Übungsblatt 4

Aufgabe 1 – 20%

Wie hängen Durchschnitt, Median, l_2-Loss und l_1-Loss zusammen? Beweisen/begründen Sie Ihre Aussage.

Aufgabe 2 – 20%

In der Vorlesung haben wir ein zentrales Theorem zu Konvexität kennengelernt, das besagt, dass jeder kritische Punkt einer konvexen Funktion gleichzeitig ein *globales Minimum* ist. Es gibt also keine lokalen Minima in konvexen Funktionen.
Zeigen Sie, dass dieses Theorem wahr sein muss. Für die Aufgabenstellung genügt eine grafische Darstellung und eine Erklärung.
Optional: Beweisen Sie mathemathisch, dass das Theorem stimmt. Es gibt hier mehrere Wege. Ein Weg: zuerst zeigen, dass für konvexe f gilt:

Lemma. *Für alle $x, y \in \mathbb{R}^d$ gilt $f(y) \geq f(x) + \nabla f(x)^T (y - x)$.*

...und anschließend das Theorem mit Hilfe des Lemmas beweisen.

Aufgabe 3 – 10%

Gegeben ist $f(x)$, eine reellwertige Funktion für $x \in \mathbb{R}$. Das Gradientenverfahren macht Update-Schritte nach folgenden Schema:

$$x^{(i+1)} = x^{(i)} - \gamma \nabla f(x^{(i)}), \tag{1}$$

wobei $\gamma > 0$ die Lernrate ist.
Angenommen $f(x) = |x|$, $x^{(0)} = x_0 \in \mathbb{R} \setminus 0$ und $\gamma = 2x_0$. Berechnen Sie die ersten 3 Iterationen. Sie werden feststellen, dass es ein Problem bei der Minimierung von $f(x)$ geben wird. Erklären Sie das Problem und schlagen Sie vor wie man dies umgehen könnte.

Aufgabe 4 – 50%

In der Vorlesung haben Sie die Methode der Regularisierung zur Kontrolle der Komplexität eines Modells kennengelernt. Für die beiden Datensätze `regularization_dataset0.csv` und `regularization_dataset1.csv` sollen Sie nun jeweils einen Regularisierungspfad für ein polynomiales Modell mit einem Polynomgrad von 6 berechnen, dass über L2-Norm regularisiert ist. Validierungs- und Trainingserror der verschiedenen Modelle sollen mit 10-facher Kreuzvalidierung berechnet werden. Variieren Sie zur Berechung des Pfades den Regularisierungsparameter α im Intervall $[10^{-12}, 10^{6}]$. Visualisieren Sie den Regularisierungspfad indem Sie die Validierungs- und Trainingserror der Modelle in Abhängigkeit von α plotten.
Sie dürfen für diese Aufgabe `scikit-learn` verwenden. Die Implementation des linearen Regressors in `scikit-learn` erlaubt Regularisierung. Ein Template, dass Ihnen bei der Implementation helfen kann finden Sie im Jupyter Notebook `regularization.ipynb`.

Die Datensätze `robust_regression_dataset0` und `robust_regression_dataset1` liegen jeweils in einer Version mit starken Ausreißern und ohne Ausreißer vor. Trainieren Sie die beiden linearen Regressoren `LinearRegressor` und `HuberRegressor` der `scikit-learn` Bibliothek auf allen vier Datensätzen. Was fällt Ihnen auf, wenn Sie die trainierten Modelle vergleichen? Visualisieren Sie die trainierten Regressoren jeweils mit Ihren zugehörigen Trainingsdaten und stellen Sie `HuberRegressor` und `LinearRegressor` gegenüber.
Sie dürfen für diese Aufgabe `scikit-learn` verwenden. Ein Template, dass Ihnen helfen kann finden Sie im Jupyter Notebook `robust_regression.ipynb`. Als besondere Herausforderung können Sie auch den linearen Regressor sowie den Huber-Regressor selber mit `numpy` implementieren. Der Huber-Regressor unterscheidet sich vom linearen Regressor durch die Huber loss function.

Bitte lösen Sie die Fragen und die Programmieraufgaben bis zum **20. November 2023**. Ihren Python-Code, sowie Visualisierungen können Sie in Form eines Jupyter Notebooks oder PDFs hochladen.