Arbeitsbericht

Projekt: LDA and SVM

Autor: Daniel Fridljand

Nach den Weihnachtsferien hatten wir unser erstes Treffen als Team um das weitere Vorgehen zu planen. Es wurde eine grobe Zeitplanung erstellt, ersten Ideen der Implementierung und Klassenstrukturen entwickelt, doch die wichtigste Entscheidung des Tages war wohl die Verteilung der einzelnen Tasks an die Gruppenmitglieder. Ich wurde mit den FDAs betraut. Ich habe zu dem Zeitpunkt bereits wie allen anderen auch, die für uns in der Projektbeschreibung als relevant gekennzeichneten Abschnitte der Literatur überflogen und wusste somit bereits, dass die FDAs einen im Vergleich übersichtlichen und scheinbar nachvollziehbaren Abschnitt bildeten.

Meine Einschätzung basierte darauf, dass von den paar den FDAs gewidmeten Seiten, ein beträchtlicher Anteil aus Graphiken und einem ausführlichen Beispiel bestand, und darüber hinaus relativ wenig auf Griechisch stand. Und von allen Dingen schloss der Abschnitt mit einem Unterkapitel „Computations“ ab. Selbstsicher, dass diese Aufgabe nicht viel Zeit in Anspruch nehmen würde, erwartete ich baldig bereits andere Tasks zuwenden zu können.

Doch leider stellte sich der Mangel an mathematischen Formeln und eben jene Kürze des Abschnittes als problematisch heraus: es ging aus den wenigen Zeilen kaum hervor, wie der Algorithmus funktioniert, geschweige denn zu implementieren sei. Und obwohl in allen Abschnitten Hintergrundwissen aus zuvorkommenden Kapiteln benötigt wurde, wurden hier Terminologien benutzt, die sich nirgends sonst im Buch finden.

Das zu erkennen hat einiges an Zeit gekostet, bis Christoph mir vorgeschlagen hat, mir stattdessen die Seiten nach QDA als Task zu nehmen, also Regularized Discriminant Analysis. Da dies im Wesentlichen eine Mischung der zu dem Zeitpunkt bereits von Niklas implementierten Algorithmen LDA und QDA war, konnte ich viel übernehmen. Hier traten für mich persönlich Schwierigkeiten damit auf, dass z.B. während ich an dem code gearbeitet habe die R6 Klasse „dataset“ eingeführt wurde und ich konstant meinen Code überarbeiten und auf neue Fehler eingehen musste.

Generell habe ich das Gefühl viel Zeit mit Fehleruntersuchung verbracht zu haben. Um ein Beispiel auszuführen. Ich habe wohl mindestens fünf Stunden damit verbracht die Fehlerursache von sich durch den Code pflanzenden Na und NaN Werte zu suchen. Die Ursache des Problems war, dass ich mit sehr kleinen Datensätzen getestet habe und die Kovarianzmatrix eines Datensatzes einer einzigen Beobachtung logischerweise NaN warf.

Das Problem bei „Regularized Discriminant Analysis“ ist, dass es von zwei Parametern abhängt, bei denen man eigentlich nicht erwarten kann, dass der Nutzer sie eingibt. Deswegen musste offensichtlich automatische nach den Parametern getuned werden, was ich durch cross validation realisiert habe. Auch hier stellte sich die Implementierung als wesentlich aufwendiger als erwartet heraus und war meine Hauptbeschäftigung.

Der große Vorteil von Cross Validation war, dass RDA dadurch automatisch immer mindestens genauso gut, wie LDA und QDA klassifiziert. Dies konnte an einigen Tests mit testthat und in der shinyapp bestätigt werden. RDA sollte somit auf normalverteilten Validationdata (also Datensätzen derselben Verteilungen, die noch nicht im Training verwendet wurden) die besten Klassifizierungen

von allen Algorithmen liefern. SVM liefert seine phänomenalen Erfolgsquoten nur auf den Traningsdaten, da es overfittet. PDA gewichtet korrelierte Daten sehr stark. Und LDA und QDA sind, wie oben beschrieben, untere Grenzen für RDA.

Leider kommt dieser Perfomancegewinn mit einer enorm hohen Rechenaufwand einher. Dieser hängt erfreulicherweise jedoch ausschließlich von den übergebenen Werten/ Parametern der Cross Validation ab (siehe number of Validations in shinyapp). Für geringe Wahl jener, hält sich die Rechenzeit sehr in Grenzen.

Der größte Teil der Arbeit wurde natürlicherweise nach der Klausurenphase verrichtet. Ganztägig trafen wir uns entweder im Mathematikon oder der Altstadtbibliothek. Der größte Vorteil war, dass die Kommunikation erleichtert wurde und somit Schwierigkeiten, die automatisch durch das Zugreifen auf fremden Code entstanden, schnell geklärt werden konnten.