Contents

מבוא למערכות לומדות תרגיל 5

עמית בסקין 312259013

חלק I

חלק תיאורתי

1 ואלידציה

בשאלה זו נראה מתי לפרדיגמת "בחירת מודל" יש יתרון על פני השיטה הסטנדרטית כאשר בוחרים בין k מחלקות היפותיזות אפשריות:

$$\mathcal{H}_1 \subseteq \mathcal{H}_2 \subseteq \ldots \subseteq \mathcal{H}_k$$

עבור $\mathcal{H}_k$ סופית.

תהא $S_{all} = \{(x_i, y_i)\}_1^m$ קבוצת דגימות. כרגיל, נרצה ללמוד היפותיזה עם שגיאת הכללה קטנה ככל האפשר.

בכיתה דנו בבעיית התאמת פולינום, שם היו לנו k מחלקות היפותיזות לבחור מביניהן:

$$\mathcal{H}_1 \subseteq \mathcal{H}_2 \subseteq \ldots \subseteq \mathcal{H}_k$$

בשאלה זו נשווה בין שתי שיטות לבחירת היפותיזה:

- השיטה הסטנדרטית: לבחור את ההיפותיזה הטובה ביותר ב־ $\mathcal{H}_k$ על ידי שימוש בדגימות הנתונות ובכלל ה־ ERM.
- שיטת "בחירת מודל":
  - ניקח איזשהי $\alpha \in (0,1)$ ונחלק את m הדגימות לקבוצת אימון S מגודל $(1-\alpha)\,m$ וקבוצת ואלידציה V מגודל αm, עבור αm מספר שלם.
  - לכל מחלקת היפותיזות $\mathcal{H}_i$ עם $i \in [k]$, נמצא $h_i \in ERM_{\mathcal{H}_i}(S)$.
  - נחזיר $h^* \in ERM_{\mathcal{H}}(V)$ כאשר $\mathcal{H} = \{h_i\}_1^k$.

נניח ש־ $\mathcal{H}_k$ סופית ופונקציית ההפסד חסומה מלמעלה על ידי 1.

א. צ.לחסום את שגיאת ההכללה שמתקבלת בשיטה הסטדנרטית: להוכיח שלמידה־PAC־אגנוסיטית של $\mathcal{H}_k$, מקיימת את החסם הבא:

עבור $h^* \in ERM_{\mathcal{H}_k}(S_{all})$, עם הסתברות של $1-\delta$ לכל הפחות, מתקיים:

$$L_{\mathcal{D}}(h^*) \le \min_{h \in \mathcal{H}_k} L_{\mathcal{D}}(h) + \sqrt{\frac{2\ln\left(\frac{2}{\delta}\left|\mathcal{H}_k\right|\right)}{m}}$$

פיתרון:

באמצעות חסם הופדינג, הוכחנו בתרגול שלכל $\delta \in (0,1)$, בהסתברות של $1-\delta$ לכל הפחות, ולכל היפותיזה $h \in \mathcal{H}_k$, מתקיים:

$$\left|L_{S_{all}}(h) - L_{\mathcal{D}}(h)\right| \leq \sqrt{\frac{\ln\left(\frac{2}{\delta}\right)}{2m}}$$

ובפרט בהסתברות של $1-\frac{\delta}{|\mathcal{H}_k|}$ לכל הפחות, מתקיים:

$$\left|L_{S_{all}}(h) - L_{\mathcal{D}}(h)\right| \leq \sqrt{\frac{\ln\left(\frac{2}{\delta}\left|\mathcal{H}_k\right|\frac{2}{\delta}\right)}{2m}}$$

אז בהסתברות של $\frac{\delta}{|\mathcal{H}_k|}$ לכל היותר מתקיים:

$$\left|L_{S_{all}}(h) - L_{\mathcal{D}}(h)\right| \geq \sqrt{\frac{\ln\left(\frac{2}{\delta}\left|\mathcal{H}_k\right|\right)}{2m}}$$

ומחסם האיחוד, ההסתברות ש**קיימת** היפותיזה h ב- $\mathcal{H}_k$ עם הא"ש לעיל, היא לכל היותר $\frac{\delta}{|\mathcal{H}_k|}\left|\mathcal{H}_k\right|$ קרי δ, ולכן ההסתברות ש**לכל** היפותיזה h ב- $\mathcal{H}_k$ מתקיים:

$$\left|L_{S_{all}}(h) - L_{\mathcal{D}}(h)\right| \leq \sqrt{\frac{\ln\left(\frac{2}{\delta}\left|\mathcal{H}_k\right|\right)}{2m}}$$

היא לכל הפחות $1-\delta$, ובפרט לכל $h \in \mathcal{H}_k$ ההסתברות שמתקיים הא"ש הנ"ל היא לכל הפחות $1-\delta$.

אם כן, מאחר ששגיאת ההכללה גדולה יותר באופן טיפוסי, אזי שבפרט עבור h^* מתקיים בהסתברות של $1-\delta$ לכל הפחות הא"ש:

$$L_{\mathcal{D}}(h^*) \leq L_{S_{all}}(h^*) + \sqrt{\frac{\ln\left(\frac{2}{\delta}\left|\mathcal{H}_k\right|\right)}{2m}}$$

אבל $h^* \in ERM_{\mathcal{H}_k}(S_{all})$ ולכן לכל $h \in \mathcal{H}_k$:

$$\leq L_{S_{all}}(h) + \sqrt{\frac{\ln\left(\frac{2}{\delta}\left|\mathcal{H}_k\right|\right)}{2m}}$$

ומקיום הא"ש לעיל עבור כל $h \in \mathcal{H}_k$:

$$\leq L_{\mathcal{D}}(h) + 2\sqrt{\frac{\ln\left(\frac{2}{\delta}\left|\mathcal{H}_k\right|\right)}{2m}}$$

נכניס את ה- 2 לתוך השורש:

$$= L_{\mathcal{D}}(h) + \sqrt{\frac{2\ln\left(\frac{2}{\delta}\left|\mathcal{H}_k\right|\right)}{m}}$$

ומאחר שזה נכון לכל $h \in \mathcal{H}_k$, אזי ש-

$$L_{\mathcal{D}}(h^*) \leq \min_{h \in \mathcal{H}_k} L_{\mathcal{D}}(h) + \sqrt{\frac{2\ln\left(\frac{2}{\delta}\left|\mathcal{H}_k\right|\right)}{m}}$$

כאמור בהסתברות של $1-\delta$ לכל הפחות. מש"ל

ב. צ.לחסום את שגיאת ההכללה על ידי שימוש בשיטת "בחירת מודל":

נסמן: $h' = \underset{h\in\mathcal{H}_k}{\arg\min} L_{\mathcal{D}}\left(h\right)$. יהא j מינימלי עם $h' \in \mathcal{H}_j$.

צ.להוכיח שבהסתברות של לפחות $1-\delta$ מתקיים כי:

$$L_{\mathcal{D}}\left(h^{*}\right) \leq \min_{h\in\mathcal{H}_k} L_{\mathcal{D}}\left(h\right) + \sqrt{\frac{2}{\alpha m}\ln\left(\frac{4}{\delta}\left|\mathcal{H}_k\right|\right)} + \sqrt{\frac{2}{\left(1-\alpha\right)m}\ln\left(\frac{4}{\delta}\left|\mathcal{H}_j\right|\right)}$$

פיתרון:

מהסעיף הקודם, על ידי לקיחת $\delta/2$ והפעלת המסקנה על כל $\mathcal{H}_i$ בנפרד, נקבל שלכל $i\in[k]$ ו־ $h_i \in ERM_{\mathcal{H}_i}\left(S\right)$ מתקיים:

$$\mathbb{P}\left(L_{\mathcal{D}}\left(h_i\right) \leq \min_{h\in\mathcal{H}_i} L_{\mathcal{D}}\left(h\right) + \sqrt{\frac{2\ln\left(\frac{4}{\delta}\left|\mathcal{H}_i\right|\right)}{\left(1-\alpha\right)m}}\right) \geq 1-\frac{\delta}{2}$$

ועבור h^{*} שנבחרה בשלב הואלידציה:

$$\mathbb{P}\left(L_{\mathcal{D}}\left(h^{*}\right) \leq \min_{h\in\mathcal{H}_k} L_{\mathcal{D}}\left(h\right) + \sqrt{\frac{2\ln\left(\frac{4}{\delta}k\right)}{\alpha m}}\right) \geq 1-\frac{\delta}{2}$$

(כי משווים בין k היפותיזות)

אבל $h' = \underset{h\in\mathcal{H}_k}{\arg\min} L_{\mathcal{D}}\left(h\right)$, ולכן בהסתברות של לכל הפחות $1-\delta$, מתקיים:

$$L_{\mathcal{D}}\left(h^{*}\right) \leq L_{\mathcal{D}}\left(h'\right) + \sqrt{\frac{2\ln\left(\frac{4}{\delta}k\right)}{\alpha m}}$$

ומקיום הא"ש הנ"ל עבור $\mathcal{H}_j$:

$$\leq \min_{h\in\mathcal{H}_j} L_{\mathcal{D}}\left(h\right) + \sqrt{\frac{2\ln\left(\frac{4}{\delta}\left|\mathcal{H}_j\right|\right)}{\left(1-\alpha\right)m}} + \sqrt{\frac{2\ln\left(\frac{4}{\delta}k\right)}{\alpha m}}$$

אבל $\min_{h\in\mathcal{H}_j} L_{\mathcal{D}}\left(h\right) = \min_{h\in\mathcal{H}_k} L_{\mathcal{D}}\left(h\right)$ ולכן:

$$= \min_{h\in\mathcal{H}_k} L_{\mathcal{D}}\left(h\right) + \sqrt{\frac{2\ln\left(\frac{4}{\delta}\left|\mathcal{H}_j\right|\right)}{\left(1-\alpha\right)m}} + \sqrt{\frac{2\ln\left(\frac{4}{\delta}k\right)}{\alpha m}}$$

וזאת כאמור בהסתברות של $1-\delta$ לכל הפחות. מש"ל

ג. צ.להראות ששני החסמים אינם בני־השוואה: כלומר צ.לתאר מקרה שבו השיטה הסטנדרטית עדיפה ומקרה הפוך.

הוכחה:

עבור השיטה הסטנדרטית קיבלנו:

$$L_{\mathcal{D}}\left(h^{*}\right) \leq \min_{h\in\mathcal{H}_k} L_{\mathcal{D}}\left(h\right) + \sqrt{\frac{2\ln\left(\frac{2}{\delta}\left|\mathcal{H}_k\right|\right)}{m}}$$

ועבור שיטת "בחירת מודל":

$$L_{\mathcal{D}}\left(h^{*}\right) \leq \min_{h\in\mathcal{H}_k} L_{\mathcal{D}}\left(h\right) + \sqrt{\frac{2\ln\left(\frac{4}{\delta}\left|\mathcal{H}_j\right|\right)}{\left(1-\alpha\right)m}} + \sqrt{\frac{2\ln\left(\frac{4}{\delta}k\right)}{\alpha m}}$$

אז נרצה לתת דוגמה לשני הבאים:

$$\min_{h\in\mathcal{H}_k} L_{\mathcal{D}}\left(h\right)+\sqrt{\frac{2\ln\left(\frac{2}{\delta}\left|\mathcal{H}_k\right|\right)}{m}}<\min_{h\in\mathcal{H}_k} L_{\mathcal{D}}\left(h\right)+\sqrt{\frac{2\ln\left(\frac{4}{\delta}\left|\mathcal{H}_j\right|\right)}{\left(1-\alpha\right)m}}+\sqrt{\frac{2\ln\left(\frac{4}{\delta}k\right)}{\alpha m}}$$

ו־

$$\min_{h\in\mathcal{H}_k} L_{\mathcal{D}}\left(h\right)+\sqrt{\frac{2\ln\left(\frac{4}{\delta}\left|\mathcal{H}_j\right|\right)}{\left(1-\alpha\right)m}}+\sqrt{\frac{2\ln\left(\frac{4}{\delta}k\right)}{\alpha m}}<\min_{h\in\mathcal{H}_k} L_{\mathcal{D}}\left(h\right)+\sqrt{\frac{2\ln\left(\frac{2}{\delta}\left|\mathcal{H}_k\right|\right)}{m}}$$

קרי:

$$\sqrt{\frac{2\ln\left(\frac{2}{\delta}\left|\mathcal{H}_k\right|\right)}{m}}<\sqrt{\frac{2\ln\left(\frac{4}{\delta}\left|\mathcal{H}_j\right|\right)}{\left(1-\alpha\right)m}}+\sqrt{\frac{2\ln\left(\frac{4}{\delta}k\right)}{\alpha m}}$$
$$\sqrt{\frac{2\ln\left(\frac{4}{\delta}\left|\mathcal{H}_j\right|\right)}{\left(1-\alpha\right)m}}+\sqrt{\frac{2\ln\left(\frac{4}{\delta}k\right)}{\alpha m}}<\sqrt{\frac{2\ln\left(\frac{2}{\delta}\left|\mathcal{H}_k\right|\right)}{m}}$$

- אם מתקיים $j=k$ ו־ $\alpha=\frac{1}{2}$, אז עבור הא"ש הראשון מתקיים:

$$\sqrt{\frac{2\ln\left(\frac{4}{\delta}\left|\mathcal{H}_j\right|\right)}{0.5m}}+\sqrt{\frac{2\ln\left(\frac{4}{\delta}k\right)}{0.5m}}=$$

$$=2\sqrt{\frac{\ln\left(\frac{4}{\delta}\left|\mathcal{H}_k\right|\right)}{m}}+2\sqrt{\frac{\ln\left(\frac{4}{\delta}k\right)}{m}}$$

אבל:

$$2\sqrt{\ln\left(\frac{4}{\delta}\left|\mathcal{H}_k\right|\right)}+2\sqrt{\ln\left(\frac{4}{\delta}k\right)}>\sqrt{2\ln\left(\frac{2}{\delta}\left|\mathcal{H}_k\right|\right)}$$

ולכן מתקיים הא"ש הראשון במקרה הזה.

- עבור הא"ש השני:

אם $k,\left|\mathcal{H}_j\right|<e^8\delta$ ו־ $\left|\mathcal{H}_k\right|>\frac{1}{2}\delta e^{50}$, ו־ $\alpha=0.5$, אז:

$$\sqrt{\frac{2\ln\left(\frac{4}{\delta}\left|\mathcal{H}_j\right|\right)}{\left(1-\alpha\right)m}}+\sqrt{\frac{2\ln\left(\frac{4}{\delta}k\right)}{\alpha m}}<$$

$$<2\sqrt{\frac{8}{0.5m}}=2\sqrt{\frac{16}{m}}=\frac{8}{\sqrt{m}}$$

ואילו:

$$\sqrt{\frac{2\ln\left(\frac{2}{\delta}\left|\mathcal{H}_k\right|\right)}{m}}>\sqrt{\frac{2\ln\left(e^{50}\right)}{m}}=\frac{10}{\sqrt{m}}$$

ולכן מתקיים הא"ש השני במקרה הזה.

2 תכנון אורתוגונלי

במקרה המיוחד של מטריצת תכנון אורתוגונלית, קרי $XX^t = Id$, ניתן לגזור נוסחאות סגורות עבור Best Subset, Lasso ו־ Ridge, כפונקציה של LS.

תהא $\lambda > 0$. נסמן ב־ $\hat{w}^{LS}, \hat{w}_{\lambda}^{ridge}, \hat{w}_{\lambda}^{lasso}, \hat{w}_{\lambda}^{subset}$ את הפתרונות עבור רגרסיה לינארית סטנדרטית, Ridge, Lasso ו־ Best Subset, בהתאמה.

ניזכר בפונקציות הבאות:

$$\eta_{\lambda}^{soft}\left(x\right)=\begin{cases} x-\lambda & x\geq0\\ 0 & |x|<\lambda\\ x+\lambda & x\leq-\lambda \end{cases}$$

$$\eta_{\lambda}^{hard}\left(x\right)=\mathbf{1}\left[|x|\geq\lambda\right]\cdot x$$

צ.להוכיח ש־

א.

$$\hat{w}_{\lambda}^{ridge}=\frac{\hat{w}^{LS}}{1+\lambda}$$

ב.

$$\hat{w}_{\lambda}^{subset}=\eta_{\sqrt{\lambda}}^{hard}\left(\hat{w}^{LS}\right)$$

הוכחה:

א. בתרגול ראינו ש־

$$\hat{\mathrm{w}}_{\lambda}^{ridge}=\left(XX^{t}+\lambda I\right)^{-1}Xy$$

ומכך ש־ $XX^t = I$:

$$\hat{\mathrm{w}}_{\lambda}^{ridge}=\left(I+\lambda I\right)^{-1}Xy=\frac{1}{1+\lambda}Xy$$

אבל:

$$\hat{\mathrm{w}}^{LS}=\left(XX^{t}\right)^{-1}Xy=Xy$$

ולכן:

$$\hat{\mathrm{w}}_{\lambda}^{ridge}=\frac{\hat{w}^{LS}}{1+\lambda}$$

כנדרש.

ב. בדומה לטענה 2 בתרגול, נכתוב:

$$f_{l_0}(w) = \left\|y - X^t w\right\|_2^2 + \lambda \|w\|_0 =$$

$$= \|y\|_2^2 - 2y^t X^t w + w^t X X^t w + \lambda \|w\|_0 =$$

$$= \|y\|_2^2 + \left(w^t - 2\hat{w}^t\right) w + \lambda \|w\|_0 =$$

$$= \|y\|_2^2 + \sum_{1}^{d} \left(w_i^2 - 2\hat{w}_i w_i + \lambda \|w_i\|_0\right)$$

אם $w_i = 0$ אז הנסכם מתאפס ואם $w_i \neq 0$ אז מתקבל בנסכם:

$$w_i^2 - 2\hat{w}_i w_i + \lambda$$

נגזור לפי w_i ונשווה לאפס:

$$w_i = \hat{w}_i$$

קרי:

$$\hat{w}_i^2 - 2\hat{w}_i^2 + \lambda = -\hat{w}_i^2 + \lambda$$

וזה קטן מאפס אםם $\sqrt{\lambda} \geq |\hat{w}_i|$, ולכן $\hat{w}_i \neq 0$ אםם $\sqrt{\lambda} \geq |\hat{w}_i|$.

ובסה"כ קיבלנו:

$$\hat{w}_\lambda^{subset} = \eta_{\sqrt{2\lambda}}^{hard}\left(\hat{w}^{LS}\right)$$

3 רגיולריזציה

בשאלה זו נראה שלמרות ש־ Ridge מוסיפה $bias$ להערכה של w, היא עדיין מקטינה את ה־ MSE:

תהא $X \in \mathbb{R}_{d\times m}$ מטריצת רגרסיה קבועה כך ש־ XX^t הפיכה.

יהא

$$\hat{w}(\lambda) = \arg\min_{\mathrm{w}} \left(\left\|\mathrm{y} - X^t \mathrm{w}\right\|_2^2 + \lambda \|\mathrm{w}\|_2^2\right)$$

הפיתרון של Ridge וכן יהא $\hat{\mathrm{w}} \equiv \hat{\mathrm{w}}(\lambda = 0)$ הפיתרון של LS.

- נניח שהמודל הלינארי הוא נכון, קרי:

$$\mathrm{y} = X^t \hat{\mathrm{w}} + \epsilon$$

כאשר $\epsilon_i \overset{iid}{\sim} \mathcal{N}\left(0, \sigma^2\right)$

- ניזכר כי במקרה זה מתקיים ש־ $\mathbb{E}\left[\hat{\mathrm{w}}\right]=\hat{\mathrm{w}}$

א. צ.להראות שמתקיים כי

$$\hat{\mathrm{w}}\left(\lambda\right)=A_{\lambda}\hat{\mathrm{w}}$$

כאשר

$$A_{\lambda}=\left(XX^{t}+\lambda Id\right)^{-1}XX^{t}$$

הוכחה:

$$A_{\lambda}\hat{\mathrm{w}}=\left(XX^{t}+\lambda Id\right)^{-1}XX^{t}\cdot\left(XX^{t}\right)^{-1}Xy=\left(XX^{t}+\lambda Id\right)^{-1}Xy$$

ואכן ראינו בהרצאה כי:

$$Xy=\left(XX^{t}+\lambda I\right)\hat{\mathrm{w}}\left(\lambda\right)$$

ומאחר ש־ $\left(XX^{t}+\lambda I\right)$ הפיכה אזי שקיבלנו את הנדרש.

ב. צ.להסיק מהסעיף הקודם הפיתרון של Ridge הוא עם $bias$ לכל $\lambda>0$, קרי להראות שאם $\lambda>0$ אז $\mathbb{E}\left[\hat{\mathrm{w}}\left(\lambda\right)\right]\neq\hat{\mathrm{w}}$.

הוכחה:

$$\mathbb{E}\left[\hat{\mathrm{w}}\left(\lambda\right)\right]=\mathbb{E}\left[A_{\lambda}\hat{\mathrm{w}}\right]=A_{\lambda}\mathbb{E}\left[\hat{\mathrm{w}}\right]=A_{\lambda}\hat{\mathrm{w}}\neq\hat{\mathrm{w}}$$

ג. צ.להראות ש־

$$\mathrm{Var}\left(\hat{\mathrm{w}}\left(\lambda\right)\right)=\sigma^{2}A_{\lambda}\left(XX^{t}\right)^{-1}A_{\lambda}^{t}$$

הוכחה:

נשתמש בעובדה שעבור מטריצה B שאינה אקראית ווקטור אקראי z, מתקיים:

$$\mathrm{Var}\left(B\mathrm{z}\right)=B\mathrm{Var}\left(\mathrm{z}\right)B^{t}$$

$$\mathrm{Var}\left(\hat{\mathrm{w}}\right)=\sigma^{2}\left(XX^{t}\right)^{-1}$$

קרי:

$$\mathrm{Var}\left(\hat{\mathrm{w}}\left(\lambda\right)\right)=\mathrm{Var}\left(A_{\lambda}\hat{\mathrm{w}}\right)=A_{\lambda}\mathrm{Var}\left(\hat{\mathrm{w}}\right)A_{\lambda}^{t}=$$

$$= A_\lambda \sigma^2 \left(XX^t\right)^{-1} A_\lambda^t = \sigma^2 A_\lambda \left(XX^t\right)^{-1} A_\lambda^t$$

ד. צ.לגזור ביטויים מפורשים עבור ה־ $bias$ (בריבוע) והשונות של $\hat{\mathrm{w}}(\lambda)$, כפונקציה של λ, קרי, לכתוב פירוק של bias-variance עבור ה־ MSE של $\hat{\mathrm{w}}(\lambda)$.

כמו כן צ.להראות באמצעות גזירה ש־

$$\frac{d}{d\lambda}\mathrm{MSE}(\lambda)|_{\lambda=0} = \frac{d}{d\lambda}\mathrm{bias}^2(\lambda)|_{\lambda=0} + \frac{d}{d\lambda}\mathrm{Var}(\lambda)|_{\lambda=0} < 0$$

פיתרון:

$$\mathrm{MSE}(\lambda) = (A_\lambda \hat{\mathrm{w}} - \hat{\mathrm{w}})(A_\lambda \hat{\mathrm{w}} - \hat{\mathrm{w}})^t + \sigma^2 A_\lambda \left(XX^t\right)^{-1} A_\lambda^t =$$

$$= (A_\lambda - I)\hat{\mathrm{w}}\left((A_\lambda - I)\hat{\mathrm{w}}\right)^t + \sigma^2 A_\lambda \left(XX^t\right)^{-1} A_\lambda^t =$$

$$= (A_\lambda - I)\hat{\mathrm{w}}\hat{\mathrm{w}}^t (A_\lambda - I)^t + \sigma^2 A_\lambda \left(XX^t\right)^{-1} A_\lambda^t =$$

$$= (A_\lambda - I)\hat{\mathrm{w}}\hat{\mathrm{w}}^t (A_\lambda - I) + \sigma^2 A_\lambda \left(XX^t\right)^{-1} A_\lambda^t =$$

$$= \hat{\mathrm{w}}\hat{\mathrm{w}}^t \left(\left(XX^t + \lambda Id\right)^{-1} XX^t - I\right)^2$$

$$+\sigma^2 \left(XX^t + \lambda Id\right)^{-1} XX^t \left(XX^t\right)^{-1} XX^t \left(\left(XX^t + \lambda Id\right)^{-1}\right)^t =$$

$$= \hat{\mathrm{w}}\hat{\mathrm{w}}^t \left(\left(XX^t + \lambda Id\right)^{-1} XX^t - I\right)^2 +$$

$$+\sigma^2 \left(XX^t + \lambda Id\right)^{-1} XX^t \left(\left(XX^t + \lambda Id\right)^{-1}\right)$$

ועתה:

$$\frac{d}{d\lambda}\hat{\mathrm{w}}\hat{\mathrm{w}}^t \left(\left(XX^t + \lambda Id\right)^{-1} XX^t - I\right)^2$$

$$= -2\hat{\mathrm{w}}\hat{\mathrm{w}}^t \left(XX^t + \lambda Id\right)^{-1} \frac{d}{d\lambda}\left(-XX^t + \lambda Id\right)\left(XX^t + \lambda Id\right)^{-1} \left(\left(XX^t + \lambda Id\right)^{-1} XX^t - I\right) =$$

$$= -2\hat{\mathrm{w}}\hat{\mathrm{w}}^t \left(XX^t + \lambda Id\right)^{-2} \left(\left(XX^t + \lambda Id\right)^{-1} XX^t - I\right) =$$

ולכן:

$$\frac{d}{d\lambda}\text{bias}^2\left(\lambda\right)|_{\lambda=0}=$$

$$=-2\hat{\text{w}}\hat{\text{w}}^t\left(XX^t\right)^{-2}\left(\left(XX^t\right)^{-1}XX^t-I\right)=$$

$$=-2\hat{\text{w}}\hat{\text{w}}^t\left(XX^t\right)^{-2}\left(I-I\right)=0$$

ומצד שני:

$$\left(\frac{d}{d\lambda}\sigma^2\left(XX^t+\lambda Id\right)^{-1}XX^t\left(\left(XX^t+\lambda Id\right)^{-1}\right)\right)|_{\lambda=0}=$$

$$=-\sigma^2\left(XX^t+\lambda Id\right)^{-1}\frac{d}{d\lambda}\left[\sigma^2\left(XX^t+\lambda Id\right)\right]\sigma^2\left(XX^t+\lambda Id\right)^{-1}XX^t\left(\left(XX^t+\lambda Id\right)^{-1}\right)-$$

$$-\sigma^2\left(XX^t+\lambda Id\right)^{-1}XX^t\left(\left(XX^t+\lambda Id\right)^{-1}\right)\frac{d}{d\lambda}\left[\left(XX^t\right)^{-1}\left(XX^t+\lambda Id\right)\right]XX^t\left(\left(XX^t+\lambda Id\right)^{-1}\right)=$$

$$=-\sigma^2\left(XX^t+\lambda Id\right)^{-1}\sigma^2\sigma^2\left(XX^t+\lambda Id\right)^{-1}XX^t\left(XX^t+\lambda Id\right)^{-1}-$$

$$-\sigma^2\left(XX^t+\lambda Id\right)^{-1}XX^t\left(XX^t+\lambda Id\right)^{-1}\left(XX^t\right)^{-1}XX^t\left(XX^t+\lambda Id\right)^{-1}=$$

$$=-\sigma^6\left(XX^t+\lambda Id\right)^{-2}XX^t\left(XX^t+\lambda Id\right)^{-1}-$$

$$-\sigma^2\left(XX^t+\lambda Id\right)^{-1}XX^t\left(XX^t+\lambda Id\right)^{-2}$$

ולכן:

$$\frac{d}{d\lambda}\text{Var}\left(\lambda\right)|_{\lambda=0}=$$

$$=-\sigma^6\left(XX^t\right)^{-2}XX^t\left(XX^t\right)^{-1}-$$

$$-\sigma^2\left(XX^t\right)^{-1}XX^t\left(XX^t\right)^{-2}=$$

$$= -\sigma^6 \left(XX^t\right)^{-2} - \sigma^2 \left(XX^t\right)^{-2} =$$

$$= -\sigma^6 \left(\left(XX^t\right)^{-1}\right)^2 - \sigma^2 \left(\left(XX^t\right)^{-1}\right)^2$$

ובסה"כ קיבלנו:

$$\frac{d}{d\lambda}\text{MSE}\left(\lambda\right)|_{\lambda=0} = \frac{d}{d\lambda}\text{bias}^2\left(\lambda\right)|_{\lambda=0} + \frac{d}{d\lambda}\text{Var}\left(\lambda\right)|_{\lambda=0} =$$

$$= \frac{d}{d\lambda}\text{Var}\left(\lambda\right)|_{\lambda=0} = -\sigma^6 \left(\left(XX^t\right)^{-1}\right)^2 - \sigma^2 \left(\left(XX^t\right)^{-1}\right)^2 < 0$$

כנדרש.

ה. צ.להסיק שאם המודל הלינארי הוא נכון, אז מעט רגיורליזציה של $Ridge$, מסייעת להורדת ה־ MSE. הוכחה:

מהסעיף הקודם מתקבל שבסביבת $\lambda = 0$, פונקציית ה־ MSE יורדת, ומכאן שיש λ כך ש־ $MSE\left(\lambda\right) < MSE\left(0\right)$.

חלק II

חלק מעשי

4 k-Fold Cross Validation על התאמת פולינום

יהיו $\mathcal{X}, \mathcal{Y} \subseteq \mathbb{R}$. נניח שהקשר בין $\mathcal{X}$ ו־ $\mathcal{Y}$ ניתן לתיאור על ידי פולינום מגדרה $d \in [15]$.

לכל $d \in [15]$, נסמן ב־ $\mathcal{H}_d$ את מחלקת הפולינומים מדרגה d.

המשימה היא לאמן כל אחת מהמחלקות על קבוצת האימון ולבצע ואלידציה על 15 ההיפותיזות שהתקבלו, בשביל לבחור את הפלט הסופי.

לבסוף, נבחן את הביצועים של הפרדיקטור שהתקבל, על קבוצת המבחן.

א. צ.לייצר דאטא באופן הבא:

i. ניקח $\mathcal{X} = [-3.2, 2.2]$ ונדגום מתוכו בהתפלגות אחידה.

ii. נקבע את הקשר בין y ו־ x על ידי:

$$y\left(x\right) = f\left(x\right) + \epsilon$$

כאשר:

$$f\left(x\right) = \prod_{i=-2}^{3}\left(x+i\right)$$

ו־ $\epsilon \sim \mathcal{N}\left(0, \sigma^2\right)$.

iii. ניקח $\sigma = 1$ ונייצר $m = 1,500$ דגימות.

iv. נחלק את m הדגימות לשתי קבוצות: $1,000$ דגימות עבור קבוצת האימון וקבוצת הולידציה, נסמנה D, ו־ 500 עבור קבוצת המבחן, נסמנה T.

ב. נחלק את D לשתי קבוצות של 500 דגימות כל אחת: קבוצה אחת עבור קבוצת האימון - S וקבוצה שנייה עבור קבוצת הואלידציה - V.

נאמן כל אחת מהמחלקות באמצעות S בשביל לקבל היפותיזה h_d, לכל $d \in [15]$, שממזערת את ה־ $loss$ על פני S.

ג. צ.לבצע ואלידציה בעבור $\{h_i\}_1^{15}$ בשביל לקבל h^* שממזערת את ה־ $loss$ על פני קבוצת הואלידציה, V:

תוצאה:

```
The best degree with two folds for polynomial regression on the given data is:  7
```

כלומר, הרעש משפיע והחלוקה של קבוצת האימון לשני חלקים אינה מספיקה על מנת להתגבר עליו. אכן התוצאה הטובה ביותר מתקבלת עבור פולינום ממעלה שבע בעוד שהפולינום המקורי הוא ממעלה חמש.

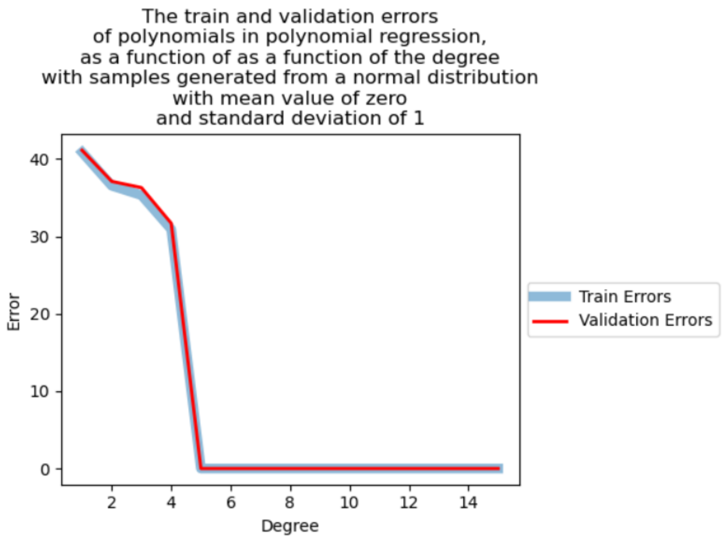
ד. סעיפים א עד ג הם k-fold cross validation עם $k = 2$.

צ.לבצע k-fold cross validation אבל הפעם עם $k = 5$ על הקבוצה D.

ה. צ.לשרטט את השגיאות על פני קבוצת האימון וקבוצת הואלידציה, כממוצע על פני k־החלקים, של הפולינומים מדרגות $d \in [15]$, שכפונקציה של d. צ.לבדוק עבור איזו דרגה מתקבלת השגיאה הנמוכה ביותר, נסמנה d^*.

פיתרון:

הגרף להלן:




```
The best degree with cross validation for polynomial regression on the given data is:  5
```

כלומר התוצאה הטובה ביותר היא עבור פולינום מדרגה 5. כמו כן יש הבדל מזערי בין השגיאה על קבוצת המבחן ובין השגיאה על קבוצת הואלידציה, קרי יש מעט מאוד $overfit$, כלומר למדנו מתוך מידע מספיק גדול.

ו. צ.לבצע $ERM_{\mathcal{H}_{d^*}}$ על הקבוצה D: למצוא פולינום מדרגה d^* בעל שגיאה מינימלית על הדגימות הללו, נסמנו ב־ h^*.

ז. צ.לבחון את הביצועים של h^* על קבוצת המבחן T: לחשב את השגיאה של h^* על פני T. צ.לבדוק האם השגיאה שונה בהרבה מהשגיאה שמצאנו בסעיף הקודם.

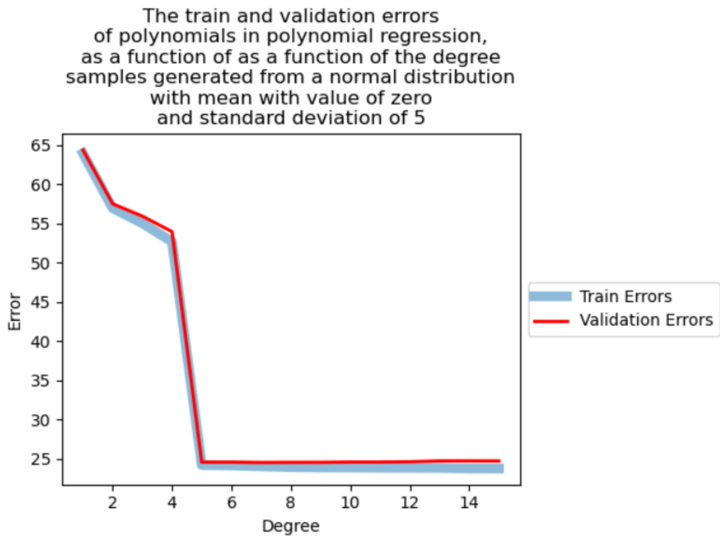
פיתרון:

```
The train error is: 0.9813278322975978
The test error is: 0.9726685025411308
The difference between the train error and the test error is: 0.008659329756466994
```

כפי שניתן לראות, ההבדל הוא מזערי, כלומר ה־ $overfit$ הינו קטן ביותר, כלומר ה־ Cross Validation עושה את העבודה.

ח. צ.לחזור על השלבים הקודמים עבור $\sigma = 5$ ולבדוק מה השתנה.

פיתרון:




```
The best degree with two folds for polynomial regression on the given data is:  6

The best degree with cross validation for polynomial regression on the given data is:  7
The train error is: 24.09464342161084
The test error is: 21.74461657093929
The difference between the train error and the test error is: 2.3500268506715507
```

הפעם הרעש יותר משמעותי, ולכן השגיאה גדלה, וכן ה־ Cross Validation לא מצליח להתגבר עליו.

5 k-fold ורגיולריזציה

בשאלה זו נערוך השוואה בין רגיולזריזציות $Lasso$ ו־ $Ridge$. נשתמש בדאטא שמעריך את מידת הגלוקוז בדם של מטופלי סוכרת.

א. נטען את הדאטא.

ב. רגיולריזציה היא שימושית כאשר קבוצת האימון קטנה. אז ניצור את קבוצת האימון להיות $m = 50$ הדגימות הראשונות, והשאר יהוו את קבוצת המבחן.

ג. צ.לבצע את הבאים פעמיים: פעם אחת עבור $Ridge$ ופעם שנייהה עבור $Lasso$:

i. צ.לערוך k-fold cross-validation מעל קבוצת האימון עם $k = 5$, תוך שימוש בערכים שונים עבור פרמטר הרגיולריזציה λ: צ.לחקור את הטווח של הערכים האפשריים.

ii. צ.לציין איזה ערכים של λ בחרתי לבדיקה על פני קבוצת הואלידציה, ולהסביר מדוע.

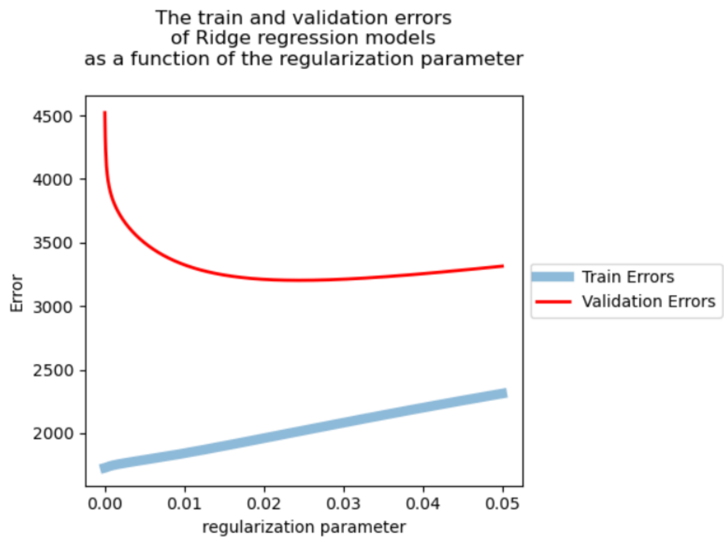
תשובה:

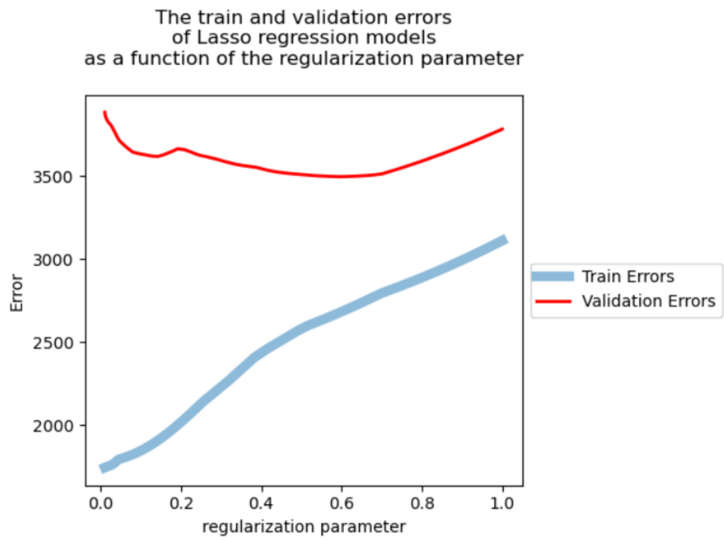
עבור $Ridge$ בחרתי בערכים בין 0 ל־ 0.05 על מנת שיהיה ניתן לראות במדוייק את האזור שבו השגיאה על ה־ Validation Set עובר מירידה לעלייה, שם פרמטר הרגיורליזציה אופטימלי (באזור ה־ 0.025).

עבור $Lasso$ בחרתי בערכים בין 0.01 ל־ 1, ראשית על מנת שיהיה ניתן לראות שערכים שקרובים מדי לאפס אינם מטיבים עמו ושנית על מנת שיהיה ניתן לראות במדוייק את האזור שבו השגיאה על ה־ Validation Set עובר מירידה לעלייה, שם פרמטר הרגיורליזציה אופטימלי (באזור ה־ 0.6).

ד. צ.לשרטט את השגיאות על פני קבוצות האימון והואלידציה (ממוצע השגיאות על פני k־החלקים) כפונקציה של λ, על פני הטווח שבחרתי.

פיתרון:





ה. צ.למצוא את ה־ λ הטובה ביותר עבור $Ridge$ וכנ"ל עבור $Lasso$.

תשובה:

```
Best regularization parameter for Ridge:  0.024424424424424426
Best regularization parameter for Lasso:  0.5966666666666667
```

ו. צ.לחשב את השגיאה של ההיפותיזות הבאות, על פני קבוצת המבחן:

i. ההיפותיזה הטובה ביותר של $Ridge$.

ii. ההיפותיזה הטובה ביותר של $Lasso$.

iii. רגרסיה לינארית.

תשובה:

```
The test error for the best Ridge model is: 3245.464541008676
The test error for a linear regression model is: 3612.249688324901
The test error for the best Lasso model is: 3640.7959659482776
```

ז. צ.לבדוק לאיזו היפותיזה יש את השגיאה הקטנה ביותר, ולבדוק האם הרגיולריזציה עזרה לשפר את התוצאות על פני קבוצת המבחן, בהשוואה לתוצאות שהתקבלו ללא רגיולריזציה על אותו הדאטא.

תשובה:

ל- $Ridge$ יש את השגיאה הקטנה ביותר, כלומר הרגיולריזציה עזרה לשפר את התוצאות של פני קבוצת המבחן במקרה של $Ridge$, אך במקרה של $Lasso$ השגיאה גדלה, קרי הרגילריזציה לא תרמה במקרה של $Lasso$.

2 ex5.py

```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from matplotlib import pyplot as plt
from sklearn import datasets
from sklearn.linear_model import Ridge, Lasso
from sklearn.linear_model import LinearRegression


LOW = -3.2
HIGH = 2.2
SIZE = 1500
TRAIN_FACTOR = 1000
MAX_DEGREE = 15
FOLDS_AMOUNT = 5
DEGS = range(1, MAX_DEGREE + 1)
FIRST_ITERATION_SIGMA = 1
SECOND_ITERATION_SIGMA = 5
RIDGE_PARAMS = np.linspace(0, 0.05, 1000)
LASSO_PARAMS = np.linspace(0.01, 1, 1000)


def default_labels_func():
    """
    The labels function for the samples of question four
    :param sigma: The standard deviation of which to create noise from
    :return: The function
    """
    return lambda x: (x + 3) * (x + 2) * (x + 1) * (x - 1) * (x - 2)



def generate_samples(low, high, size):
    """
    Generates samples uniformly from an interval
    :param low: The lower bound of the interval
    :param high: The upper bound of the interval
    :param size: The amount of samples to generate
    :return: The samples
    """
    return np.random.uniform(low, high, size)


def generate_labels(f, all_samples):
    """
    Generates the labels of all the samples that were already generated
    :param f: The label function
    :param all_samples: The samples to get their labels
    :return: The labels
    """
    return np.apply_along_axis(f, 0, all_samples)


def add_noise(labels, sigma, mean=0):
    return labels + np.random.normal(mean, sigma, len(labels))


def generate_data(sigma):
    """
```

```python
    Generates the data for question four
    :param sigma: The standard deviation for the noise of the labels
    :return: Train samples, train labels, test samples, test labels
    """
    X = generate_samples(LOW, HIGH, SIZE)
    y = generate_labels(default_labels_func(), X)
    y = add_noise(y, sigma)
    D, T = split_data(X, y, TRAIN_FACTOR)
    all_train_samples = D[0]
    all_train_samples = all_train_samples.reshape(-1, 1)
    all_train_labels = D[1]
    test_samples = T[0]
    test_samples = test_samples.reshape(-1, 1)
    test_labels = T[1]
    return all_train_samples, all_train_labels, test_samples, test_labels


def split_data(all_samples, all_labels, train_factor):
    """
    Splits the data into train set and test set
    :param all_samples: The samples to split
    :param all_labels: The labels to split
    :param train_factor: A number which specifies how many samples should
    the train set contain
    :return: The train set and test set
    """
    train_set = all_samples[:train_factor], all_labels[:train_factor]
    test_set = all_samples[train_factor:], all_labels[train_factor:]
    return train_set, test_set


def get_k_folds(k, train_samples, train_labels):
    """
    Splits the train samples into folds
    :param k: The amount of folds to split the data into
    :param train_samples: The samples to split
    :param train_labels: The labels to split
    :return: A list of the folds of the samples, where each fold should play
    the role of the validation set, and also returns a list of the samples
    where the i'th item consists of all the samples but the i'th fold,
    which should play the role of a train set that matches the i'th fold as
    the validation set"""
    total_amount = len(train_samples)
    amount_per_fold = total_amount // k
    validation_excluded = []
    folds = []

    for i in range(1, k + 1):
        a = train_samples[:(i - 1) * amount_per_fold]
        b = train_samples[i * amount_per_fold:]
        curr_samples = np.concatenate((a, b))
        a = train_labels[:(i - 1) * amount_per_fold]
        b = train_labels[i * amount_per_fold:]
        curr_labels = np.concatenate((a, b))
        validation_excluded.append((curr_samples, curr_labels))

    counter = 0
    for i in range(k):
        fold_samples = train_samples[counter:counter + amount_per_fold]
        fold_labels = train_labels[counter:counter + amount_per_fold]
        folds.append((fold_samples, fold_labels))
        counter += amount_per_fold

    return folds, validation_excluded


def two_folds_get_single_model(model, train_samples, train_labels,
                               validation_samples, validation_labels):
```

```
    """
    Gets a single model for the two folds version
    :param model: The model to train
    :param train_samples: Train samples
    :param train_labels: Train labels
    :param validation_samples: Validation samples
    :param validation_labels: Validation labels
    :return: The trained model, its train error and its validation error
    """
    model.fit(train_samples, train_labels)
    y_hat_train = model.predict(train_samples)
    train_error = np.mean((train_labels - y_hat_train) ** 2)
    y_hat_validation = model.predict(validation_samples)
    validation_error = np.mean((validation_labels - y_hat_validation) ** 2)

    return model, train_error, validation_error


def poly_generator(deg):
    """
    A generator for a polynomial regression learner
    :param deg: The degree of the polynomial to generate
    :return: The generated learner
    """
    return make_pipeline(PolynomialFeatures(deg), LinearRegression())


def two_folds_get_all_models(model_generator, params, train_samples,
                             train_labels, validation_samples,
                             validation_labels):
    """
    Gets all the models of the two folds case
    :param model_generator: The model generator
    :param params: The parameters from which to generate the learners
    :param train_samples: Train samples
    :param train_labels: Train labels
    :param validation_samples: Validation samples
    :param validation_labels: Validation labels
    :return: The models
    """
    models = []
    for param in params:
        model = model_generator(param)
        model = two_folds_get_single_model(
            model, train_samples, train_labels, validation_samples,
            validation_labels)
        models.append(model)
    return models


def multiple_folds_get_single_model(model, folds, validation_excluded):
    """
    Gets a single model from the multiple folds case
    :param model: The model to train
    :param folds: The validation sets
    :param validation_excluded: The train sets
    :return: The trained model, its mean train error and its mean validation
    error
    """
    folds_amount = len(folds)
    train_losses = []
    validation_losses = []
    for i in range(folds_amount):
        train_samples = validation_excluded[i][0]
        train_labels = validation_excluded[i][1]
        model.fit(train_samples, train_labels)

        y_hat = model.predict(train_samples)
```

```
        y = train_labels
        train_losses.append(np.mean((y - y_hat) ** 2))

        fold_samples = folds[i][0]
        y_hat = model.predict(fold_samples)
        y = folds[i][1]
        validation_losses.append(np.mean((y - y_hat) ** 2))

    train_error = np.mean(train_losses)
    validation_error = np.mean(validation_losses)

    return model, train_error, validation_error


def multiple_folds_get_all_models(model_generator, params, folds,
                                  validation_excluded):
    """
    Gets all the models for the multiple folds case
    :param model_generator: The model generator
    :param params: The parameters from which to generate the models
    :param folds: The validation sets
    :param validation_excluded: The train sets
    :return: The trained models as triplets, each one consists of the model,
    its mean train error and its mean validation error
    """
    models = []
    for param in params:
        model = model_generator(param)
        poly = multiple_folds_get_single_model(model, folds,
                                               validation_excluded)
        models.append(poly)
    return models


def get_best_model(models):
    """
    Gets the best model (triplet)
    :param models: The models to choose from
    :return: The best model (triplet)
    """
    return min(models, key=lambda poly: poly[2])


def get_errors(models):
    """
    Gets a list of the mean train errors and a list of the mean validation
    errors of the given models
    :param models: The models to get their errors
    :return: The list of train errors and the list of validation errors
    """
    train_errors = []
    validations_errors = []
    for model in models:
        train_errors.append(model[1])
        validations_errors.append(model[2])
    return train_errors, validations_errors


def get_multiple_folds_models(model_generator, params,
                              all_train_samples,
                              all_train_labels):
    """
    Gets the models for the multiple folds case
    :param model_generator: The model generator
    :param params: The parameters from which to generate the models
    :param all_train_samples: Train samples
    :param all_train_labels: Train labels
    :return: The models
```

```python
    """
    S, V = get_k_folds(FOLDS_AMOUNT, all_train_samples, all_train_labels)
    return multiple_folds_get_all_models(model_generator, params, S, V)


class Data:
    """
    Data object for question four
    """
    def __init__(self, sigma):
        """
        Initializes a data object for question four according to the noise
         of the labels
        :param sigma: The noise to give to the labels
        """
        self.all_train_samples, self.all_train_labels, self.test_samples, \
        self.test_labels = generate_data(sigma)


def fourth_question_items_b_c(all_train_samples, all_train_labels,
                              param_getter, param_description,
                              regression_description):
    """
    Runs a general version of items b and c of question four
    :param all_train_samples: Train samples
    :param all_train_labels: Train labels
    :param param_getter: Gets the parameter from the model
    :param param_description: Description of the parameter
    :param regression_description: Description of the regression
    :return: None
    """
    S, V = get_k_folds(2, all_train_samples, all_train_labels)
    S, V = S[0], V[0]
    train_samples = S[0]
    train_labels = S[1]
    validation_samples = V[0]
    validation_labels = V[1]
    models = two_folds_get_all_models(poly_generator, DEGS, train_samples,
                                      train_labels, validation_samples,
                                      validation_labels)
    best_model = get_best_model(models)
    best_param = param_getter(best_model)
    print()
    print('  The best {} with two folds for {} is: '.
          format(param_description, regression_description), best_param)


def fourth_question_items_d_e(model_generator, params, all_train_samples,
                              all_train_labels, xlabel, title1, title2,
                              title3, param_getter, param_description,
                              regression_description):
    """
    Runs a general version of items d and e of question four
    :param model_generator: The model generator
    :param params: The parameters from which to generate the models
    :param all_train_samples: Train samples
    :param all_train_labels: Train labels
    :param xlabel: The labels for the X axis
    :param title1: One part of the title for the graph
    :param title2: Second part of the title for the graph
    :param title3: Third part of the title for the graph
    :param param_getter: Gets the parameter from the model
    :param param_description: Description of the parameter
    :param regression_description: Description of the regression
    :return: None
    """
    models = get_multiple_folds_models(model_generator, params,
                                       all_train_samples,
```

```
                                        all_train_labels)

    best_model = get_best_model(models)
    best_param = param_getter(best_model)
    print()
    print('  The best {} with cross validation for {} is: '.
          format(param_description, regression_description), best_param)

    train_errors, validation_errors = get_errors(models)

    plt.plot(params, train_errors, linewidth=6, alpha=0.5, label='Train '
                                                                 'Errors')
    plt.plot(params, validation_errors, 'r-', linewidth=2,
             label='Validation Errors')
    plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
    plt.xlabel(xlabel)
    plt.ylabel('Error')
    plt.title('The train and validation errors\nof {}\nas a function '
              'of {}\n{}'.format(title1, title2, title3))
    plt.show()


def fourth_question_items_f_g(model_generator, all_train_samples,
                              all_train_labels,
                              test_samples, test_labels, best_param):
    """
    Runs a general version of items f and g of question four
    :param model_generator: The model generator
    :param all_train_samples: Train samples
    :param all_train_labels: Train labels
    :param test_samples: Test samples
    :param test_labels: Test labels
    :param best_param: The parameter of the best model
    :return: None
    """
    model = model_generator(best_param)
    model.fit(all_train_samples, all_train_labels)

    y_hat = model.predict(all_train_samples)
    y = all_train_labels
    train_error = np.mean((y - y_hat) ** 2)
    y_hat = model.predict(test_samples)
    y = test_labels

    test_error = np.mean((y - y_hat) ** 2)
    diff = np.fabs(train_error - test_error)

    print('  The train error is: {}'.format(train_error))
    print('  The test error is: {}'.format(test_error))
    print('  The difference between the train error and the test error is: '
          '{}'.format(diff))


def run_all_items(model_generator, params, xlabel, title1, title2, title3,
                  best_param, all_train_samples, all_train_labels,
                  test_samples, test_labels, param_getter, param_description,
                  regression_description):
    """
    Runs a general version of all items from question four
    :param model_generator: The model generator
    :param params: The parameters from which to generate the models
    :param xlabel: The label of the X axis
    :param title1: One part of the title for the graph
    :param title2: Second part of the title for the graph
    :param title3: Third part of the title for the graph
    :param best_param: The parameter of the best model
    :param all_train_samples: Train samples
    :param all_train_labels: Train labels
```

```python
    :param test_samples: Test samples
    :param test_labels: Test labels
    :return: None
    """
    fourth_question_items_b_c(all_train_samples, all_train_labels,
                              param_getter, param_description,
                              regression_description)
    fourth_question_items_d_e(model_generator, params, all_train_samples,
                              all_train_labels, xlabel, title1, title2,
                            title3, param_getter, param_description,
                              regression_description)
    fourth_question_items_f_g(model_generator, all_train_samples,
                              all_train_labels,
                              test_samples, test_labels, best_param)


def get_deg_from_poly_model(poly):
    """
    Gets the degree of a given polynomial regression learner
    :param poly: A triplet of a polynomial regression learner
    :return: The degree of the polynomial regression learner
    """
    return poly[0].named_steps['polynomialfeatures'].degree


def run_question_four(sigma):
    data = Data(sigma)

    all_train_samples, all_train_labels, test_samples, test_labels = \
        data.all_train_samples, data.all_train_labels, data.test_samples, \
        data.test_labels

    models = get_multiple_folds_models(poly_generator, DEGS,
                                       all_train_samples,
                                       all_train_labels)

    best_deg = get_deg_from_poly_model(get_best_model(models))

    run_all_items(poly_generator, DEGS, 'Degree',
                  'polynomials in polynomial regression,',
                  'as a function of the degree',
                  'samples generated from a normal distribution\nwith mean '
                  'with value of zero\nand standard deviation of {}'.format(
                      sigma),
                  best_deg, all_train_samples, all_train_labels,
                  test_samples, test_labels, get_deg_from_poly_model,
                  'degree', 'polynomial regression on the given data')


def get_diabetes_data():
    X, y = datasets.load_diabetes(return_X_y=True)
    train_samples = X[:50]
    train_labels = y[:50]
    test_samples = X[50:]
    test_labels = y[50:]
    return train_samples, train_labels, test_samples, test_labels


def ridge_generator(alpha):
    """
    A generator for a Ridge regression learner
    :param alpha: The regularization parameter of the learner we want to
    generate
    :return: The generated learner
    """
    return Ridge(alpha)
```

```python
def lasso_generator(alpha):
    """
    A generator for a Lasso regression Learner
    :param alpha: The regularization parameter of the learner we want to
    generate
    :return: The generated learner
    """
    return Lasso(alpha, max_iter=10**4)


def get_best_reg_param(model):
    return model[0].alpha


def run_question_five():
    """
    Run question five
    :return: None
    """
    train_samples, train_labels, test_samples, test_labels = get_diabetes_data()

    models = get_multiple_folds_models(ridge_generator, RIDGE_PARAMS,
                                       train_samples,
                                       train_labels)
    ridge_best_param = get_best_reg_param(get_best_model(models))
    ridge_best_model = ridge_generator(ridge_best_param)
    ridge_best_model.fit(train_samples, train_labels)
    ridge_y_hat = ridge_best_model.predict(test_samples)
    test_error = np.mean((test_labels - ridge_y_hat) ** 2)
    print()
    print('  The test error for the best Ridge model is: {}'.format(test_error))

    linear_regression_model = ridge_generator(0)
    linear_regression_model.fit(train_samples, train_labels)
    linear_regression_model_y_hat = linear_regression_model.predict(test_samples)
    test_error = np.mean((test_labels - linear_regression_model_y_hat) ** 2)
    print('  The test error for a linear regression model is: {}'.format(
        test_error))

    models = get_multiple_folds_models(lasso_generator, LASSO_PARAMS,
                                       train_samples,
                                       train_labels)
    lasso_best_param = get_best_reg_param(get_best_model(models))
    lasso_best_model = lasso_generator(lasso_best_param)
    lasso_best_model.fit(train_samples, train_labels)
    lasso_y_hat = lasso_best_model.predict(test_samples)
    test_error = np.mean((test_labels - lasso_y_hat) ** 2)

    print('  The test error for the best Lasso model is: {}'.format(test_error))
    fourth_question_items_d_e(ridge_generator, RIDGE_PARAMS, train_samples,
                              train_labels,
                              'regularization parameter',
                              'Ridge regression models',
                            'the regularization parameter', '',
                              get_best_reg_param, 'regularization parameter',
                              'Ridge regression on the given data')

    fourth_question_items_d_e(lasso_generator, LASSO_PARAMS, train_samples,
                              train_labels,
                              'regularization parameter',
                              'Lasso regression models',
                            'the regularization parameter', '',
                              get_best_reg_param, 'regularization parameter',
                              'Lasso regression on the given data')


# run_question_four(FIRST_ITERATION_SIGMA)
# run_question_four(SECOND_ITERATION_SIGMA)
```

```
# run_question_five()
```