# „Masterarbeitstagebuch“

19.04.2017:

* Noll pos def weiter
* Gespräch mit Lukas:
  + Mail an Andre:  
    Liste mit Möglichkeiten, die noch machbar wären 🡪 was davon tun?  
    nochmal fragen, wie ausführlich conv-inex  
    Anwendung 🡪 soll ich noch mal größere Datensätze nehmen? Mehr Parameter?  
    andere Zielfunktion in Anwendung?
  + Ableitung noch mal plotten in \lambda-Plot
  + Möglichkeit bei mehr Parametern: eigenen Datensatz duplizieren und noise drauf
  + Möglichkeiten was noch tun:
    - Anwendung: mehr Parameter
    - Größere Datensätze
    - Conv-inex
    - Asymptotische Verbesserung des Fehlers
    - Besseres BFGS-Update durch bessere Subgradienten
    - Bedingung auf eta, damit beschränkt 🡪 Kommentar über Noll

18.04.2017:

* Testing-by-doing Fehler im Algorithmus funden
* Vergleich mit Hare-Version auf f1-f5 Testfunktionen
* Sicherstellen, dass Q+1/t\*I pos semi def

14.04.-15.04.2017:

* Programmierung „Noll-Version“ des Hare-Bundle-Algortihmus

13.04.2017:

* Aufschreiben SVM-Kapitel
* Noll Paper lesen, zusammenfassenn

06.04.-12.04.2017:

* Einlesen in SVM
* Aufschreiben SVM-Kapitel
* Einlesen Noll

Um 01.04.2017:

* Ableitungen getestet

24.03.2017:

* Fehlerfunktionen getestet 🡪 stimmen

23.03.2017:

* LTH-Übung zur Fehlersuche herangezogen
* W, b Berechnungen überprüft 🡪 stimmt jetzt (!Wie nehme ich Formeln 🡪 für meine Berechnungen Wb anstatt bW und +b anstatt –b)
* Misclassification loss überprüft

22.03.2017:

* Großes Skript zur Fehlererkennung / Vergleichbarkeit geschrieben

20.03.2017:

* 1 Seite Latex geschrieben
* UCI-Datensätze standardisiert und gespeichert
* „Zählfunktion“ für Klassifizierungsfehler programmiert

21.03.2017:

* Fehler in w-Berechnung ausgebessert

13.10.2016:  
Debuggen; Test testen; Löschroutine für normalen Algo geändert, die „langsamen“ Stellen im Sparse-Algo lassen sich wahrscheinlich nicht beheben. 🡪 jetzige Überlegung: bei normalem Algo bleiben; Tests alle mit erster Abbruchbedingung ausgeführt;  
Erkenntnis: normal und sparse machen wirklich genau dasselbe 🡪 exakt selbe Algorithmen

26.10.2016:  
ToDo für heute:

* Einfache Tests für nonconv schreiben; dabei darauf achten, welche Größen positiv… etc. – done, noch schauen, ob sie auch durchlaufen
* Testfunktionen wie in Paper programmieren und benutzen (vielleicht eher morgen?)
* Die schon vorhandenen „as expected“ –Tests noch mal anschauen und dokumentieren - done
* Nicht vergessen zu prüfen, ob alles Ausgabewerte an der richtigen Stelle bevor Tests laufen lassen - done
* In den Test von conv-ex kann man noch mal einfügen, ob xi wirklich den Wert annimmt, den es soll (zweite Formulierung durch f-Werte)

In Tests:

* Wie negativ wird e?
* Wie groß wird eta? In nonconv-inex-Paper sehr wichtig, dass eta beschränkt bleibt; wird im Beweis angenommen, ist aber keine Annahme, die allgemein getroffen werden kann, bzw. die beweisbar gilt; prüfe also hier bereits, ob eta beschränkt bleibt
* Test ob Komponenten von alpha immer ≥ 0; Test ob alpha ungefähr = 1; Test wie klein alpha-Komponenten werden für eventuelle Bundle Kompression
* Test wie groß der Unterschied der beiden Formulierungen von delta ist;  
  Wenn hier ein großer Unterschied ist, noch mal genauer anschauen woran das liegen kann
* Test wie groß der Unterschied der xi-Formulierungen ist
* Test ob c negativ wird; sollte auf keinen Fall passieren
* Die verschiedenen Formulierungen getestet.

27.10.2016:

Tests durchlaufen lassen (für 1-Norm)

* Nur in 1D Lösung gefunden – vielleicht doch t anpassen?
* e nicht besonders negativ (1e-13), passt, da konvexe Funktion
* verschiedene Formulierungen von delta unterscheiden sich recht stark, korreliert mit Lösung des Algorithmus: da wo deltas sehr unterschiedlich, keine Lösung nach 1000 Schritten gefunden 🡪 herausfinden wo der Unterschied in den deltas herkommmt
* alpha nicht so gut, nur 1e-6 (Abstand zu 1)
* xi und C stimmen in den verschiedenen Formulierungen überein
* S ~ -1/t\*d nur 1e-5
* Andere Tests laufen durch

ToDo

* Tests zum Durchlaufen bringen
* In den Test von conv-ex kann man noch mal einfügen, ob xi wirklich den Wert annimmt, den es soll (zweite Formulierung durch f-Werte)
* herausfinden wo der Unterschied in den deltas herkommmt
* Testfunktionen wie in Paper programmieren und benutzen
* Noch mal prüfen, ob es sein kann, dass xi und C genau auf 0 getestet werden
* Herausfinden, wo der Unterschied zw. S und d herkommt
* Was hat die „Gleichheit der deltas“ damit zu tun, wie lange der Algo braucht?
* Warum wird xi mit steigender Schrittanzahl schlechter?
* Nächster Schritt: eigenen „quadProg“ programmieren
* Anschauen, wie quadProg und anderer Subproblemlöser im Paper abgeschnitten haben

Tests durchlaufen lassen (für 2-Norm)

* Lösung überall in Toleranz, trotzdem einmal nicht abgebrochen 🡪 dort wo delta schlecht war
* e sogar noch besser, 1e-14
* deltas besser als vorher, aber immer noch nur 1e-3; ein delta 0,… dort kein Abbruch
* alpha nun in 1e-6
* S zu d nun in 1e-6
* Xi und C passen
* Andere Tests laufen durch

Aufgefallen: C und xi passen genau; noch mal prüfen, ob das sein kann

* Fehler in xi, C – Tests gefunden, waren auf Wert 0 gesetzt und dann auf > getestet, klar kommt da nur 0 raus; Test war falschrum, muss mit 0 anfangen und auf <= testen 🡪 ausgebessert
* Modellfunktionsformulierung von xi auch noch in Test geschrieben

Neuer Test mit 2-Norm und gleichem delta wie davor (delta1)

* Alle Minima werden in ~< 300 Schritten berechnet
* e in 1e-14
* deltas fast alle gleich??? Warum auf einmal???, bei zweitem Test 1e-3
* c nie negativ, alpha nie negativ
* Alpha 1e-5 zu 1
* Unterschied xi\_1 bei 1e-2
* Unterschied xi\_2 viel!, bis zu 10,… Modellfunktion falsch?
* S, d 1e-5
* C 1e-17
* Zweiter Durchlauf liefert ähnliche Ergebnisse

Neuer Test mit 1-Norm und gleichem delta wie davor (delta1)

* Nur 1D kann gelöst werden (und ein paar Probleme von 2D)
* Deltas wieder an entsprechenden Stellen schlecht
* Alpha 1e-5
* Xi\_1 0,…; xi\_2 > 1
* S-d 1e-5
* C ok

Test nochmal (1-Norm) mit anderem delta (delta2)

* Abbruchbedingung erfüllt, alle k unter 1000, aber tol nicht eingehalten; wie kann das sein? (nur bis 1e-4)
* Deltas ähnlich, 1e-3
* Xi\_1: 1e-3; xi\_2: 1
* S-d: 1e-5
* Die obigen Werte scheinen sich für diesen Test eingependelt zu haben

Test für conv-Algorithmus angepasst 🡪 Vergleich zu nonconv:

* G und d scheinen grundsätzlich etwas näher beieinander zu liegen (aber nicht viel)
* Xi\_1,2 so wie bei nonconv 🡪 schauen, wo xi\_2 falsch 🡪 falsche Formel in mk

Schlussfolgerungen:

* 2. Delta passt nicht, herausfinden warum 🡪 Fehler ist der S-d-Fehler, dieser „vererbt“ sich auf delta2; in delta1 ist nur der C-Fehler (sehr gering, 1e-16) und der d-Fehler aus der Optimierung, der überall drin ist.
* xi am genauesten für mk-Formulierung, macht auch Sinn, da bei anderer Formulierung S-d-Fehler mit dazu kommt; aber trotzdem komisch, müsste eigentlich besser aus der Optimierung rauskommen, ist doch eine Nebenbedingung…
* Je ähnlicher die deltas, desto weniger Schritte braucht der Algorithmus; WARUM?  
  Unterschied im Bereich 1e-4 ok, danach schlecht
* xi wird mit steigender Schrittanzahl schlechter; WARUM?

28.10.2016:

ToDo:

* Tests zum Durchlaufen bringen
* Testfunktionen wie in Paper programmieren und benutzen
* Herausfinden, wo der Unterschied zw. S und d herkommt
* Was hat die „Gleichheit der deltas“ damit zu tun, wie lange der Algo braucht?
* Warum wird xi mit steigender Schrittanzahl schlechter?
* Nächster Schritt: eigenen „quadProg“ programmieren
* Anschauen, wie quadProg und anderer Subproblemlöser im Paper abgeschnitten haben
* Eckdaten für numerische Tests in Paper aufschreiben und mit meinen Daten vergleichen

Getan

* Testfunktionen und Ableitungen implementiert
* Angefangen Test für Ableitungen zu schreiben

31.10.2016:

ToDo: siehe 28.10.

Getan:

8.11.2016:

ToDo:

* Tests mit Testfunktionen? zum Durchlaufen bringen
* Testfunktionen wie in Paper programmieren und benutzen
* Herausfinden, wo der Unterschied zw. S und d herkommt
* Was hat die „Gleichheit der deltas“ damit zu tun, wie lange der Algo braucht?
* Warum wird xi mit steigender Schrittanzahl schlechter?
* Nächster Schritt: eigenen „quadProg“ programmieren
* Anschauen, wie quadProg und anderer Subproblemlöser im Paper abgeschnitten haben
* Eckdaten für numerische Tests in Paper aufschreiben und mit meinen Daten vergleichen
* Änderung für t/mu in extraalgo und mit ihren Ergebnissen vergleichen

Getan

* Alle Ableitungen von Testfunktionen jetzt richtig
* Parameterwahl für Tests in Paper:
  + X0 = (1,…1); M = 10, funeval = 300; tol = 10^-6; R = 10; m = 0.05; gamma = 2
* Nonconv ex zweite Version mit Update von t/mu gemacht

9.11.2016:

ToDo:

* Siehe 8.11.
* Algo-Ergebnisse mit denen vom Paper vergleichen…
* Test Subgradienten fertig programmieren und testen

Getan:

* programmiert, dass Subgradientenfunktioenen Spaltenvektoren zurückgeben und getestet, dass sowohl Zeilen- als auch Spaltenvektoren als Input gehen
* Subgradiententest funktioniert noch nicht, aber sehr komisch, weil nicht besser wenn eps kleiner, sondern sogar schlechter…

10.11.2016:

ToDo:

Getan:

* Version des Algorithmus mit Update fertig gestellt
* Versuch, den Algorithmus mit Ergebnissen aus dem Paper zu vergleichen
  + Funktioniert definitiv nicht ich >1 Dimension (
* Testfunktionen und deren Subgradienten jetzt richtig
* Hauptproblem wahrscheinlich das xi, dadurch delta falsch und deswegen deltas auch so unterschiedlich und dadurch schlechter Abbruch…  
  komisch nur, dass auch wenn xi = f\_hat – mk + … immer noch keine Lösung…
* Angefangen Nonconv inex Beweis abzuschreiben

11.11.2016:

Getan: