_amount):

    def true_hypothesis(sample):
        return np.sign(WEIGHTS_VEC @ sample + INTERCEPT)

    identity_mat = np.identity(2)
    zero_vec = np.zeros((2,))

    samples = np.random.multivariate_normal(
        zero_vec, identity_mat, size=samples_amount).transpose()

    true_labels = np.array(
        [true_hypothesis(sample)[0] for sample in samples.transpose()])

    return samples, true_labels


def draw_points(samples_amount, helper):
    samples, true_labels = helper(samples_amount)
    max_sum = len(true_labels)
    current_sum = np.sum(true_labels)

    while current_sum == max_sum or current_sum == -max_sum:
        samples, true_labels = helper(samples_amount)
        current_sum = np.sum(true_labels)

    return samples.transpose(), true_labels


def q9_helper(samples_amount):
    samples, true_labels = \
        draw_points(samples_amount, draw_points_helper(samples_amount))
    xx = np.linspace(np.min(samples), np.max(samples))

    def get_yy(weights, intercept):
        slope = -weights[0] / weights[1]
        return slope * xx - intercept / weights[1]

    yy_f = get_yy(WEIGHTS, INTERCEPT)
    perceptron = Perceptron(samples, true_labels).model
    yy_perceptron = get_yy(perceptron[1:], perceptron[0])
    svm = SVM(samples, true_labels).model
    svm_weights = svm.coef_[0]
```

```
    yy_svm = get_yy(svm_weights, svm.intercept_[0])

    df = DataFrame({'x1': samples[:, 0], 'x2': samples[:, 1],
                    'class': true_labels})

    plt.scatter(df.loc[df['class'] == 1]['x1'],
                df.loc[df['class'] == 1]['x2'],
                color='blue', label='labeled positive')

    plt.scatter(df.loc[df['class'] == -1]['x1'],
                df.loc[df['class'] == -1]['x2'],
                color='orange', label='labeled negative')

    plt.plot(
        xx, yy_f, 'm', linewidth=2, alpha=0.7, label='true hypothesis')

    plt.plot(
        xx, yy_perceptron, '--r', linewidth=2, alpha=0.7,
        label='perceptron')

    plt.plot(
        xx, yy_svm, 'g', linewidth=2, alpha=0.7, label='svm')

    plt.title('comparing hyperplane classifiers for {} samples'
              '\nfrom a normal distribution'
              '\nwith the zero vector as the mean vector'
              '\nand the identity as the covariance matrix'
              .format(samples_amount))

    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
    plt.show()


def q9():
    for amount in SAMPLES_AMOUNTS:
        q9_helper(amount)


# q9()


def get_accuracy(model, test_samples, test_true_labels):
    test_samples_amount = len(test_samples)
    pred = model.predict(test_samples)
    compare_mat = pred - test_true_labels
    accuracy_amount = test_samples_amount - np.count_nonzero(
        compare_mat)
    return accuracy_amount / test_samples_amount


def get_mean_accuracy(runs_amount, model, test_samples, test_true_labels):
    start_time = time.time()
    accuracy = 0
    for i in range(runs_amount):
        current_acc = get_accuracy(model, test_samples, test_true_labels)
        accuracy += current_acc
    print('  {}, {} training samples: {} seconds'.
          format(model.model.__class__.__name__,
                 model.training_amount,
                 round(time.time() - start_time, 2)))
    return accuracy / runs_amount


def get_train_samples_and_labels_dict(samples_amounts, draw_points_helper):
    samples_and_labels_dict = dict()
    for amount in samples_amounts:
```

```
        samples_and_labels_dict[amount] = \
            draw_points(amount, draw_points_helper)
    return samples_and_labels_dict


def get_model(model_class, train_samples, train_true_labels):
    return Model(model_class(train_samples, train_true_labels),
                 len(train_samples))


def get_model_versions(samples_amounts, model_class, samples_and_labels_dict):
    return [get_model(model_class, samples_and_labels_dict[amount][0],
                      samples_and_labels_dict[amount][1])
            for amount in samples_amounts]


def get_mean_accuracies(samples_amounts, runs_amount, model_class,
                        samples_and_labels_dict,
                        test_samples, test_true_labels):
    model_versions = get_model_versions(samples_amounts, model_class,
                                        samples_and_labels_dict)
    return [get_mean_accuracy(runs_amount, model, test_samples,
                              test_true_labels)
            for model in model_versions]


def question_helper(train_samples_and_labels_dict, test_samples,
                    test_true_labels, samples_amounts, models, runs_amount):
    for model_class in models:
        model_mean_accs = get_mean_accuracies(samples_amounts, runs_amount,
                     model_class, train_samples_and_labels_dict,
                     test_samples, test_true_labels)
        if model_class.__name__ == 'Logistic':
            plt.plot(samples_amounts, model_mean_accs, '--m', linewidth=4,
                     alpha=0.7, label='{}'.format(model_class.__name__))
        else:
            plt.plot(samples_amounts, model_mean_accs, linewidth=2,
                     label='{}'.format(model_class.__name__))
    plt.xlabel('Train Samples Amount')
    plt.ylabel('Mean Accuracies')
    plt.title('comparing the mean accuracies of classifying models'
              '\nas a function of the number of training samples')
    plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
    plt.show()


def run_question(samples_amounts, models, runs_amount, draw_points_helper,
                 test_samples, test_true_labels):

    train_samples_and_labels_dict = \
        get_train_samples_and_labels_dict(samples_amounts, draw_points_helper)

    question_helper(train_samples_and_labels_dict, test_samples, test_true_labels,
                    samples_amounts, models, runs_amount)


def q10(samples_amounts, test_amount, models, runs_amount, draw_points_helper):

    test_samples, test_true_labels = draw_points(
        test_amount, draw_points_helper)

    run_question(samples_amounts, models, runs_amount,
                 draw_points_helper, test_samples, test_true_labels)


# q10(SAMPLES_AMOUNTS, TEST_AMOUNT, MODELS, RUNS_AMOUNT, draw_points_helper)
```

3 mnist data.py

```
import matplotlib.pyplot as plt
from models import *
from comparison import run_question
from sklearn.neighbors import NearestCentroid


class NearestNeighbors:
    model = NearestCentroid()

    def __init__(self, samples, true_labels):
        self.model.fit(samples, true_labels)

    def predict(self, samples):
        return self.model.predict(samples)


SAMPLES_AMOUNTS = [50, 100, 300, 500]
RUNS_AMOUNT = 50
MODELS = [Logistic, SVM, DecisionTree, NearestNeighbors]


image_size = 28
no_of_different_labels = 2
image_pixels = image_size * image_size
data_path = r'C:\Users\amitb\PycharmProjects\IML\ex3\dataset\mldata'
train_data = np.loadtxt(data_path + r"\mnist_train.csv",
                        delimiter=",")
test_data = np.loadtxt(data_path + r"\mnist_test.csv",
                       delimiter=",")

fac = 0.99 / 255
all_train_imgs = np.asfarray(train_data[:, 1:]) * fac + 0.01
all_test_imgs = np.asfarray(test_data[:, 1:]) * fac + 0.01

all_train_labels = np.asfarray(train_data[:, :1])
all_test_labels = np.asfarray(test_data[:, :1])

binary_train_indices = np.logical_or((all_train_labels == 0),
                                     (all_train_labels == 1))
binary_test_indices = np.logical_or((all_test_labels == 0),
                                    (all_test_labels == 1))

binary_train_indices = binary_train_indices.reshape(
     binary_train_indices.shape[0])
binary_test_indices = binary_test_indices.reshape(
    binary_test_indices.shape[0])

x_train, y_train = all_train_imgs[binary_train_indices], \
                   all_train_labels[binary_train_indices]
x_test, y_test = all_test_imgs[binary_test_indices], \
                 all_test_labels[binary_test_indices]

y_train = y_train.reshape((len(y_train),))
y_test = y_test.reshape((len(y_test),))
y_train[y_train == 0] = -1
y_test[y_test == 0] = -1


def q12():
```

```python
    positive_counter = 0
    negative_counter = 0
    while positive_counter < 3 or negative_counter < 3:
        for i in range(len(x_train)):
            img = x_train[i].reshape((image_size, image_size))
            if y_train[i][0] == 1 and positive_counter < 3:
                plt.imshow(img)
                plt.show()
                positive_counter += 1
            elif y_train[i][0] == -1 and negative_counter < 3:
                plt.imshow(img)
                plt.show()
                negative_counter += 1


# q12()


def q14_draw_points_helper(samples_amount):
    true_arr = np.ones(samples_amount, dtype=bool)
    false_arr = np.zeros(len(x_train) - samples_amount, dtype=bool)
    indices_arr = np.concatenate((true_arr, false_arr))
    np.random.shuffle(indices_arr)
    train_samples = x_train[indices_arr]
    true_labels = y_train[indices_arr]
    return train_samples.transpose(), true_labels


def q14():
    run_question(SAMPLES_AMOUNTS, MODELS, RUNS_AMOUNT,
                 q14_draw_points_helper, x_test, y_test)


# q14()
```

4 models.py

```
import numpy as np
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from pandas import *


def model_score(model, samples, true_labels):
    return Model(model, model.training_amount).score(samples, true_labels)


class Model:
    model = None
    training_amount = 0

    def __init__(self, model, training_amount):
        self.model = model
        self.training_amount = training_amount

    def predict(self, samples):
        return self.model.predict(samples)

    def score(self, samples, true_labels):
        samples_amount = len(samples.transpose())
        predictions = self.predict(samples)
        compare_mat = predictions - true_labels
        accuracy_amount = samples_amount - np.count_nonzero(compare_mat)
        accuracy_rate = accuracy_amount / samples_amount
        error_amount = samples_amount - accuracy_amount
        error_rate = error_amount / samples_amount

        false_positive_mat = (compare_mat > 0)
        false_positive_amount = np.sum(false_positive_mat)
        false_positive_rate = false_positive_amount / samples_amount

        predicted_positive = (predictions == 1)
        labeled_positive = (true_labels == 1)
        true_positive_mat = (predicted_positive == labeled_positive)
        true_positive_amount = np.sum(true_positive_mat)
        true_positive_rate = true_positive_amount / samples_amount
        predicted_true_amount = np.sum(predicted_positive)
        precision = true_positive_amount / predicted_true_amount
        labeled_true_amount = np.sum(labeled_positive)
        recall = true_positive_amount / labeled_true_amount

        scores = dict()
        scores['num_samples'] = samples_amount
        scores['error'] = error_rate
        scores['accuracy'] = accuracy_rate
        scores['FPR'] = false_positive_rate
        scores['TPR'] = true_positive_rate
        scores['precision'] = precision
        scores['recall'] = recall

        return scores


def get_conditions_mat(samples, labels):
    samples = add_ones_col(samples)
```

```
    conditions_mat = np.array(
        [samples[i] * labels[i] for i in range(len(labels))])
    return conditions_mat


def add_ones_col(samples):
    samples_amount = len(samples)
    new_row = np.ones((samples_amount, 1))
    samples = np.concatenate((new_row, samples), axis=1)
    return samples


class Perceptron:
    model = None
    training_amount = 0

    def __init__(self, samples, labels):
        self.training_amount = len(samples)
        self.fit(samples, labels)

    def fit(self, samples, labels):
        conditions_mat = get_conditions_mat(samples, labels)
        row_len = conditions_mat.shape[1]
        weights = np.zeros((row_len,))
        min_val = 0
        while min_val <= 0:
            current_result = np.matmul(conditions_mat, weights)
            min_val_index = np.argmin(current_result)
            min_val = current_result[min_val_index]
            if min_val <= 0:
                min_vec = conditions_mat[min_val_index]
                weights += min_vec
                continue
        self.model = weights

    def predict(self, samples):
        samples = add_ones_col(samples)
        return np.sign(np.matmul(samples, self.model))

    def score(self, samples, true_labels):
        return model_score(self, samples, true_labels)


class LDA:
    model = None
    training_amount = 0

    def __init__(self, samples, labels):
        self.training_amount = len(samples)
        self.fit(samples, labels)

    def fit(self, samples, labels):

        labeled_positive_indices = list((labels == 1).nonzero())[0]
        labeled_negative_indices = list((labels == -1).nonzero())[0]

        positive_samples = samples[labeled_positive_indices, :]
        negative_samples = samples[labeled_negative_indices, :]

        positive_mean = np.mean(positive_samples, axis=0)
        negative_mean = np.mean(negative_samples, axis=0)

        samples_amount = len(samples)
        total_mean = np.mean(samples, axis=1)
        samples_centered = (samples.transpose() - total_mean).transpose()
        positive_samples = samples_centered[labeled_positive_indices, :]
        negative_samples = samples_centered[labeled_negative_indices, :]

```

```
        positive_cov = positive_samples.transpose() @ positive_samples
        negative_cov = negative_samples.transpose() @ negative_samples

        general_scalar = 1 / samples_amount
        general_cov = general_scalar * (positive_cov + negative_cov)
        general_cov_inverse = np.linalg.pinv(general_cov)

        positive_amount = len(positive_samples)
        negative_amount = len(negative_samples)
        positive_prob = positive_amount / samples_amount
        negative_prob = negative_amount / samples_amount

        def delta_label(sample, label_mean, label_prob):
            first = sample @ general_cov_inverse @ label_mean \
                   - 0.5 * label_mean.transpose() @ general_cov_inverse @ \
                   label_mean
            return first + np.log(label_prob)

        def delta_positive(sample):
            return delta_label(sample, positive_mean, positive_prob)

        def delta_negative(sample):
            return delta_label(sample, negative_mean, negative_prob)

        def max_delta(sample):
            return max(delta_positive(sample), delta_negative(sample))

        def get_label(sample):
            if max_delta(sample) == delta_positive(sample):
                return 1
            else:
                return -1

        def get_predictions(new_samples):
            return np.array([get_label(sample) for sample in new_samples])

        self.model = get_predictions

    def predict(self, samples):
        return self.model(samples)

    def score(self, samples, true_labels):
        return model_score(self, samples, true_labels)


class SVM:
    model = SVC(C=1e10, kernel='linear')
    training_amount = 0

    def __init__(self, samples, labels):
        self.training_amount = len(samples)
        self.fit(samples, labels)

    def fit(self, samples, labels):
        self.model.fit(samples, labels)

    def predict(self, samples):
        return self.model.predict(samples)


class Logistic:
    model = LogisticRegression(solver='liblinear')
    training_amount = 0

    def __init__(self, samples, labels):
        self.training_amount = len(samples)
        self.fit(samples, labels)

```

```python
    def fit(self, samples, labels):
        self.model.fit(samples, labels)

    def predict(self, samples):
        return self.model.predict(samples)

    def score(self, samples, true_labels):
        return model_score(self, samples, true_labels)


class DecisionTree:
    model = None
    training_amount = 0

    def __init__(self, samples, labels):
        self.training_amount = len(samples)
        self.model = DecisionTreeClassifier(max_depth=5)
        self.fit(samples, labels)

    def fit(self, samples, labels):
        self.model.fit(samples, labels)

    def predict(self, samples):
        return self.model.predict(samples)

    def score(self, samples, true_labels):
        return model_score(self, samples, true_labels)
```