Beobachtung:

Beim Experimentieren mit MLR wurde mir klar das die vielen 0 werte das lernen beinflussen desshalb muss das problem in einzelteile zerteilt werden verschiedene intervalle wie z.b von 1800 bis 2000 helfen vllt.

A graph with blue lines

Description automatically generated

Aproximationen sind um einen Faktor ~10-100, fehler gefunden sieht nun besser aus außer bei data mit wenig messpunkten 0-10 pro jahr.

RMSE könnte hier unpraktisch sein. Not sure. Weil einheit übereinstimmt?

 **Mean Squared Error (MSE):** Das MSE ist eine Metrik, die die durchschnittliche quadratische Abweichung zwischen den vorhergesagten und den tatsächlichen Werten angibt. Es wird berechnet, indem der durchschnittliche quadratische Unterschied zwischen jeder Vorhersage und dem tatsächlichen Wert im Testdatensatz ermittelt wird. Die Formel für MSE lautet:

MSE=1n∑i=1n(yi−y^i)2MSE=n1​∑i=1n​(yi​−y^​i​)2

Hierbei stehen yiyi​ für die tatsächlichen Werte, y^iy^​i​ für die vorhergesagten Werte und nn für die Anzahl der Datenpunkte.

Eine größere MSE-Zahl bedeutet, dass das Modell größere Fehler in den Vorhersagen hat, wohingegen eine kleinere Zahl darauf hinweist, dass das Modell besser ist.

 **Root Mean Squared Error (RMSE):** Das RMSE ist die Quadratwurzel des MSE und wird häufig verwendet, um die Fehler des Modells in derselben Einheit wie die Zielvariable zu interpretieren. Im Wesentlichen ist es eine Anpassung des MSE, um die Einheit des ursprünglichen Outputs beizubehalten. Die Formel für RMSE lautet:

RMSE=1n∑i=1n(yi−y^i)2RMSE=n1​∑i=1n​(yi​−y^​i​)2

1. ​

Da RMSE die Wurzel des MSE ist, gibt es eine direkte Interpretation in den Einheiten der Zielvariable. Ein niedrigeres RMSE deutet darauf hin, dass das Modell besser in der Vorhersage der Zielvariable ist.

Im Wesentlichen ist der Hauptunterschied zwischen RMSE und MSE die Präsenz der Wurzel in der Formel. RMSE ist die Wurzel des MSE und wird häufig bevorzugt, wenn eine Interpretation der Fehler in der ursprünglichen Einheit der Zielvariable erforderlich ist. Beide Metriken sind nützlich, um die Genauigkeit und Leistung von Regressionsmodellen zu bewerten, wobei niedrigere Werte auf ein besseres Modell hinweisen.

MSE könnte besser sein weil Einheit einen Einfluss auf die genauigkeit der Approximation hat MSE vllt nicht

ich benutze RMSE und MSE um den fehler meiner functions approximation zu bestimmen hier habe ich jedoch das problem das die fhler werte mit der scala scalieren somit kann ich approximationen von werten die 10000 mal vorkommen schlecht mit werten die 100 mal vorkommen vergleichen

Problem gelöst mit zscore normalisierung

Strings mit Komma können nicht eingelesen werden in meiner implementierung wegen delimiter problemen.

Probleme:

* Lfzip läuft nicht
* Sz3 läuft auf unseren datan jedoch kein error rate angezeigt
* Corad läuft aber nicht auf unseren daten wegen DictionaryLearning does not accept missing values encoded as NaN natively. Skitlearn problem maybe daten formatierung wenig zeit reingesteckt
* DWT noch nicht angegangen
* Einlesen Problem pd.read\_csv ist effizient schnell 3 sec für 20000 rows und chunkweises laden sehr einfach und effektiv dafür falsche formatierung not sure if fixable
* Einlesen auf alte art sehr langsam 6 min pro 20000 zeilen und nur teilweise chunkable
* String search über n-1 gram ist zu langsam variierende suchgeschwindigkeit alle files durchgehen schnell für 1 file langsam für alle   
  mögliche lösung semi sortierung ausnutzen

Pyspark string search:

1Gram:

df\_2gram.select(['Id', 'Tokens'])

Bis search 1s auf 1 gram , transformation zu pd + data search 0.8s

Fasziniert suche 18,3s weil tail von df1gram mit Data spalte

Problem 2 gramme tauchen nicht auf? Besipiel 3 gram „ bis seine oder ‚ Geschlecht '

compressWithError2gram 1000

Test 2gramme baisic = 1m 6 sec

Test 2gramme mit partition(5) = 1m 26sec

Test 2gramme mit config = 3m 33 sec

Order hat keine garantie beim einlesen und schreiben von parquet dateien!!

2gram als eine parquet datei ist schneller als mehrere

2 gram in einer parquet datei beschleunigt nochmal!!

conf= SparkConf().setAll([('spark.executor.memory', '16g'), ('spark.executor.cores', '4'), ('spark.cores.max', '4'), ('spark.driver.memory','16g')])

diese konfiguration sorgt für ein stetig langsamer werdende ausführung anfangs jedoch schneller

allgemeines problem das sich mit Länge der ausführung verschlechtert nicht nur diese konfig sonder jede

String suche ohne Data auf parquet ist faktor 100 schneller mit baisic test

Oberes df.head(10)

A screenshot of a computer program

Description automatically generated

A screenshot of a computer screen

Description automatically generated

Versuch code zu optimieren hat geklappt von ~140/s auf 210/s  
  
parralelisierung hat nicht geklappt!