

The Winton Stock Market Challenge

Vorhersage von Finanzmarkttrenditen durch Machine Learning Algorithmen

Freie wissenschaftliche Arbeit zur Erlangung des akademischen Grades
Bachelor of Arts
an der Friedrich-Alexander-Universität Erlangen-Nürnberg

Gutachter:	Prof. Dr. Klein
Betreuer:	Dr. Krauß
Studiengang:	Wirtschaftswissenschaften
Bearbeiter/in:	Mattias Luber
Matrikelnummer:	21834393
Adresse:	Avenariusstraße 13a 90409 Nürnberg

Eingereicht am:

Inhalt

1	Einführung.....	3
1.1	Kaggle – die Plattform für Data Science	3
1.2	Die Winton Stock Market Challenge	4
1.3	Hintergrund	6
1.4	Verwendete Software	6
2	Explorative Datenanalyse	7
2.1	Fehlende Werte und Aufbereitung der Daten	7
2.2	Analyse von Feature_7.....	9
2.3	Analyse von Feature_5.....	11
2.4	Analyse der Zeitreihen und Renditen	11
3	Methodik.....	15
3.1	Die Berechnung der Zero-Benchmark	15
3.2	Base Models	17
3.3	Cross-Validation unter Berücksichtigung der Zeitachse	19
3.4	Hyper-Parameter-Tuning.....	22
3.5	Feature Engineering	24
3.6	Feature Selektion.....	26
3.7	Modellierung der Algorithmen	27
4	Resultate	30
4.1	Algorithmik	31
4.2	Feature Selektion.....	32
4.3	Features.....	33
5	Fazit.....	36
6	Literaturverzeichnis.....	Fehler! Textmarke nicht definiert.
7	Anhang.....	41
7.1	Anhang 1: Das Winton Statement.....	41
7.2	Anhang 2: Tabelle der trainierten Modelle	43
7.3	Anhang 3: Feature Bedeutung	46
8	Eidesstattliche Erklärung	48

1 Einführung

1.1 Kaggle – die Plattform für Data Science

Mittlerweile sind Schlagworte wie Machine Learning und Big Data zwar schon eine ganze Weile im Mainstream angekommen, trotzdem scheinen sich viele Firmen immer noch schwer damit zu tun, deren Methoden gewinnbringend in ihre Geschäftsprozesse zu integrieren. Über die Wichtigkeit der Themen ist man sich meistens einig, nichtsdestotrotz verzichten viele Unternehmen, teils aufgrund fehlender Expertise, teils aus Kostengründen darauf, eigene Data-Science Abteilungen aufzubauen. Selbst die, die es gerne würden, finden aufgrund der großen Nachfrage nach geschultem Personal in diesem Bereich häufig nicht die richtigen Mitarbeiter, um Projekte umzusetzen. (vgl. Shah & Goldbloom, 2013)

Aus diesen Umständen heraus entstand 2011 die Plattform Kaggle. Was als Projekt des Gründers Anthony Goldbloom begann, entwickelte sich schnell zur größten Data-Science Community weltweit. Grundsätzlich funktioniert die Seite wie ein Marktplatz für prognostische Modellierung. Unternehmen mit Problemen in diesem Bereich können Datensätze hochladen und Mitglieder der Community versuchen diese dann im Rahmen von sog. *Competitions* zu lösen. Die Wettbewerbe folgen dabei dem Prinzip des Crowdsourcings. Viele Teilnehmer aus den verschiedensten Bereichen und dementsprechend mit den verschiedensten Ansätzen geben ihre Modelle ab und die besten Algorithmen gewinnen anschließend ein Preisgeld. Evaluiert werden die Modelle über ein Leaderboard in dem jeweiligen Contest auf welchem die Teilnehmer ihre Ergebnisse hochladen können und in Form einer Score werden diese dann bewertet und gerankt. (vgl. Shah & Goldbloom, 2013)

Ein zentraler Kerngedanke von Kaggle Competitions ist neben Crowdsourcing außerdem *Shared Information*, also das Teilen von Informationen. In diesem Zusammenhang sind insbesondere auch die sog. Kernels zu nennen, die Teilnehmer innerhalb von den Wettbewerben für alle einsehbar veröffentlichen können, damit andere Wettbewerber auf diesen Ergebnissen aufbauen können und sich zu neuen Problemlösungsansätzen inspirieren lassen können (vgl. Marr, 2016, pp. 281–286). Zusätzlich gibt es außerdem noch ein Forum für Fragen und Diskussionen rund um den Wettbewerb.

Bei den Kernels handelt es sich um R oder Python Skripte bzw. Notebooks, die über den Browser in der Cloud erstellt und ausgeführt werden können, ohne dass der Nutzer sich eine eigene Entwicklungsumgebung einrichten muss. Da der Code so auf den Servern von Kaggle ausgeführt wird, wird außerdem die Rechenleistung der lokalen Rechner nicht beansprucht. Die Kernels können dann mit Wettbewerben oder Datensätzen verknüpft werden und so direkt auf diese zugreifen, dadurch entfällt das hochladen von u.U. großen

Datenmengen und fördert das einfache Teilen von Informationen und Ergebnissen zwischen den Mitgliedern (vgl. Guo, 2017). Angeboten wird somit quasi ein kostenloses Rundumpaket für Analysten, welches auch außerhalb von Wettbewerben für private oder kommerzielle Projekte genutzt werden kann und mit dem Kaggle versucht ihrem Slogan „your home for data science“ gerecht zu werden.

Ein bisschen ausgegliedert aus der eigentlichen Plattform betreibt Kaggle auch noch den hauseigenen Blog „No Free Hunch“. Darauf werden neben allgemeinen Informationen über die Entwicklung der Plattformen oder aktuelle Themen in Bereich Data Science insbesondere auch Interviews mit den Gewinnern verschiedener Wettbewerbe veröffentlicht. Darin erklären diese, wie sie vorgegangen sind und was ihre wesentlichen Erkenntnisse waren, damit auch die weniger erfolgreichen Teilnehmern von diesen Informationen profitieren können und um deren Lernprozess zu unterstützen.

1.2 Die Winton Stock Market Challenge

The Winton Stock Market Challenge war die zweite *Recruitment-Competition* die von der Investment-Management-Firma Winton Capital auf Kaggle veranstaltet wurde und dementsprechend gab es neben den Preisgeldern von bis zu 20.000 \$ außerdem Bewerbungsgespräche mit dem Analytics-Team von Winton zu gewinnen. Die Aufgabe im Wettbewerb war es, zukünftige Wertpapierrenditen auf Basis ihrer vergangenen Entwicklung, sowie 25 maskierter Merkmale oder Kennzahlen vorherzusagen. Die Maskierung ist insbesondere deshalb gewählt worden, um Teilnehmern mit Branchenwissen im Bereich Trading keinen Vorteil gegenüber Quereinsteigern oder Analysten aus anderen Bereichen zu gewähren. Für Chancengleichheit war somit gesorgt. (vgl. Anderson, 2016)

Gestartet wurde der Wettbewerb im Oktober 2015 und lief dann bis Januar 2016. Anders als bei den meisten Kaggle-Competitions lag die Laufzeit somit nicht bei zwei, sondern bei drei Monaten. Auch das Teilen von Code oder veröffentlichen von Kernels im Zusammenhang mit dem Wettbewerb - wie eigentlich üblich - wurde nicht gestattet, wodurch selbst nach Abschluss der Competition nur begrenzte Informationen verfügbar waren. (vgl. Winton Capital, 2015b)

Designt wurde die Competition von zwei Mitarbeitern der Forschungsabteilung von Winton, die nach eigenen Angaben versucht haben, damit einen guten Überblick ihres Tagesgeschäfts abzubilden. Demnach galt es sich mit großen Mengen an verrauschten Daten und instationären Prozessen auseinander zusetzen (vgl. Anderson, 2016).

Für die Teilnehmer wurde ein Trainingsdatensatz mit 40.000 Einträgen bereitgestellt. Dieser enthielt abgesehen von den 25 Features noch die beiden vergangenen Tagesrenditen, die Minutenrenditen von eins bis 180 und darauf folgen dann die beiden zukünftigen Tagesrenditen. Außerdem je eine Gewichtung für die Tages- und Minutenrenditen (vgl. Winton Capital, 2015a).

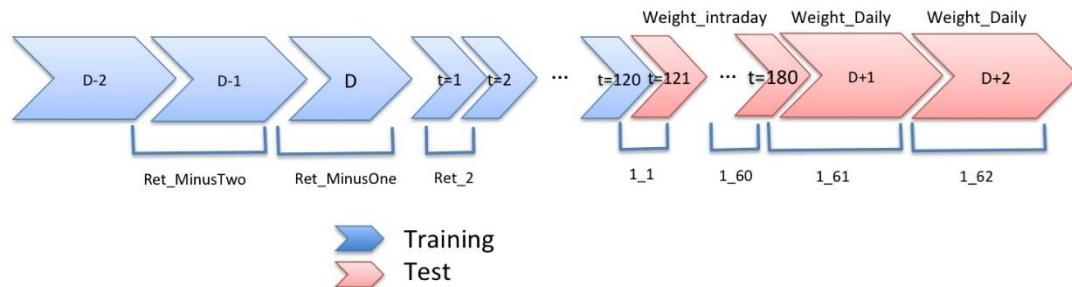


Abbildung 1 graphische Darstellung der Datensätze. (Winton Capital, 2015a)

Im Testdatensatz mit 120.000 Einträgen sind nur die Features, die vergangenen Tagesrenditen und die Minutenrenditen von 1 bis 120 enthalten. Die folgenden 60 Minuten, sowie die folgenden zwei Tage sollten dann von den Teilnehmern entsprechend prognostiziert werden. Als Evaluationsmetrik für die Vorhersagen wurde die gewichtete durchschnittliche absolute Abweichung vom wahren Wert der Renditen gewählt (Weighted Mean Absolute Error, kurz: WMAE), den es zu minimieren galt. Der Veranstalter gestand zwar ein, dass es sich dabei um eine nicht ganz perfekte Messgröße für den Erfolg handelte, allerdings wurde durch die Gewichtung versucht in einer sehr vereinfachten Form Trading-Kosten zu simulieren, mit dem Ziel den Contest noch realitätsnäher zu gestalten. (vgl. Winton Capital, 2015b)

$$WMAE = \frac{1}{n} \sum_{i=1}^n \omega_i * |y_i - \hat{y}_i|$$

Wie für Kaggle Competitions üblich, gab es ein für alle sichtbares öffentliches Leaderboard, dass den Teilnehmern nach dem Hochladen ihrer Vorhersagen den WMAE von 25% der Testdaten als Punktestand errechnete. Für die entscheidende Endwertung des Wettbewerbs wurde nach Abschluss ein privates Leaderboard veröffentlicht, das den WMAE der restlichen 75% der Testdaten abbildete und für die letztendliche Platzierung ausschlaggebend war (vgl. Anderson, 2016; vgl. Winton Capital, 2015b).

Außerdem erwähnenswert ist, dass während des Wettbewerbs der Testdatensatz geändert werden musste. Einigen Teilnehmern gelang es, durch das wiederholte Abgeben von Prognosen für jeweils einige wenige Datenpunkte, zu entschlüsseln, wie die Testdaten zwischen den beiden Leaderboards aufgeteilt wurden. Da auf dem öffentlichen Leaderboard schon während der Challenge validiert werden konnte, war davon auszugehen, dass so auch Rückschlüsse auf das private Leaderboard gezogen werden konnten und Winton entschied sich nach dem Bekanntwerden dazu, die Testdaten anzupassen. Wohlwissend, dass dies ein markanter Eingriff in den Wettbewerb war, begründeten sie ihre Entscheidung damit, dass das Entschlüsseln der Werte keine statistischen Modelle darstellt und somit nicht zugelassen werden könnte. Des Weiteren seien Teilnehmer mit robusten statistischen Modellen auch weiterhin in der Lage den neuen Datensatz ähnlich gut vorherzusagen und diese somit auch nicht maßgeblich benachteiligt würden (vgl. Anderson, 2015).

1.3 Hintergrund

Thema dieser Arbeit war die Vorhersage von Finanzmarkttrenditen am Beispiel der Winton-Stock-Market-Challenge. Die Schwierigkeiten lagen insbesondere darin, Signale in der großen Menge an Störeinflüssen sichtbar zu machen. Letztendlich scheiterte ein Großteil der Teilnehmer an diesem Problem und nicht einmal der Hälfte gelang es überhaupt, die von Winton ausgewählte Benchmark zu schlagen (vgl. Winton Capital, 2016).

Auffällig war außerdem, dass, soweit ersichtlich, im Nachgang des Wettbewerbs tatsächlich kaum funktionierende Modelle anderer Teilnehmer veröffentlicht wurden. Auch wie vom Veranstalter eigentlich angekündigt, wurden die besten Modelle des Wettbewerbs später nicht der breiten Öffentlichkeit zugänglich gemacht (vgl. Anderson, 2016) und dementsprechend ist auch noch keine Top-X-Lösung verfügbar.

Grundsätzlich lässt sich sagen, dass Winton auch lange nach Abschluss des Contests viele Fragen aus dem Forum ungeklärt ließ und bis heute keine detaillierte verfügbar ist. Im Rahmen dieser Arbeit soll demnach ein Ansatz entwickelt werden, der sich mit den oben angesprochenen Schwierigkeiten auseinandersetzt und weiter schrittweise ein zielgerichtetes Vorgehen gezeigt werden, mit dem auch vergleichbare Wettbewerbe erfolgreich absolviert werden können.

1.4 Verwendete Software

Für das Vorgehen wurde auf verschiedene Softwarepakete zurückgegriffen. Aufbauend auf der Entwicklungsumgebung *Anaconda* wurde *IPython* (vgl. Perez & Granger, 2007) als Programmiersprache gewählt. Dieses sogenannte interaktive Python ermöglicht das Erstellen von Jupyter-Notebooks (vgl. Thomas et al., 2016) direkt im Browser, was im Bereich wissenschaftliches Arbeiten einige Vorteile ermöglicht. Die wohl entscheidendsten sind das visuelle Formatieren von Text und Formeln, sowie das interaktive Ausführen von

Code. So können z.B. Graphiken und anderer Output live im Browser angezeigt werden und dokumentenartig in Textpassagen eingebettet werden.

Für die Analyse der Daten wurde dann auf das Paket *Pandas* (vgl. McKinney) zurückgegriffen und die Graphiken darauf aufbauend weitgehend mit *Matplotlib* (vgl. Hunter, 2007) visualisiert.

Mit Ausnahme des *XGBoost* (vgl. Chen & Guestrin, 2016) entspringen die anderen Algorithmen einer Python Bibliothek für maschinelles Lernen Anwendung namens *Scikit-Learn* (vgl. Pedregosa et al., 2011). Abgesehen von unüberwachten und überwachten Lernern werden hier auch für Bereiche wie Datenaufbereitung, Feature Selektion, Cross-Validation und Modelloptimierung entsprechende Module angeboten, weshalb auch in diesen Abschnitten auf *SciKit-Learn* zurückgegriffen wurde. Dass den Entwicklern neben Effizienz insbesondere Benutzerfreundlichkeit ein Anliegen war, zeigt sich auch durch die leichte Integration von ‚fremden‘ Algorithmen über ihre Schnittstelle (vgl. Pedregosa et al., 2011). Über diese wurde dann auch der *XGBoost* Algorithmus der *Distributed (Deep) Machine Learning Community* integriert.

2 Explorative Datenanalyse

Aus dem abschließenden Statement von Winton über den Wettbewerb geht hervor, dass es sich bei den Daten um echte, wenn auch aus Gründen der Vertraulichkeit, durch Transformation und leichtes Verrauschen unkenntlich gemachte Finanzmarktdaten handelt. Die Features sind Kennzahlen der Fundamental- und Chartanalyse, vergangene Entwicklung oder teilweise auch ohne jegliche Bedeutung. Trotzdem wird angenommen das mit Ausnahme von *Feature_5* und *Feature_7* die meisten keine besonders ausgeprägte Vorhersagekraft haben. Die Tages- und Minutenrenditen waren außerdem bereits logarithmiert. (vgl. Anderson, 2016)

2.1 Fehlende Werte und Aufbereitung der Daten

Allgemein zeigte sich allerdings, dass abgesehen von fehlenden Werten, die Datensätze von Haus aus relativ sauber waren. Das war darauf zurückzuführen, dass das Entwickeln einer funktionierenden Cross-Validation-Methode und von Modellen mit tatsächlicher Vorhersagekraft von Winton als anspruchsvoll genug betrachtet wurde und nicht zusätzlich durch Datenaufbereitung erschwert werden sollte (Anderson, 2016). Trotzdem erfordern viele Machine-Learning Algorithmen vollständige und komplette Inputvariablen, weshalb im ersten Schritt der Datensatz auf fehlende Werte untersucht und entsprechend bereinigt werden muss.

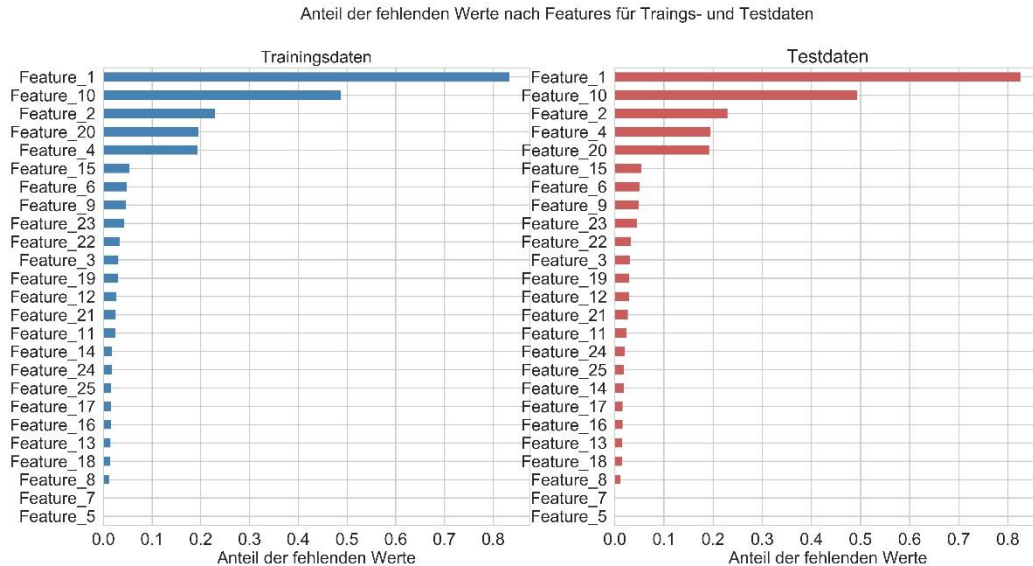


Abbildung 2 Anteil der fehlenden Werte für Trainings- und Testdaten

Wie auf Abbildung 2 zu sehen ist, ist der Anteil der fehlenden Werte bei den Merkmalen über Trainings- und Testdaten ähnlich verteilt. Während einzelne Variablen mit äußerst hohen Fehlerraten hervorstechen, liegt sie bei dem Großteil der Features unterhalb von 10%.

Außerdem fällt auf, dass *Feature_5* und *Feature_7* jeweils überhaupt keine fehlenden Werte enthalten.

Ebenfalls ohne fehlende Werte sind die vergangenen Tagesrenditen. Für die vergangenen Minutenrenditen verteilt sich der Anteil der fehlenden Werte wie folgt:

	Train	Test
Count	40000	120000
Mean	0.046859	0.048532
Min	0.000000	0.000000
25%	0.000000	0.000000
50%	0.000000	0.000000
75%	0.050000	0.050000
Max	0.500000	0.500000

Abbildung 3 Verteilung des Anteils an fehlenden Werten bei den vergangenen Minutenrenditen nach Datensätzen

Man sieht das der überwiegende Teil der Datensätze in Bezug auf die Minutenrenditen relativ sauber ist. Die meisten Einträge sind komplett und enthalten für jede Minute eine Rendite. Selbst bei denen, die unvollständig sind, liegt die Fehlerrate häufig nur bei wenigen

Prozent. Trotzdem gibt es allerdings auch einige Einträge, bei denen bis zu der Hälfte der Renditen nicht enthalten sind.

Nach Kaiser (2014) lässt sich grundlegend nach 3 verschiedenen Ansätzen zum Umgang mit fehlenden Daten unterscheiden. Als erstes dem Reduzieren des Datensatzes um die Einträge, die unvollständig sind. Zweitens das behandeln der fehlenden Werte als „special values“, also dem Zuweisen eines neuen Wertes, der ersichtlich macht, dass es sich dabei um fehlende Werte handelt. Oder drittens, verschiedene Techniken zur Imputation. Wobei hier versucht wird, die nichtvorhandenen Informationen mit sinnvollen Schätzungen zu füllen. Jede der drei Möglichkeiten hat verschiedene Vor- und Nachteile und ihre Anwendung ist abhängig von den gegebenen Daten und der Charakteristik bzw. den Gründen für die Lücken im Datensatz. Nachdem hier allerdings nichts genaueres bekannt ist, worauf die fehlenden Werte zurückzuführen sind, wurde versucht, mit einer Kombination aus Reduktion und Imputation zu arbeiten um den Datensatz zu säubern.

Für die fehlenden Werte in den Minutenrenditen wurde Imputation durch dem Mittelwert entlang der jeweiligen Zeitreihe angewandt, da es sich hierbei um ein sehr einfaches und gängiges Verfahren handelt, und bei den meisten Zeitreihen der Anteil der Fehler ohnehin sehr gering war. Somit ist von einem maßgeblichen Informationsverlust hierdurch eher nicht auszugehen (vgl. Kaiser, 2014). Bei den maskierten Merkmalen wurde für *Feature_5*, *Feature_16* und *Feature_20* angenommen, dass sie kategorialer Natur waren, denn sie traten jeweils nur in 10 oder weniger ganzzahligen Ausprägungen auf. Weil einige Algorithmen sich allerdings äußerst schwer damit tun, diesen Zusammenhang korrekt zu interpretieren, wurden diese Variablen zu Beginn mittels One-Hot-Kodierung in Dummy-Variablen überführt. Anschließend wurden auch hier die Lücken im Datensatz mit den jeweiligen Mittelwerten gefüllt (vgl. Sarkar, Bali, & Sharma, 2018b, p. 169).

2.2 Analyse von Feature_7

Schon während der Competition ist vielen Teilnehmern *Feature_7* im besonderen Maße ins Auge gestochen, wenn auch im Diskussionsforum häufig Unsicherheit herrschte, was die genaue Bedeutung anging oder wie man damit am besten verfahren sollte. Auffällig war besonders die große Bedeutung, die dem Merkmal durch Decision-Tree-Algorithmen zugesprochen wurde, wie hier auf Abbildung 3 beispielhaft zu sehen ist.

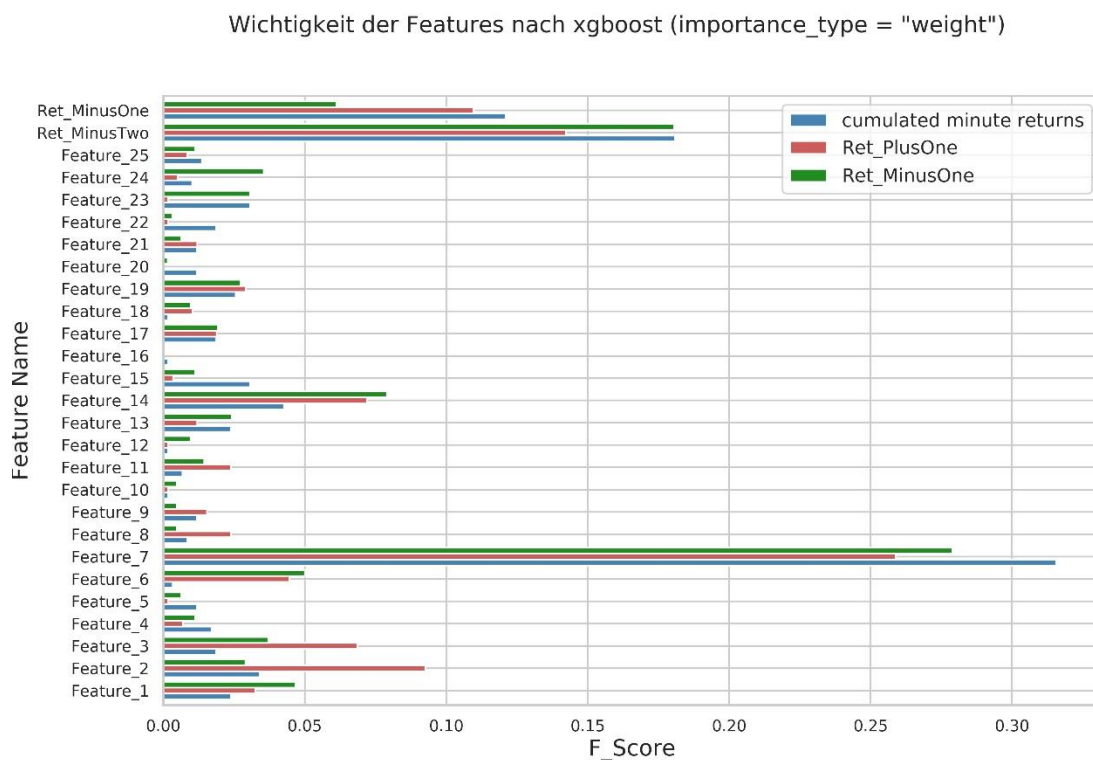


Abbildung 4 Feature Importance nach XGBRegressor auf die rohen Trainingsdaten. Die Minutenrendite Ret_121 bis Ret_180 wurden zu einer kumulierten Periodenrendite zusammengefasst.

Trotz der vermeintlich großen Bedeutung ließen sich damit auf der anderen Seite allerdings kaum Modelle erstellen, die auch auf dem Leaderboard gute Ergebnisse erzielten. Ein Grund dafür war, dass sich die Werte für *Feature_7* zwischen den Trainings- und Testdaten grundlegend unterschieden bzw. sich nicht überschneiden. Während innerhalb eines Datensatzes einzelne Ausprägungen im Schnitt um die 50 Mal auftraten, traten ebendiese Ausprägungen im anderen Datensatz wiederum überhaupt nicht auf. Außerdem sind die Werte für *Feature_7* nicht fortlaufend und springen in unregelmäßigen Abständen z.B. von 26 auf 64 auf 138 und so weiter.

Ersichtlich wurde das Ganze, nachdem Winton bekannt gab, dass es sich hierbei um eine Art Kennnummer für verschiedene Tage handelt und weiter, dass die Wertpapiere Tendenzen aufweisen, innerhalb eines Tages zu korrelieren. Dem entsprechen lernen die Decision-Tree-Algorithmen quasi einfach auswendig was an den jeweiligen Tagen passiert. Nachdem in den Testdaten dann allerdings komplett andere Tage enthalten sind können sie das Gelernte nicht mehr anwenden und liefern entsprechend schlechte Prognosen (vgl. Anderson, 2016).

Diese Erkenntnisse werden später insbesondere beim Entwickeln einer Cross-Validation-Methode wieder aufgegriffen werden und auch im Bereich Merkmalsgenerierung ergeben sich hinsichtlich der zeitlichen Struktur einige Möglichkeiten.

2.3 Analyse von Feature_5

Das zweite Feature, dass Winton als wichtig beschrieben hatte, war *Feature_5*. Hierbei handelt es sich um ein ganzzahliges Feature mit Ausprägungen von eins bis zehn. Die Werte sind zwischen den beiden Datensätzen in etwa gleich verteilt und reichen von ca. 2,0% bis ca. 17,5%.

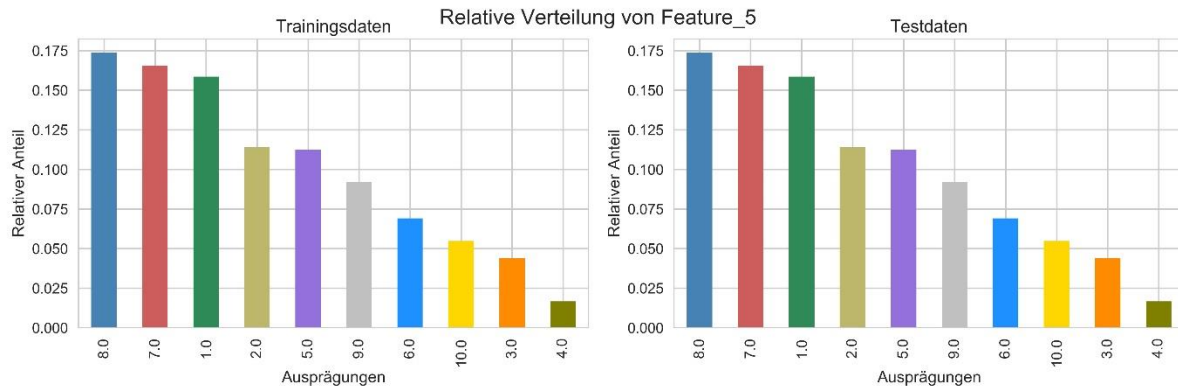


Abbildung 5 Relative Verteilung der Ausprägungen von Feature_5

Eine offizielle Auflösung, um was es sich genau handelt, gab es zwar auch nach der Challenge nicht, allerdings wurde im Forum vermutet, dass es sich hierbei um die Monate Januar bis Oktober handeln könnte (vgl. Forumsdiskussion, 2016). Eine andere Möglichkeit, die in Anbetracht der Verteilung der relativen Anteile denkbar wäre, wäre außerdem eine Art Branchenindex.

2.4 Analyse der Zeitreihen und Renditen

Betrachtet man die empirische Verteilung der Tagesrenditen im Vergleich zur entsprechenden Normalverteilung, zeigen sich die als stilisierte Fakten von Renditen bekannte Spitzigkeit und bei genauerer Betrachtung der Extremwerte auch schwere Ränder. Folglich treten verhältnismäßig häufig sehr kleine Renditen auf, oder stark positive bzw. stark negative. Die Verteilung ist außerdem leicht rechtschief, was allerdings als atypisch gilt (vgl. Cont, 2001).

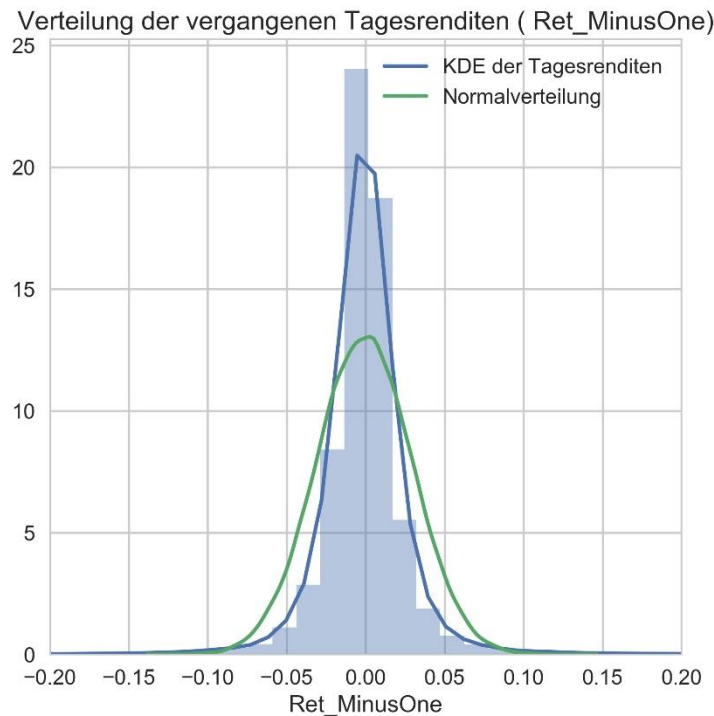


Abbildung 6 Kerndichteschätzung und Histogramm für die vergangenen Tagesrenditen ($Ret_PlusOne$) im Vergleich zur Normalverteilung

Volatilitätscluster sind auf Abbildung 7 nicht so deutlich erkennbar. Zum Teil lässt sich das aber dadurch begründen, dass die Werte für Feature_7, also die Zeitpunkte, nicht durchgehend sind und auch nicht bekannt ist, wie viel Zeit jeweils zwischen zwei Werten vergangen ist. Trotzdem äußern sich Cluster dahingehend, dass sich um bestimmte Zeitpunkte herum die Ausreißer und Extremwerte eher zu konzentrieren scheinen, als bei den gleichen Tagesrenditen, die zufällig nochmal neu auf die Zeitpunkte verteilt, also gemischt, wurden. Beispiele hierfür liegen um 3000, 4500 oder um 9000 (vgl. Schmid & Trede, 2006, pp. 12–15).

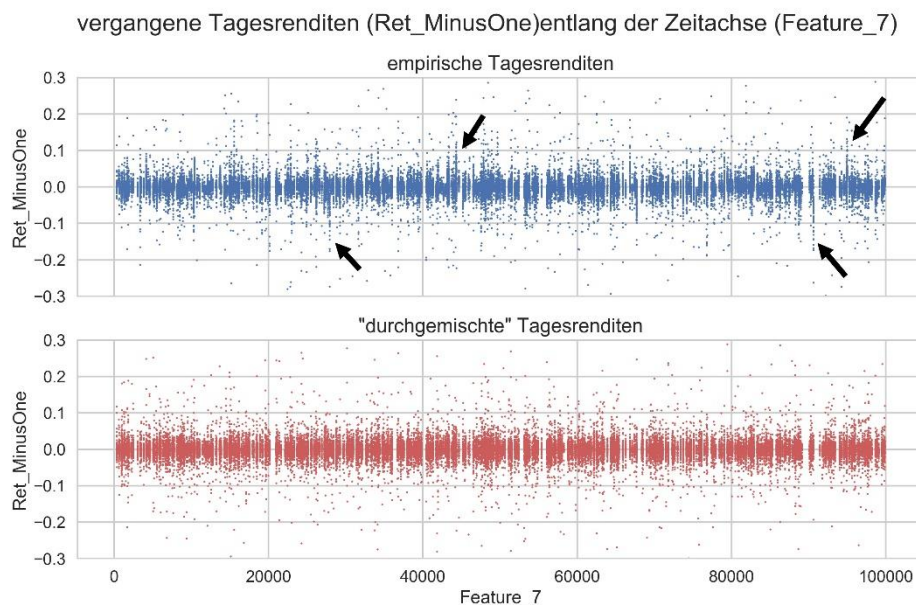


Abbildung 7 vergangene Tagesrenditen ($Ret_PlusOne$) entlang der Zeitachse (Feature_7)

Auch für die Minutenrenditen sind volatile Phasen zu erkennen und in den Kursverläufen finden sich außerdem immer wieder auffällige Sprünge, also für hin und wieder plötzlich auftretende betragsmäßig besonders hohe Renditen.

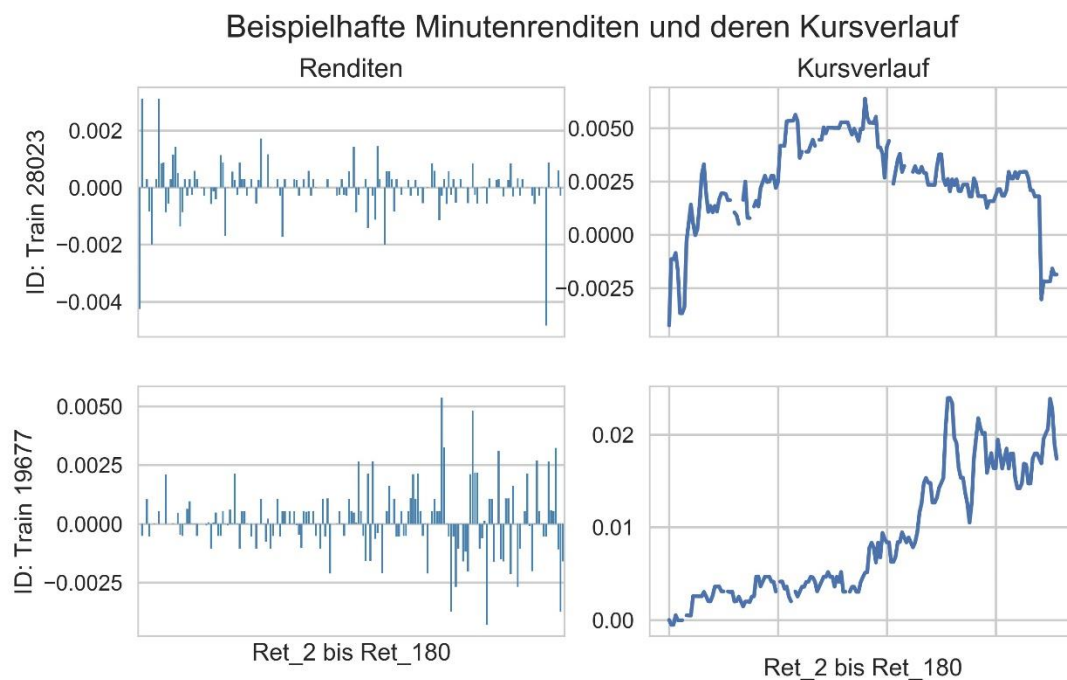


Abbildung 8 beispielhafte Minutenrenditen aus dem Trainingsdaten und deren Kursverlauf.

Anschließend soll nun untersucht werden, wie und ob Merkmale untereinander bzw. mit den Zielvariablen korrelieren. Da der Datensatz mit 210 Variablen schon von Grund auf relativ groß ist und ohnehin nicht davon ausgegangen werden kann, dass einzelne Minutenrenditen über den gesamten Datensatz signifikante Korrelationen aufweisen, da sie nicht nur von verschiedenen Zeitpunkten, sondern auch von unterschiedlichen Tageszeiten innerhalb eines Zeitpunktes stammen können, wurden die vergangenen 120 Minutenrenditen zu *Ret_MinutePast* und die zukünftigen 60 zu *Ret_MinuteFut* zusammengefasst. Daraus ergab sich außerdem, dass die folgende Korrelationsmatrix nicht noch größere Ausmaße annahm und sich noch relativ übersichtlich gestaltete.

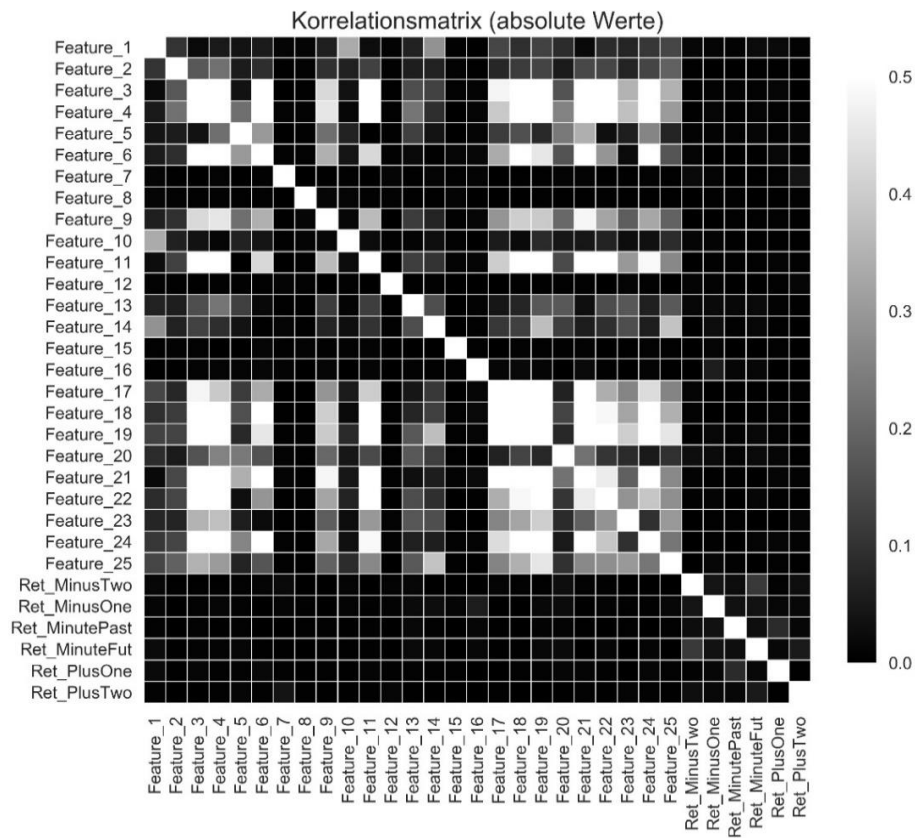


Abbildung 9 Korrelationsmatrix für die 25 Features und die Renditen, *Ret_MinutePast* ergibt sich aus den kumulierten ersten 120 Minutenrenditen, *Ret_MinuteFut* ergibt sich aus den folgenden 60 Minutenrenditen

Zu sehen war, dass sich zwar zwischen den Merkmalen mitunter starke Korrelationen ergaben, zwischen den Merkmalen und den (kumulierten) Renditen allerdings kaum. Das ändert sich jedoch ein wenig, wenn man Korrelationen für einzelne Zeitpunkte untersucht und den Datensatz deshalb nach *Feature_7* gruppierte. Innerhalb eines Zeitpunktes waren durchaus deutlichere Zusammenhänge erkennbar, die sich aber kaum über mehrere Zeitpunkte hinweg generalisieren lassen. Genauer bedeutet das, dass die Korrelationen innerhalb einer Gruppe zwar allgemein schon stärker sind, allerdings für jede Gruppe andere Features hervorstechen. In Gruppe 37168 sind das zum Beispiel *Feature_25* und *Feature_13* für andere Zeitpunkte gilt das dann wiederum überhaupt nicht und andere Merkmale schienen dominanter.

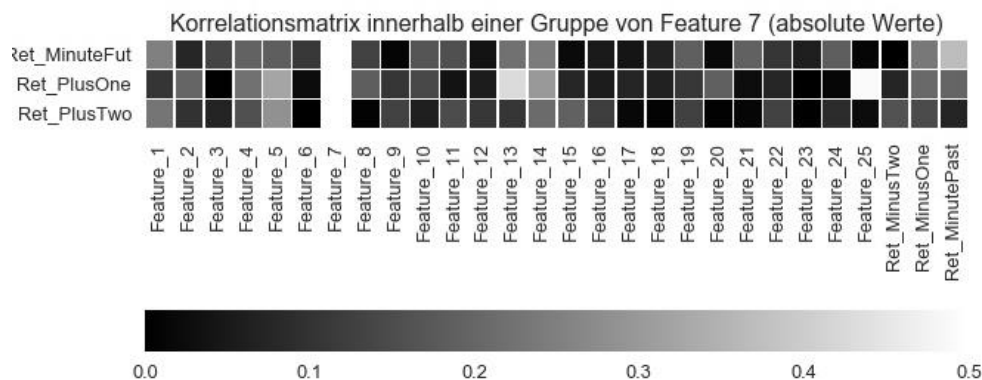


Abbildung 10 Korrelationen gruppiert nach *Feature_7* für die Gruppe 37168

3 Methodik

Der Machine Learning Prozess wurde im folgendem in einige Unterschritte aufgeteilt. Nachdem die Daten im letzten Kapitel erforscht und bereinigt worden sind, wird sich der nächste Abschnitt aber erst einmal mit der Evaluationsmetrik beschäftigen, sowie die Benchmark betrachten, die als Richtwert gilt, ob ein Modell funktioniert oder nicht. Da sich für diese während der Arbeit verschiedene Werte ergeben haben, ist es wichtig zu verstehen nach welchem Schema sie gebildet werden und wie sie ins Verhältnis gesetzt werden müssen.

Nachdem dann bekannt ist, wie Modelle bewertet werden, wurden anschließend auf den von Winton bereit gestellten Daten erstmal einfache Algorithmen *out-of-the-box* getestet. Einmal, um Vergleichswerte für spätere Modelle zu finden und außerdem um zu schauen, wie erfolgreich man damit auf dem Leaderboard hätte abschneiden können

Da Cross-Validation sich quasi als zentrales Thema durch den ganzen Wettbewerb zog und aus dem Winton-Statement schon bekannt war, dass das auf *Feature_7* zurück zuführen war, ging es dann in Sektion 3.3 um die korrekte Implementierung eines Validation-Settings unter Berücksichtigung der Zeitachse. Integriert wurde darin außerdem ein Vorgehen zur Optimierung der Hyperparameter. Diese beiden Prozessschritte wurden beispielhaft an der Ridge-Regression entwickelt, da es sich dabei um einen sehr unkomplizierten, schnellen Algorithmus handelt, mit dem sich trotzdem gute Ergebnisse erzielen ließen. Nachdem anschließend etwas ausgegliedert erklärt wurde, nach welchen Prinzipien Feature Engineering und der Feature Selektion betrieben wurde, wurde dann das gesamte Vorgehen im Abschnitt *Modellierung der Algorithmen* auch auf andere Algorithmen übertragen und die besten Ergebnisse zu einem finalen Modell zusammengeführt.

3.1 Die Berechnung der Zero-Benchmark

Wie gerade bereits angesprochen, ist es für den weiteren Verlauf wichtig zu verstehen, wie genau die Modelle auf dem Leaderboard von Winton bewertet werden, bevor im Anschluss zum eigentlichen Modellieren der Algorithmen übergegangen werden kann.

Als Evaluationsmetrik ist von den Entwicklern der Kaggle-Competition die gewichtete absolute Abweichung vom wahren Wert der Renditen gewählt worden. Die Gewichte sollen in einer sehr einfachen Form Trading-Kosten simulieren und finden sich für den Trainingsdatensatz in den letzten beiden Spalten. Für die Intraday-Renditen und Tagesrenditen wurde für jeden Datenpunkt jeweils mit zwei unterschiedlichen Werten gewichtet und die Gewichte der Testdaten waren aus offensichtlichen Gründen nicht bekannt.

Ins Verhältnis gesetzt wird diese Score zu der sog. Zero-Benchmark, also dem Wert, den man für den WMAE erhält, wenn man für jeden Datenpunkt eine Null vorhersagt, und somit davon ausgeht, dass sich die Kurse der Wertpapiere nicht verändern.

Der massive Einfluss der Gewichtung wird deutlich, wenn man die ungewichtete absolute Abweichung mit dem WMAE vergleicht. Während sich für die Zero-Benchmark im Trainingsdatensatz ungewichtet lediglich ein Wert von 0.00111 ergibt, beträgt die der WMAE 1773.9244. Da sich die Gewichte für jedes Wertpapier bzw. jede Zeile in einem Datensatz unterscheiden, ergeben sich für den Trainingsbereich je nach Leaderboard somit auch unterschiedliche Zero-Benchmarks. Insbesondere zwischen den beiden Leaderboards weichen die Scores deutlich voneinander ab.

	Trainings-Set	Public Leaderboard	Private Leaderboard
Zero-Benchmark	1773.92440	1770.03211	1728.62346

Abbildung 11 die jeweiligen Zero-Benchmarks im Vergleich

Gleiches zeigt sich außerdem, wenn man den Trainingsdatensatz weiter in Trainings- und Validationsdaten unterteilt. Da jede Unterteilung der Daten andere Werte für die Gewichte enthält, errechnet sich somit auch ein neuer WMAE. Ein guter Wert bei der Validierung ergibt sich demnach unter Umständen lediglich aus ‚besseren‘ Gewichten und nicht aus besseren Prognosen.

	Trainings-Set	Validations-Set
Split 1	1774.320947	1772.734739
Split 2	1772.171982	1779.181636
Split 3	1778.381342	1760.553556

Abbildung 12 Zero-Benchmarks für drei zufällige Splits in Trainings- und Validationsdaten mit jeweils 75% Trainingsdaten und 25% Validationsdaten

Veranschaulicht wurde dies in Abbildung 12, auf der für jeweils drei zufällige Unterteilungen die beiden Zero-Benchmarks abgebildet sind. Zu erkennen ist, dass nicht nur zwischen den einzelnen Splits, sondern auch innerhalb eines Splits die Werte der beiden Scores teils deutlich voneinander abwichen. Es sollte also unbedingt darauf geachtet werden, Verbesserungen immer relativ im Vergleich zur entsprechenden Score der gleichen Aufteilung zu betrachten.

Dieses Phänomen wirkte sich für manche Teilnehmer auch auf ihre Platzierung zwischen den beiden Leaderboards mehr oder weniger glücklich aus. Nachdem die Endergebnisse veröffentlicht wurden, war zu sehen, dass sich die Platzierungen auf dem privaten Leaderboard teilweise deutlich von denen auf dem öffentlichen unterschieden. Für fast die komplette Top 10 bedeutete dies mitunter massive Abwertungen in der Rangliste, lediglich die beiden Besten blieben hiervon unberührt. Auf der anderen Seite konnten dafür viele

Teilnehmer mit konservativeren Public-Scores überzeugen und stellten so die neue Top 10. Unter Berücksichtigung dieser Beobachtung sollte später bei der Modellauswahl also unbedingt darauf geachtet werden, nicht einfach das Modell zu nehmen, das den absolut besten Wert auf öffentlichen Leaderboard erreicht. Vielmehr geht es darum abzuwägen, ob von den Modellen zu erwarten ist, dass sie konsistent gute Werte erzielen und sich demnach auch auf dem privaten Leaderboard als robust erweisen werden (vgl. Winton Capital, 2016).

3.2 Base Models

Um zu erkunden, wie viel Vorhersagekraft in den von Winton bereitgestellten Daten tatsächlich enthalten ist, wird zu Beginn versucht, die beiden zukünftigen Tagesrenditen auf Basis der 25 Features und der vergangenen Tagesrenditen zu prognostizieren. Die zukünftigen Minutenrenditen werden hier außer Acht gelassen. Einmal weil im Winton-Statement bereits erklärt wurde, dass sich einzelne Minutenrenditen eigentlich nicht vorhersagen lassen (Anderson, 2016) und zum anderen weil im Forum-Thread ‚Solution Sharing‘ einige Teilnehmer erkennen ließen, dass sie sich ebenfalls darauf beschränkt hatten und trotzdem oder gerade deswegen gute Ergebnisse erzielten. Darunter zum Beispiel auch der Zweitplatzierte Humberto Brandão, der nach eigenen Angaben nur 74 Werte für *Ret_PlusOne* prognostizierte und den Rest mit Nullern auffüllte (vgl. Forumsdiskussion, 2016).

Verglichen werden im Folgenden die Ergebnisse für einen *XGBoost*-Regressor und eine Ridge-Regression. Gewählt wurden diese Algorithmen, da *XGBoost* aufgrund von Boosting und Ridge durch Schrumpfen der Regressionskoeffizienten als robust gegenüber Overfitting gelten (vgl. James, Witten, Hastie, & Tibshirani, 2017, pp. 217–219; vgl. Schapire, 2013). Nachdem bereits bekannt ist, dass *Feature_7* besonders behandelt werden sollte, wurden die Algorithmen jeweils einmal mit und einmal ohne diesem trainiert.

Da *XGBoost* auch mit fehlenden Werten umgehen kann (vgl. Chen & Guestrin, 2016), wird dieser nochmal auf die rohen Trainingsdaten angewendet. Für die Ridge Regression werden zunächst die Features 1, und 10 entfernt, da der Anteil der fehlenden Werte teilweise weit größer als 40% bzw. über 80% ist und eine Imputation mittels des Mittelwerts demnach vermutlich eher ungenau. Die restlichen Features wurden wie in Abschnitt 2.1 beschrieben, aufbereitet.

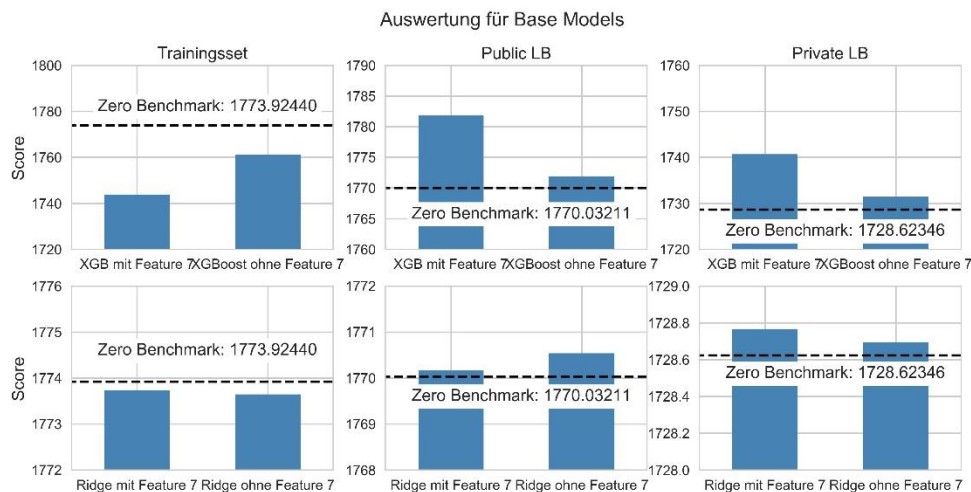


Abbildung 13 Auswertung der Basis Modelle

Die Ergebnisse für die beiden Algorithmen sind auf Abbildung 13 zu sehen. Beide Algorithmen scheinen prinzipiell zu overfitten, da die Ergebnisse in der Trainingsumgebung die Zerobenchmark mit Leichtigkeit schlagen. Auf den Leaderboards gelingt ihnen das allerdings nicht und somit lässt sich sagen, dass die Prognosen nicht generalisieren. Um ein Gefühl für die Größenordnung zu bekommen, die es im Vergleich zur Benchmark zu erreichen gilt, sei gesagt, dass die Score des Erstplatzierten mit 1727.53575 auf dem privaten Leaderboard gerade einmal eine Verbesserung von ca. 1,1 darstellt und ein Wert von 1728.04 bereits gereicht hätte um sich in den Top 10 zu platzieren. Daran wird ersichtlich, dass vor allem die Ergebnisse für *XGBoost* mehr als bescheiden abschneiden und dazu außerdem stark variieren. Die Ridge-Regression liefert stabilere Ergebnisse, die zwar auf den Trainingssets nicht ganz so gut wie der Boosting-Algorithmus abschneiden, auf der anderen Seite dafür auf den Leaderboards auch nicht ganz so schlecht. Abgesehen davon scheint Feature_7 für den XGB tatsächlich Überanpassung zu begünstigen, da hier die Differenzen zwischen den Trainings- und Test-Scores mit Abstand am größten sind.

Nun ist es nicht verwunderlich, dass das Vermeiden von Überanpassung eine der Hauptaufgaben der Challenge werden sollte, ist es schließlich auch eins der großen Themen im Bereich Machine Learning. Um dem Überanpassen entsprechend zu begegnen, ist es erst einmal wichtig zu verstehen, was genau es ist und wo die Ursachen dafür liegen könnten. Domingos (2012) zieht hierfür den Bias-Varianz-Trade-Off herbei und unterteilt schlechte Prognosen in „high bias“ und „high variance“. Modelle mit großem Bias neigen dazu, aus verschiedenen Trainingsdatensätzen die gleichen Fehler zu lernen und so konsistentere aber verzerrte Schätzungen zu liefern. Große Varianz bedeutet, abgesehen vom eigentlichem Signal außerdem zufällige Informationen aus dem Noise zu lernen und so für verschiedene Datensätze stark abweichende Prognosen zu liefern. Allgemein tendieren lineare Modelle eher zu verzerrten, aber konsistenten Schätzungen, und flexiblere Modelle

wie z.B. Entscheidungsbäume zu geringerem Bias, dafür aber größerer Varianz. Grundlegendes Ziel wäre Modelle zu finden, die in beiden Bereichen gut abschneiden.

Die Ursachen für Overfitting können vielfältig sein und sind nicht immer nur auf Störeinflüsse zurück zu führen. Dementsprechend gibt es auch keine one-fits-all Lösung um es zu vermeiden, sondern es gilt eher problemspezifisch unterschiedliche Ansätze zu testen und zu kombinieren. Neben Cross-Validation wird von Domingo (2012) Regularisierung angesprochen, also dem Einführen eines Strafterms, der Überanpassung vermeiden soll. Bei der Ridge Regression wird das mit dem bereits erwähnten *Shrinkage*-Koeffizienten erreicht, der die Regressionskoeffizienten in Richtung Null drückt und so zu konservativeren Schätzungen führt. Die Regression mittels *XGBoost* versucht das Problem mit Boosting anzugehen. Der Algorithmus kombiniert hier viele einfache Modelle zu einem robusterem Komplexen (vgl. Hastie, Tibshirani, & Friedman, 2017, 61-68 ; 337-341). Trotzdem scheint hier keiner der beiden Ansätze recht zu funktionieren. Allerdings wurden auch jeweils die Standardparameter der in *SciKit-Learn* bzw. *xgboost* implementierten Algorithmen beibehalten, auf Cross-Validation verzichtet und außerdem ist nicht einmal gesichert ob in den Features überhaupt geeignete Informationen enthalten sind.

Dementsprechend geht es im nächsten Schritt nun darum eine geeignete Cross-Validation-Methode zu entwickeln, anhand der dann die jeweiligen Modellparameter optimiert werden können.

3.3 Cross-Validation unter Berücksichtigung der Zeitachse

Das Entwickeln einer funktionierenden Cross-Validation-Methode war das große Problem, an dem bei der Challenge so viele gescheitert sind. Während sich einige damit abgefunden haben, dass sich die Trainingsergebnisse nicht aufs Leaderboard übertragen ließen und voll und ganz ihrem Cross-Validation-Set vertraut haben, gab es auch einige die stattdessen lieber jedes Mal ihre Ergebnisse hochgeladen und auf dem Leaderboard überprüft haben. Eine sehr aufwändige Methode zur Validierung der Ergebnisse, die allerdings auch vom Drittplatzierten Mendrika Ramarlina angewandt wurde oder um es in seinen Worten zu sagen: „My conclusion, local CV was useless in this competition.“ (Forumsdiskussion, 2016).

Der Veranstalter der Challenge räumte klassischer Cross-Validation-Techniken tatsächlich keine großen Erfolgsaussichten ein. Stattdessen empfahl er, in jedem Fall die Daten so zu teilen, dass sich zwischen Trainings- und Validationsdaten keine gemeinsamen Werte für *Feature_7* finden lassen und die Daten so nach Zeitpunkten zu gruppieren (vgl. Anderson, 2016).

Die einfachste Form der Cross-Validation ist das Aufteilen der Trainingsdaten in Trainings- und Validationsset. Normalerweise werden die Daten zufällig einer der beiden Gruppen zugeteilt. Da allerdings soweit möglich keine Überschneidungen für *Feature_7* gewünscht

sind, werden die Daten zuerst nach den Zeitpunkten sortiert und dann die ersten 30.000 Einträge der Trainingsgruppe zugeteilt und die nächsten 10.000 dem Validationsset. Auch hier ist wieder zu beachten, dass die Gewichte des WMAE für die Score eine große Rolle spielen, demnach müssen nun für die Trainings- und Validationsdaten wiederum neue Zero-Benchmarks errechnet werden.

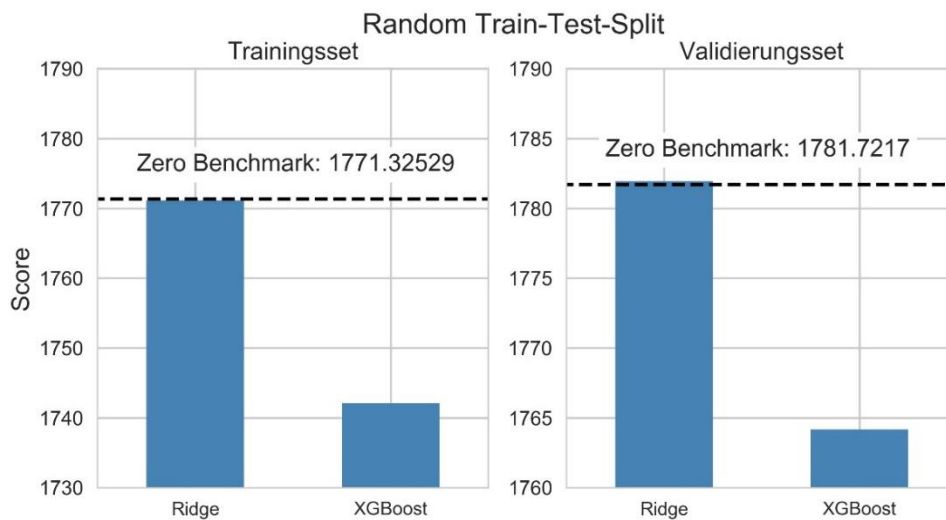


Abbildung 14 randomisierter Train-Test-Split



Abbildung 15 Train-Test-Split gruppiert nach Feature_7

Im Vergleich zu einer zufälligen Aufteilung der Daten, bei der der *XGBoost* die Zeitpunkte auswendig lernen kann, liefert das Validationsset mit Berücksichtigung der Zeitachse ein ähnliches Bild wie für die Testdaten. *Feature_7* scheint somit tatsächlich der Schlüssel für eine repräsentative Cross-Validation-Methode zu sein. Die *Ridge*-Regression ist davon zwar nicht ganz so stark betroffen, da es *Feature_7* als diskretes Merkmal wahrnimmt und demnach kaum in einzelne Zeitpunkte separieren kann, trotzdem trat auch hier eine Annäherung an das Bild der Testscore ein.

Auch wenn dieser *Train-Test-Split* relativ leicht und umgänglich zu implementieren ist und im ersten Resultat auch vielversprechende Ergebnisse erzielt hat, hat er auf der anderen Seite zwei entscheidende Nachteile: Erstens können die Ergebnisse auf dem Validierungs-Set stark von den dort enthaltenen Daten abhängig sein und verschiedene Aufteilungen in die jeweiligen Gruppen stark abweichende Ergebnisse erzielen. Bzw. anders formuliert: Die Ergebnisse des Validation-Sets lassen sich ggf. schlecht auf andere Daten übertragen. Zweitens kann auf einem Viertel der Daten nun nicht mehr trainiert werden, weil diese für die Validierung reserviert sind. Unter Umständen gehen so wertvolle Informationen verloren (vgl. James et al., 2017, pp. 176–178).

Besser ist meistens die Verwendung einer *KFold*-Cross-Validierung. Hier wird der Datensatz in *k* aufeinanderfolgende Gruppen aufgeteilt und so entsprechend durchiteriert, dass jeweils auf *k-1* Gruppen trainiert und die verbleibende geschätzt wird, bis für alle Unterteilungen eine Score vorliegt. Dadurch kann der gesamte Datensatz zum Trainieren benutzt werden und durch das Mitteln der jeweiligen Scores erhält man am Ende ein robusteres Ergebnis. (vgl. Clarke, Fokoue, & Zhang, 2009, pp. 600–611)

SciKit-Learn bietet verschiedene Cross-Validation Methoden an, die es erlauben die Unterteilung entsprechend eines Gruppenparameters - in diesem Fall *Feature_7* - so einzustellen, dass sich dessen Werte zwischen den Aufteilungen nicht überschneiden. Neben *GroupKFold*, die, abgesehen von der Nichtüberschneidung, wie die *KFold* Cross-Validierung funktioniert, existiert außerdem *GroupShuffleSplit*. Hier wird der Datensatz nicht in feste Gruppen unterteilt, sondern es werden *n* Permutationen von überschneidungsfreien Train-Test-Splits erstellt. Ein Vorteil, der sich hier ergibt, ist, dass das Verhältnis von Trainings- und Validierungsdaten individuell eingestellt werden kann und z.B. ein größeres Validierungs-Set gewählt werden kann, um dadurch ggf. das Leaderboard noch besser abbilden zu können.

Method	N-Splits	Train/Validation Ratio	Mean Validation Score	Std Validation Score
GroupShuffleSplit()	10	0.4/0.6	0.015812	0.000202
GroupShuffleSplit()	5	0.4/0.6	0.015883	0.000164
GroupShuffleSplit()	10	0.25/0.75	0.015853	0.000090
GroupShuffleSplit()	5	0.25/0.75	0.015830	0.000101
GroupKFold()	10	0.90/0.10	0.015784	0.001179
GroupKFold()	5	0.80/0.20	0.015781	0.000897

Abbildung 16 Vergleich der verschiedenen Cross-Validation-Verfahren

Als Score dient hier, anders als bei der vorherigen Validierung, der mittlere absolute Fehler (Mean Absolute Error, kurz: MAE) ohne Gewichte. Zum einen deswegen, weil es *Scikit-Learn* standardmäßig nicht erlaubt Gewichte für dessen Berechnung einzufügen und zum anderen, da sich dadurch für jede Unterteilung innerhalb der Cross-Validation die Score von den dort enthaltenen Gewichten abhängig wäre und sich so nicht untereinander vergleichen lassen würden. Für *GroupKFold* ergeben sich so tendenziell niedrigere Mittelwerte und *GroupShuffleSplit* erzielt je nach dem Verhältnis der Validierungsdaten eine geringere Abweichung. Da keine Methode in beiden Punkten herausragend abschneidet, *GroupShuffleSplit* mit 10 Splits und einem Anteil von 60% für die Validierungsdaten jeweils zumindest überdurchschnittlich gut, wird diese als Cross-Validation-Methode für den weiteren Verlauf gewählt.

3.4 Hyper-Parameter-Tuning

Durch die Cross-Validierung ist es nun möglich, auf den Trainingsdaten die Vorhersagekraft des Modells bzw. den MAE als deren Maß zu bestimmen. Nun gilt es genau diese durch die Optimierung der Hyperparameter der Algorithmen weiter zu verbessern. Bei Hyperparametern handelt es sich um die Parameter, mit der die Leistung der Modelle weiter eingestellt werden kann. Bei der *Ridge*-Regression ist das z.B. der *Shrinkage*-Koeffizient und bei Random Forest u.a. die Anzahl der Bäume. Auch wenn in *SciKit-Learn* bereits für nahezu jeden Algorithmus sinnvolle Standardeinstellungen implementiert wurden, kann die Optimierung der Parameter die Leistung des Modells trotzdem maßgeblich verbessern. Während die *Ridge*-Regression mit dem einen Parameter noch händisch optimiert werden könnte, wird das spätestens bei steigender Parameterzahl äußerst aufwendig und es wird ein systematischeres Vorgehen erforderlich. Eine der bekanntesten Möglichkeiten dafür ist die Gitter-Suche. Hierbei wird für jeden Parameter eine Liste von Werten angegeben und dann systematisch alle Möglichkeiten durchgerechnet bis das Optimum gefunden ist. In *Scikit-Learn* ist das im Modul *GridSearch* umgesetzt, das sich außerdem äußerst einfach in die Cross-Validation-Prozedur eingliedern lässt (vgl. Swamynathan, 2017, pp. 140–163).

Welche entscheidenden Auswirkungen Modell-Tuning haben kann, zeigt sich spätestens bei dessen Anwendung.

Optimierung des Alpha Koeffizienten für Ridge

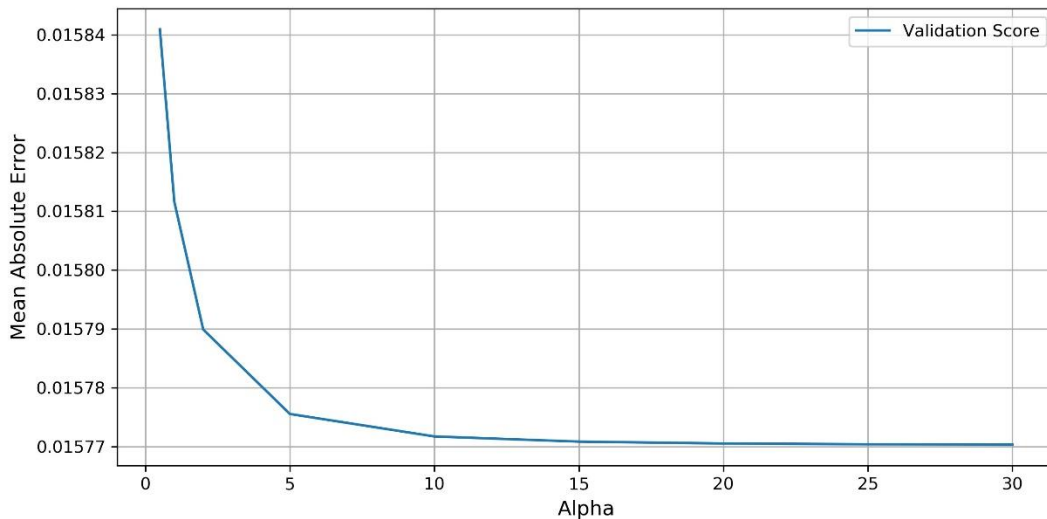


Abbildung 17 Mean Absolute Error auf dem Validation-Set für verschiedene Werte des Regularisierungsparameters α

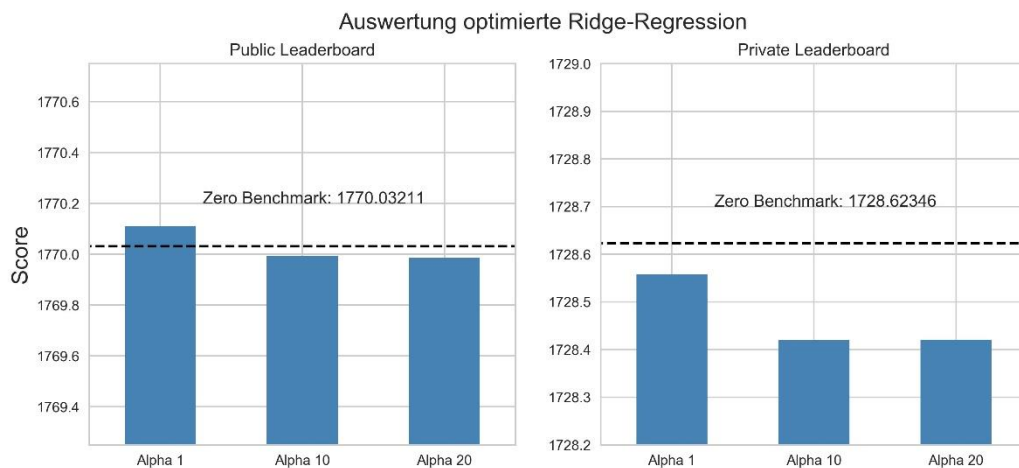


Abbildung 18 Auswertung der optimierten Ridge-Modelle auf den beiden Leaderboards

Schon eine leichte Erhöhung des Alpha Koeffizienten hat eine verhältnismäßig starke Verbesserung des MAE zur Folge, die sich dann auch direkt auf die Leaderboard-Wertungen übertragen lässt. Anzumerken ist allerdings, dass sich ab einem gewissen Wert nur noch eine marginale Verbesserung einstellt und deshalb vernachlässigt werden kann. Die Gitter-Suche ergab hier ein Optimum für den Wert 20, mit dem auf dem privaten Leaderboard eine Score von 1728.42104 errechnet. Damit schlägt dieses Modell zur Vorhersage von *Ret_PlusOne* basierend auf den 25 gegebenen Features und der vergangenen Tagesrenditen zwar die Zerobenchmark um 86 Platzierungen und rangiert damit auf dem 286. Platz, trotzdem ergibt sich daraus noch kein wirklich zufriedenstellendes Ergebnis.

In Betracht auf die Ergebnisse der Top-Platzierten ist also noch deutlich Luft nach oben. Doch nach Cross-Validierung und *GridSearch*-Optimierung kann davon ausgegangen

werden, dass sich durch den Algorithmus alleine keine allzu großen Verbesserungen mehr erzielen werden lassen. Trotzdem gibt es noch weitere Methoden um das Modell weiter zu verbessern. Diese beruhen darauf, dass in den gegebenen Daten noch weitere Informationen enthalten sind, die nur noch nicht so dargestellt sind, dass der Algorithmus sie erkennen kann. Es müssen aus den vorhandenen Daten also neue Features generiert werden um die versteckten Informationen besser sichtbar zu machen (vgl. Sarkar et al., 2018b, pp. 171–172).

3.5 Feature Engineering

„Coming up with features is difficult, time-consuming, requires expert knowledge. ‘Applied machine learning’ is basically feature engineering.“ (Ng zit. nach Sarkar et al., 2018b, p. 174). Dementsprechend ist es auch nicht verwunderlich, dass in den vielen Kaggle-Competitions die meiste Zeit dafür aufgewendet wird, Merkmale aus den vorhandenen Daten zu extrahieren oder ggf. mit Hilfe von branchenspezifischen Wissen Neue zu generieren. Das Vorgehen sowie die Resultate beim Feature Engineering sind stark abhängig von der jeweiligen Problemstellung und allgemein gibt es keine wirklichen Vorgaben, wie Merkmale generiert werden sollen, solange sie die Vorhersagekraft des Modells verbessern (vgl. Sarkar, Bali, & Sharma, 2018a).

Leider hatte Winton Capital das Teilen und Veröffentlichen von Informationen während des Contests verboten und auch im Nachgang wurden von anderen Teilnehmern kaum detaillierte Ergebnisse öffentlich, wodurch auch nicht ersichtlich wurde, auf welchen Features deren Prognosen basierten. Einzige wirklich valide Quelle hierfür war ein Interview mit dem Drittplatzierten Mendrika Ramarlina, das auf dem Kaggle-Blog *No-Free-Hunch* veröffentlicht wurde. Hier gab er zwar an, dass Feature Engineering sein Schlüssel zum Erfolg war, was seine genauen Merkmale anging, lies er allerdings nur allgemein verlauten, unter anderem mit Drawdown-Duration, Drawdown-Magnitude und kumulierten Minutenrendite gearbeitet zu haben (vgl. Kaggle Team, 2016).

Daneben waren in direktem Bezug zu der Challenge nur noch im Forum-Thread *Solution Sharing* weitere Anhaltspunkte zu finden. Der Zweitplatzierte gab hier z.B. an mit Volatilität gearbeitet zu haben, andere Teilnehmer nannten außerdem gewichtete Mittelwerte/Medians der vergangenen Renditen als Teil ihres Feature-Sets (vgl. Forumsdiskussion, 2016).

Hilfreich gestalteten sich außerdem noch zwei Interviews mit Top-5 Platzierten einer ähnlichen Trading-Challenge von 2Sigma. Konkret gab einer der beiden neben seinem Vorgehen beim Modellieren insbesondere an, mit verzögerten Renditen und nach Zeitpunkt gruppierten Mittelwerten, bzw. der jeweiligen Abweichung davon gearbeitet zu haben (vgl. Kaggle Team, 2017a). Der andere versuchte u.a. mit einer Mischung aus deskriptiven

Merkmale wie der Standardabweichung oder marktbezogenen Features, wie der Volatilität zu einem bestimmten Zeitpunkt das Problem zu lösen (vgl. Kaggle Team, 2017b).

Anhand der dort verfügbaren Informationen und unter Einfluss eines Papers von Zura Kakushadze (2016) in dem veranschaulicht wurde, wie Merkmale für Finanzmarktdaten allgemein generiert werden können, wurde anschließend ein Feature-Set aufgebaut, das sich unter anderem wie folgt zusammensetzt:

- (absolute) Differenzen und Summer der vergangenen Tagesrenditen
- Deskriptive Merkmale der vergangenen Minutenrenditen
- Deskriptive Merkmale der geglätteten vergangenen Minutenrenditen
- Der Mittelwert der nach Zeitpunkt (Feature_7) gruppierten Tagesrenditen
- Die mittlere absolute Abweichung der nach Zeitpunkt (Feature_7) gruppierten Tagesrenditen, sowie deren (absolute) Differenzen
- Der Mittelwert der nach vermeintlicher Branche (Feature_5) gruppierten vergangenen Tagesrenditen, sowie deren (absolute) Differenzen
- Die mittlere absolute Abweichung der nach vermeintlicher Branche (Feature_5) gruppierten vergangenen Tagesrenditen, sowie deren (absolute) Differenzen
- Der Mittelwert und die mittlere Abweichung der nach Zeitpunkt und Branche gruppierten vergangenen Tagesrenditen, sowie deren (absolute) Differenzen
- Kumulierte Minutenrenditen für die letzten x Minutenrenditen
- Absolute Abweichung der letzten x Minutenrenditen
- Maximaler Drawdown
- Einzelne Interaktionsterme
- ...

Insgesamt wurden so über 100 neue Merkmale generiert und berechnet, anhand deren im weiteren Verlauf Modelle trainiert werden sollen. Die Herangehensweise war bei vielen Features allerdings ähnlich. Nachdem außer *Feature_7* keines der Anderen genauer erklärt wurde, begrenzten sich die Informationen ansonsten letztendlich auf die Zeitreihendaten der Minutenrenditen, anhand derer (gruppierte) Mittelwerte, Streuung und Interaktionen darunter errechnet wurden, wohingegen im Bereich Trading viele Alphas eigentlich auf Handelsvolumen und Orderbooks aufbauen (vgl. Kakushadze, 2016). Auch Tageshoch/-tief oder Schlusskurse waren durch das begrenzte Zeitfenster von 120 Minuten nicht ersichtlich und das Bilden von Langzeittrends bei nur zwei Tagesrenditen nicht möglich, was den Handlungsspielraum zusätzlich einschränkte. Trotzdem wurde versucht, möglichst vielfältige Merkmale zu generieren die Volatilität, branchentypisches Verhalten und zeitpunktbezogene Trends berücksichtigen.

Für einen Teil der neu errechneten Features wurden zusätzlich noch mithilfe von *SciKit-learn's PolynomialFeatures*-Modul generische Interaktionen mit dem Grad Zwei erstellt. Damit soll später getestet werden ob eine höhere Ordnung und somit flexiblere Modelle insbesondere bei den linearen Modellen bessere Werte erzielen.

3.6 Feature Selektion

Aufgrund der hohen Dimensionalität, die durch das Feature Engineering entstanden ist ergeben sich allerdings auch einige Nachteile. Nicht nur die benötigte Rechenzeit für die Modelle, sondern insbesondere auch Gefahr von Überanpassung nimmt mit steigender Anzahl an Variablen zu. In diesem Zusammenhang wird oft der von Bellman (1961) eingeführte Ausdruck „*the curse of dimensionality*“ genannt. Es beschreibt grundsätzlich den exponentiellen Anstieg an benötigten Datenpunkte bei zunehmender Dimensionalität für einige Algorithmen. Daraus ergibt sich in Bezug auf maschinelles Lernen auch, dass bei einer hohen Anzahl an irrelevanten Variablen im Datensatz, das wahre Signal von deren Rauschen bis zur Unkenntlichkeit überlagert werden kann. Selbst bei einer hohen Anzahl an relevanten Merkmalen kann es so bei einigen Modellen zu Schwierigkeiten kommen (vgl. Domingos, 2012).

Daraus ergibt sich demnach auch die Notwendigkeit das hochdimensionale Datenset auf einige hilfreiche Variablen zu reduzieren, bevor damit Modelle zur Vorhersage der Renditen erstellt werden können.

Für die Reduktion fiel die Wahl zunächst auf Filter-Methoden basierend auf den Korrelationen und von Mutual-Information-Koeffizienten zwischen den Input- und den Zielvariablen. Ganz allgemein ergibt sich aus Filter-Methoden der Vorteil, dass das Selektionskriterium unabhängig vom Algorithmus ist und die Variablen in dieser Hinsicht neutral ausgewählt werden. Durch die Auswahl durch Korrelationskoeffizienten ergibt sich dafür allerdings auch der Nachteil, dass dadurch auch sehr leicht redundante Informationen mehrfach ins Modell aufgenommen werden, was die Ergebnisse wiederum verschlechtern kann. Aus diesem Grund wurden daneben auch noch Mutual-Information-Koeffizienten als zweites Kriterium gewählt. Bei deren Berechnung wird entsprechend berücksichtigt, wie viel neue Informationen eine Variable tatsächlich enthält und so vermieden das das Modell durch Informationsredundanz zu verzerren (vgl. Guyon & Elisseeff, 2003).

Für die Ridge-Regression wurden daneben außerdem drei Feature-Sets durch rekursives Eliminieren von Merkmalen gebildet. Dieses Vorgehen zählt zu den sog. *Wrapper*-Methoden und nutzt die Regressionskoeffizienten, um damit so lange die ‚schlechteste‘ Variable zu entfernen, bis die gewünschte Anzahl an Merkmalen übrigbleibt. Das bedeutet allerdings auch, dass vor jeder Elimination ein neues Modell mit neuen Koeffizienten trainiert werden muss, wodurch diese Methode sehr schnell sehr rechenintensiv wird und ein bisschen Zeit in

Anspruch nehmen kann. Für die Ridge-Regression spielt das eher eine untergeordnete Rolle, da sie als lineares Modell nur wenig Zeit zum Trainieren braucht. Aus Gründen der Effizienz wurde deshalb aber darauf verzichtet, dieses Verfahren auch bei den im nächsten Kapitel eingeführten Algorithmen Random Forest, Gradient Boost und Huber Regression anzuwenden (vgl. Huijskens, 2018).

Nachdem auch die Anzahl der Variablen in einem Modell eine Bedeutung spielen kann, wurde auch hier variiert und jeweils zwischen 10 und 30 Werten ausgewählt. Letztendlich ergaben sich dadurch folgende 10 Feature-Sets:

Name	Features
Core	25 Basis-Features von Winton
mic10	Die 10 Variablen mit dem höchsten Mutual-Information-Koeffizienten
mic20	Die 20 Variablen mit dem höchsten Mutual-Information-Koeffizienten
mic30	Die 30 Variablen mit dem höchsten Mutual-Information-Koeffizienten
corr10	Die 10 Variablen mit dem absolut höchsten Korrelationskoeffizienten
corr20	Die 20 Variablen mit dem absolut höchsten Korrelationskoeffizienten
corr30	Die 30 Variablen mit dem absolut höchsten Korrelationskoeffizienten
polycorr15_2	Die 15 Variablen mit dem absolut höchsten Korrelationskoeffizienten aus den polynomialen Merkmalen
micpoly15	Die 15 Variablen mit dem höchsten Mutual-Information-Koeffizienten aus den polynomialen Merkmalen

Abbildung 19 Übersicht über die verwendeten Feature-Sets

Für die Ridge-Regression kamen wie bereits angesprochen außerdem die rekursive Variablenauswahl dazu.

Name	Features
rfe10	Die verbleibenden 10 ‚besten‘ Merkmale nach rekursiver Elimination
rfe20	Die verbleibenden 20 ‚besten‘ Merkmale nach rekursiver Elimination
rfe30	Die verbleibenden 30 ‚besten‘ Merkmale nach rekursiver Elimination

Abbildung 20 Feature-Sets nach durch rekursiven Feature Elimination

Mit der Selektion der Variablen ist somit auch der letzte vorbereitende Schritt zur Modellierung abgeschlossen und es kann zur Auswahl verschiedener Algorithmen und dem eigentlichen Prognostizieren der Renditen übergegangen werden.

3.7 Modellierung der Algorithmen

In *SciKit-Learn* sind im Bereich überwachtes Lernen einige Algorithmen implementiert, die jeweils auf verschiedenen mathematischen Verfahren beruhen und somit jeweils andere Vor- bzw. Nachteile aufweisen. Da aus Kapazitätsgründen natürlich nicht für jeden ein Modell trainiert werden kann, wird versucht sich auf ein paar möglichst verschiedene zu begrenzen.

Aus der Gruppe der generalisierten linearen Modelle wurden aufgrund der Regularisierung die *Ridge-Regression* ausgewählt und außerdem noch die *Huber-Regression*, da sie Ausreißer in den Daten besonders berücksichtigt. Unter den Ensemble-Methoden ist es ein *ExtraTreeRegressor* und weiter ein *GradientBoostingRegressor* als Boosting-Algorithmus. Die Auswahl wurde bewusst so getroffen, da jeder auf eine unterschiedliche Art versucht overfitting zu vermeiden und um so einen guten Überblick über die verschiedenen Arten von Lernen in *SciKit-Learn* zu geben.

Allgemein wurde dann jeweils so vorgegangen, dass die Algorithmen mit jedem der Feature-Sets die Cross-Validation Prozedur durchlaufen und die Hyperparameter entsprechend mittels Gittersuche optimieren. Analysiert wurden nicht nur die Werte für die (gewichtete) Trainingsscore, sondern ebenfalls die ungewichtete absolute Abweichung, nach der innerhalb der *GridSearch* optimiert wurde (hier Validationscore genannt). Bevorzugt wurden dann Modelle ausgewählt, die in beiden Bereichen überdurchschnittlich gut abschnitten und sich somit als einigermaßen robust erwiesen. Der Hintergrund hierfür war, dass so auch eher anzunehmen war, dass sich die Ergebnisse zwischen den beiden Leaderboards gut übertragen ließen. Diese ausgewählten Modelle wurden dann auf Kaggle hochgeladen, um auch die Score für die Testdaten zu erhalten. Darauf basierend wurden dann die jeweils besten und robustesten zu einem finalen Modell zusammengeführt.

Für *Ret_PlusOne* wurden folgende fünf Modelle abgegeben:

Algorithmus	Feature-Set	Trainings-Score	Validation-Score	Public Leaderboard	Private Leaderboard
Ridge	corr30	1773,07735	0,015754584	1770,02983	1728,19687
GradientBoosting	corr20	1772,54671	0,015757635	1769,97922	1728,33719
Huber	corr20	1773,37159	0,015752186	1769,8872	1728,17555
Ridge	corr20	1773,23601	0,01575706	1769,96408	1728,22647
Ridge	rfe30	1773,16997	0,015773929	1769,86364	1728,11325

Abbildung 21 Übersicht über verschiedene Modelle für *Ret_PlusOne*

Auffällig ist besonders, dass vier der fünf Algorithmen auf den korrelierenden Merkmalen aufbauen. Scheinbar scheint das für *Ret_PlusOne* ein solides Auswahlkriterium zu sein. Geschlagen wird dieses Feature-Set hier nur durch rekursive Elimination. Auch beachtenswert ist, dass sich im Großen und Ganzen die Ergebnisse zwischen den jeweiligen Leaderboards und der Trainings-Score decken. Modelle die auf einem davon gut abschneiden, schneiden oft auch auf den anderen gut ab. Demnach ist anzunehmen, dass so tatsächlich robuste Modelle ausgewählt wurden, die gut generalisierende Prognosen liefern.

Für *Ret_PlusTwo* wurde nach dem gleichen Prinzip selektiert:

Algorithmus	Feature-Set	Trainings-Score	Validation-Score	Public Leaderboard	Private Leaderboard
GradientBoosting	polycorr15_2	1770,71726	0,015160851	1769,89034	1728,55118
Huber	mic10	1773,83897	0,015176167	1769,96813	1728,49637
GradientBoosting	mic20	1772,12563	0,015188886	1769,97526	1728,53272
GradientBoosting	corr10	1771,50038	0,015181409	1769,99037	1728,32765
Ridge	polycorr15_2	1772,99782	0,015181914	1769,95755	1728,20435

Abbildung 22 Übersicht über verschiedene Modelle für *Ret_PlusTwo*

Während sich für *Ret_PlusOne* insbesondere Ridge-Algorithmen durchsetzen konnten, lieferten für *Ret_PlusTwo* eher *GradientBoosting*-Methoden verhältnismäßig überdurchschnittliche Werte. Außerdem markant ist, dass diesmal auch auf die polynomialen Feature-Sets zurückgegriffen, bei denen nur Interaktionen oder quadrierte Merkmale enthalten waren. Eine höhere Ordnung scheint also für *Ret_PlusTwo* ein guter Ausgangspunkt zu sein. Daneben kamen auch die nach Mutual-Information selektierten Merkmale zur Geltung. Diese schnitten absolut gesehen zwar letztendlich nicht ganz so gut ab wie die Anderen, in Bezug auf die Übertragbarkeit der Ergebnisse zwischen den Leaderboards erwiesen sie sich allerdings als sehr robust.

Für die Vorhersage der Minutenrenditen konnte trotz vieler verschiedener Techniken kein Modell gefunden werden, dass die Zerobenchmark schlägt. Das deckt sich, entnommen aus dem Forum, auch mit den Erfahrungen, die viele andere Teilnehmer gemacht hatten und die meisten begrenzten sich darauf, die beiden Tagesrenditen vorherzusagen (vgl. Forumdiskussion, 2016). Zwar bestätigte Winton, dass dies grundsätzlich möglich sei und empfahl anstatt einzelner Minuten lieber Periodenrenditen oder Volatilität zu prognostizieren (vgl. Anderson, 2016), trotzdem stellten sich auch dadurch keine nennenswerten Verbesserungen ein. Auch das eliminieren der Ausreißer aus den Trainingsdaten änderte nichts und so wurde sich für die Minutenrenditen anstatt von Algorithmik für die einfache Vorhersage von Nullern entschieden.

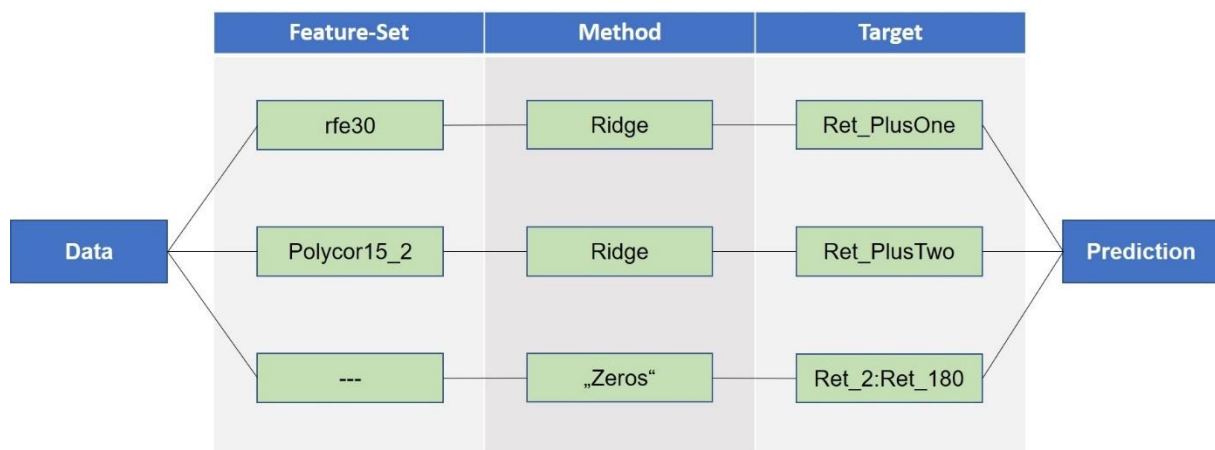


Abbildung 23 graphische Darstellung der Modellkomponenten

Die jeweils besten und in den Tabellen grau hinterlegten Algorithmen der Tagesrenditen wurden dann anschließend zusammengeführt und gemeinsam mit den Nuller-Prognosen für die Minutenrenditen bilden sie nun das finale Modell, welches auf Abbildung 23 auch nochmal graphisch dargestellt ist.

Ein letztes Mal wurden dann damit die Renditen der Challenge vorhergesagt und evaluiert. Die Ergebnisse zeigen, dass sowohl auf den Trainings- als auch auf den Testdaten die Zerobenchmark deutlich geschlagen werden konnte und in Bezug auf das Leaderboard wäre letztendlich die Winton-Stock-Market-Challenge mit einem herausragenden zweiten Platz abgeschlossen worden.

	Trainings-Set	Public Leaderboard	Private Leaderboard
Zero-Benchmark	1773.92440	1770.03211	1728.62346
Finales Modell	1772.43613	1769.81651	1727.72102

Abbildung 24 Auswertung des finalen Modells


#	△pub	Team Name	Kernel	Team Members	Score ?	Entries	Last
1	—	Just Pay your Bill's			1727.53575	178	3y
2	—	humbertbrandao.com consu...			1727.73785	68	3y
3	▲12	Mendrika Ramarlina			1727.81187	122	3y
4	▲24	Statistical_Instigator			1727.91537	5	3y

Abbildung 25 Screenshot des privaten Leaderboards (Winton Capital, 2016)

Submission and Description	Private Score	Public Score	Use for Final Score
Final.csv 7 days ago by matze add submission details	1727.72102	1769.81651	<input type="checkbox"/>

Abbildung 26 Scores des finalen Modells

4 Resultate

Nun wurde im letzten Kapitel gezeigt, durch welches Vorgehen die Competition erfolgreich absolviert hätte werden können und das Ziel einer Top-1%-Lösung sogar noch übertroffen. Trotzdem ist damit über die Hintergründe des Erfolgs noch nicht viel ausgesagt. Daher soll im Folgenden erörtert werden, warum sich bestimmte Algorithmen durchsetzen konnten und welche Features dabei am meisten geholfen haben. Auch auf die Methoden der Feature Selektion wird noch einmal genauer eingegangen, da die Reduktion des Datensatzes beim Vorgehen ein besonders zentraler Bestandteil des Prozesses gewesen ist.

4.1 Algorithmik

Für die Tagesrenditen ließen sich besonders mit dem Ridge-Algorithmus positive Ergebnisse erzielen, der deshalb im Folgenden genauer betrachtet werden soll. Grundlegend handelt es sich dabei um ein erweitertes lineares Modell, dass von Hoerl and Kennard (1970) entwickelt wurde, um dem Problem von multikollinearen Regressoren bei der Methode der kleinsten Quadrate zu begegnen. Diese beruht eigentlich auf der Annahme von unabhängigen Prädiktoren und auch, wenn sich bei verletzter Annahme noch unverzerrte Schätzungen ergeben, kann die Residuenquadratsumme schnell, stark ansteigen. Eine hohe Varianz in der Schätzung ist die Folge. Weiter resultiert daraus außerdem, dass die Koeffizienten der kollinearen Regressoren oft unverhältnismäßig große absolute Werte annehmen und u.U. sogar mit einem falschen Vorzeichen versehen sind. Für das Modell bedeutet das nach den Ansätzen von Melkumova and Shatskikh (2017), dass schon geringe Änderungen in den Datenpunkten dazu führen können, dass sich deutlich verschiedene Koeffizienten ergeben. Das Modell lässt sich zu stark von Noise beeinflussen, anstatt den darunterliegenden Prozess zu beschreiben und liefert daher auch keine generalisierenden Ergebnisse. Um die genannten Punkte zu umgehen, führten Hoerl and Kennard einen *Shrinkage*-Parameter k (in der Implementierung von *SciKit-Learn* wurde dieser *alpha* genannt) ein, der die Koeffizienten bei steigendem Wert Richtung Null drückt, so das lineare Modell regularisiert wird und für realistischere Koeffizienten sorgt. Trotzdem werden die Koeffizienten nie komplett Null und anders als bei der verwandten *Lasso*-Regression bleiben alle Variablen im Modell enthalten (vgl. McDonald, 2009).

Formal ergibt sich so für die Berechnung der Koeffizienten:

$$\widehat{\beta}^* = [X'X + kI]^{-1}X'Y$$

Für $k = 0$ entspricht das dem klassischen linearen Modell, ansonsten ist die Wahl des Parameters immer ein trade-off zwischen Bias und Varianz, weshalb ein allgemeines Optimum hier nur schwer definiert werden kann. Eher wird k bzw. *alpha* durch Cross-Validation so ermittelt, dass eine jeweils gewünschte Messgröße optimiert wird. Im Fall des Wettbewerbs war das der MAE. Andere Ansätze empfehlen stattdessen darauf zu achten, dass die Koeffizienten sich bei dem gewählten Wert bereits stabilisiert haben und bei weiterem Anheben von *alpha* nur noch geringfügig schrumpfen. Für gewöhnlich reicht oft schon eine geringfügige Regularisierung aus, um den gewünscht Effekt zu erzielen. Im Rahmen der Challenge wurde für die beiden Tagesrenditen allerdings ein weitaus höherer Wert von 18 für Ret_PlusOne bzw. 15 für Ret_PlusTwo ermittelt. Das scheint in Anbetracht der stark verrauschten Daten allerdings auch durchaus schlüssig, wenn Überanpassung effektiv entgegengewirkt werden soll.

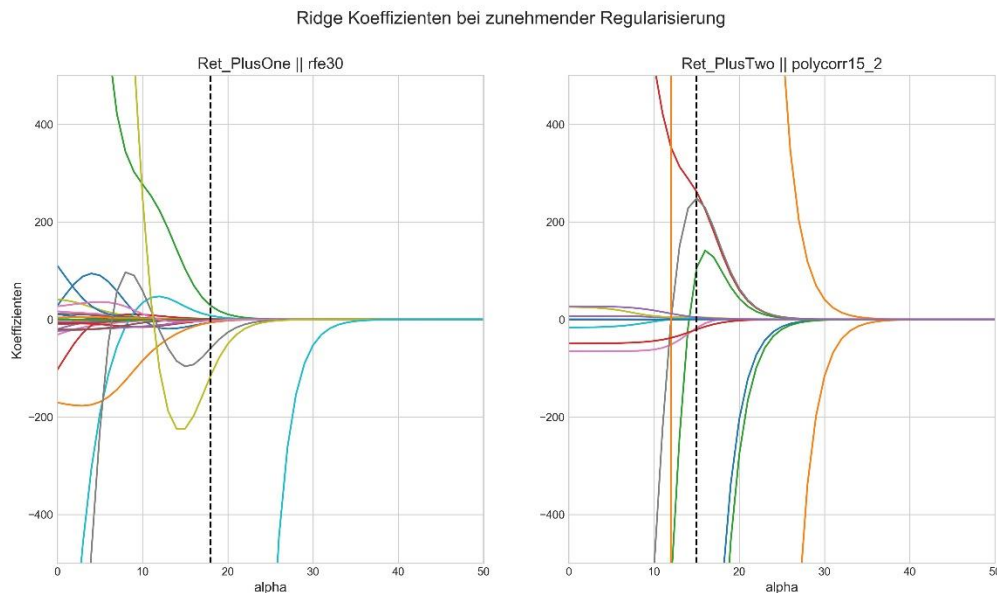


Abbildung 27 Ridge-Koeffizienten bei zunehmender Regularisierung, die gestrichelte Linie gibt jeweils das im Modell gewählte alpha an.

Grundlegender Kerngedanke der Ridge-Regression ist der, dass durch den zusätzlichen Parameter bewusst leichter Bias zur Schätzung hinzugefügt wird, dafür dadurch auf der anderen Seite die Varianz deutlich verringert werden kann. Insgesamt können so, trotz leichter Verzerrung, Modelle erzeugt werden, die eine geringere Residuenquadratsumme aufweisen, als die Methode der kleinsten Quadrate. Insbesondere kollineare Regressoren werden dadurch geschrumpft und im Vorzeichen korrigiert.

In Bezug auf die Challenge lässt sich dadurch ableiten, dass es vielleicht gerade durch die bewusste Inkaufnahme von Bias zugunsten von robusteren Prognosen dazu geführt hat, dass nicht nur auf den Trainingsdaten, sondern auch konsistent auf beiden Leaderboards vergleichbare Ergebnisse erreicht werden konnten.

4.2 Feature Selektion

Feature Selektion war auch deshalb ein wichtiger Punkt beim Vorgehen, da durch die Art, wie neue Features generiert wurden, Kollinearität nicht ausgeschlossen wurde, bzw. eher sogar bewusst in Kauf genommen. So wurden z.B. für die Minutenrenditen die Varianz und die mittlere Absolutabweichung (Mean Absolute Deviation, kurz: MAD) berechnet, obwohl davon auszugehen ist, dass diese nicht ganz unabhängig voneinander agieren und das ist nur eins von vielen Beispielen. Zwar lässt sich Kollinearität nicht zwangsläufig durch Korrelationen bestimmen, da in den Feature-Sets allerdings durchaus sehr hohe paarweise Korrelationen auftraten, ist zumindest davon auszugehen, dass einige der Merkmale lineare

Abhängigkeiten aufweisen (vgl. Mansfield & Helms, 1982). Auch durch die Auswahl der Features auf Basis von betragsmäßig hohen Korrelationen wurde dahingehen nicht selektiert. Nachdem allerdings viele der Algorithmen für *Ret_PlusOne* ebensolche Feature-Sets bevorzugt hatten, scheinen die enthaltenen Merkmale tatsächlich die richtigen Informationen zu enthalten. Umso wichtiger war es demnach, dass die angewendeten Modelle entsprechend mit Kollinearität umgehen können und sich dadurch nicht negativ beeinflussen lassen. Neben der Ridge-Regression, die wie im vorherigem Abschnitt beschrieben, extra für solche Situationen entwickelt worden ist, schien auch der *GradientBoostingRegressor* damit keine Probleme zu haben. *ExtraTreeRegressor* auf der anderen Seite schnitten hier nicht so gut ab. U.a. wahrscheinlich auch, da sie eine flache Struktur der Features voraussetzten und lineare Abhängigkeiten oder Korrelationen nur schlecht modellieren können (vgl. Matsuki, Kuperman, & van Dyke, 2016, p. 10).

Ein weiteres Indiz, das in die gleiche Richtung deutet, ergibt sich aus der Auswahl durch *Mutual-Information-Koeffizienten*. Hier wird explizit darauf geachtet, Informationsredundanz und somit auch Multikollinearität zu vermeiden (vgl. Huijskens, 2018). Auch wenn es aus Abbildung 24 nicht ganz so deutlich hervorgeht, konnte trotzdem festgestellt werden, dass nahezu jeder Algorithmus zusammen mit den so ausgewählten Merkmalen ähnlich solide Ergebnisse lieferte. Diese schlugen unabhängig von der Art des Lernalgorithmus auf beiden Leaderboards die Benchmark deutlich und in einem ähnlichen Maßstab. Trotzdem konnten dadurch nicht die Werte erreicht werden, die mittels Korrelationen und *Ridge* bzw. *GradientBoosting* unter Berücksichtigung von linearen Abhängigkeiten erzielt wurden. Zusammenfassend lässt sich hier sagen, dass die Auswahl auf Basis von Korrelationen im Schnitt dann für bessere Prognosen lieferte, wenn ein Algorithmus gewählt wurde, der robust gegenüber Multikollinearität war.

4.3 Features

Da die Daten der Challenge mittels des in *SciKit-Learn* implementierten Scaler der *Ridge*-Regression jeweils automatisch normalisiert wurden, können die Features außerdem hinsichtlich ihrer ceteris-paribus-Änderungsrate leicht miteinander verglichen werden. Die Modelle der Ridge-Regression lassen sich dementsprechend also außerdem gut interpretieren (vgl. Melkumova & Shatskikh, 2017; vgl. Nanny, 1975). Vergleicht man die als einflussreich bewerteten Features der beiden Tagesrenditen, so fällt relativ schnell auf, dass sich für den Tag weiter in der Zukunft allgemein auch eine höhere Ordnung in den Features ergibt. Während schon für *Ret_PlusOne* die Interaktionsterme deutlich höher bewertet werden, als die linearen Features, scheint sich dieser Effekt bei *Ret_PlusTwo* sogar noch zu verstärken. Hier besteht das Set mit den besten Prognosen quasi nur noch aus Features höherer Ordnung. Während die polynomialen Features hier i.d.R. die Ordnung Zwei haben,

stechen insbesondere $x_0 x_{20}$ und $x_0 x_{21}$ durch hohe Interaktionen hervor, hinter denen sich sogar eine noch höhere Ordnung verbirgt.

Da die Namen der jeweiligen Features auf der Abbildung 28, insbesondere für das polynomielle Set, kryptische Züge ausweisen, findet sich im Anhang eine Auflistung der jeweiligen Merkmale und wie sie berechnet wurden. Aus Gründen der Übersichtlichkeit wurde hier allerdings darauf verzichtet, die vollen Namen auszuschreiben bzw. aufzuführen.

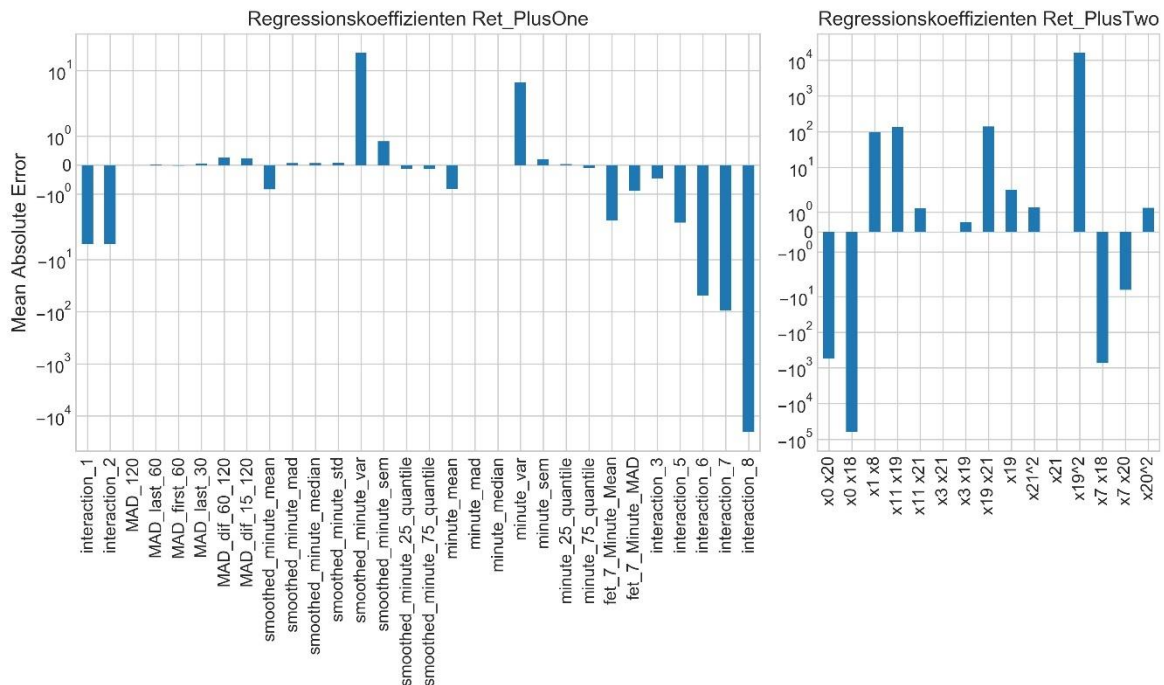


Abbildung 28 Reg (Clarke et al., 2009) regressionskoeffizienten der Modelle für die Tagesrenditen

Um zu ergründen, warum und welche Informationen genau für die Vorhersage der Renditen entscheidend sein können, werden daher die Merkmale mit dem vermeintlich größten Einfluss in Abbildung 29 genauer betrachtet.

	Feature	Koeffizient	Bedeutung
Ret_PlusOne	interaktion_8	-20292,61	Interaktion der nach Feature_7 gruppierten MAD und der nach Feature_7 gruppierten Mittelwerte der Minutenrenditen
	interaktion_7	-95,12	Interaktion zwischen der Periodenrendite über 120 Minuten und dem MAD der Minutenrenditen gruppiert nach Feature_7
	interaktion_6	-49,18	Interaktion zwischen dem Mittelwert der geglätteten Minutenrenditen und der nach Feature_5 gruppierten MAD
	smoothed_minute_var	21,89	Varianz der geglätteten Minutenrenditen

Ret_PlusTwo	X0 x20	-552,21	Interaktion zwischen dem MAD der 120 Minutenrenditen, der Periodenrendite der letzten 5 Minuten und dem Mittelwert der Periodenrendite gruppiert nach Feature_7
	X0 x18	-62951,93	Interaktion zwischen dem MAD der 120 Minutenrenditen, der Periodenrendite der letzten 5 Minuten und dem Mittelwert der Minuten gruppiert nach Feature_7
	X19^2	16109,17	Nach Zeitpunkt (Feature_7) gruppierte MAD der Minutenrenditen (quadriert)
	X7x18	-739,48	Summe der letzten beiden Minutenrenditen und Mittelwert der Minutenrenditen zu einem Zeitpunkt (Feature_7)

Abbildung 29 Die vermeintlich wichtigsten Features mit Koeffizienten und Erklärung

Vor allem die Features auf Grundlage der Minutenrenditen kamen zur Anwendung, die vergangenen Tagesrenditen allerdings kaum in irgendeiner Form. Daraus kann man schließen, dass die Informationen der 120 Minutenrenditen allgemein besser geeignet sind, um die zukünftigen Tagesrenditen zu prognostizieren, als die statischen 25 Basis-Features aus der Fundamental- und Chartanalyse, oder als vergangene Tagesrenditen.

Weiter zeigte sich deutlich, dass nahezu jedes wichtige Merkmal in irgendeiner Art und Weise die Zeitachse in Form von Feature 7 berücksichtigt. Meistens wurde danach gruppiert, um so Charakteristika des Marktes zu einem bestimmten Zeitpunkt abbilden zu können. Für die Vorhersage wurden dann oft die Interaktionen von Informationen über einzelne Wertpapiere mit denen, des gesamten Marktes am jeweiligen Tag berechnet. Für interaction_7 wurde beispielsweise nach diesem Prinzip die Streuung eines Wertpapiers mit dem Trend des gesamten Marktes kombiniert, um die Rendite von Morgen vorherzusagen.

Letztendlich konnte u.a. dadurch die zweitbeste Prognose der Wertpapiere im ganzen Wettbewerb erreicht werden. Trotzdem zeigt sich bei der Analyse der Ergebnisse auf Abbildung 30, wie schwierig Finanzmarktdaten zu prognostizieren sind. Somit finden sich zwischen den tatsächlich eingetretenen Renditen und den Vorhergesagten, immer noch teils große Differenzen. Insbesondere für einen Bereich der positiven Renditen konnte der Algorithmus allerdings nahezu konstant die entsprechende Tendenz vorhersagen und so die wahren Werte merklich besser abbilden.

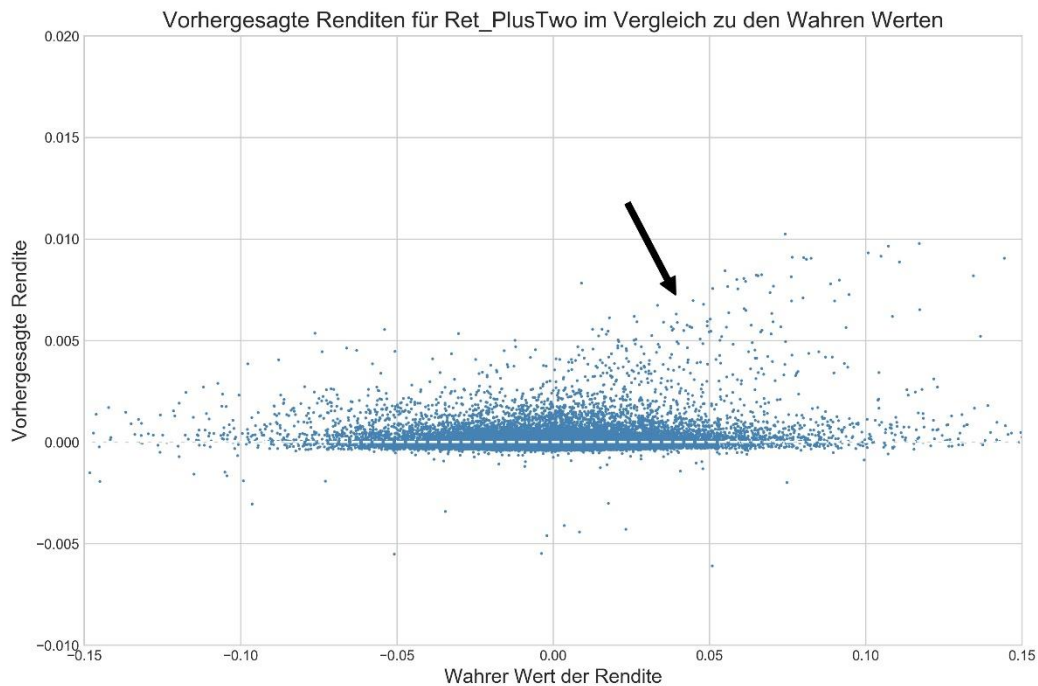


Abbildung 30 Die vorhergesagte Rendite von *Ret_PlusTwo* im Vergleich zu den Tatsächlichen Renditen der Trainingsdaten

5 Fazit

Zusammenfassend lässt sich sagen, dass die Winton-Stock-Market-Challenge in vielerlei Hinsicht einer interessanter Kaggle-Wettbewerb war und der Umgang mit schwachen Signalen in verrauschten Datenmengen eine echte Herausforderung. Während die bereitgestellten Daten keiner großer Aufbereitung mehr bedurften, war das Entwickeln einer guten Cross-Validation-Methode der große Knackpunkt in dem Contest. Der Schlüssel zum Erfolg war hier für viele Teilnehmer, das Aufspüren der Hintergründe von *Feature_7*. Während sich ohne Berücksichtigung der Zeitpunkte, Trainingsergebnisse kaum bis gar nicht auf das Leaderboard übertragen ließen, besserte sich das deutlich durch die Gruppierung und Aufteilung der Datenpunkte nach den Werten von *Feature_7*. Auch im Bereich Feature-Engineering half das Wissen um dessen Bedeutung merklich weiter. Anstatt nach Mustern in den Zeitreihendaten zu suchen, erwies es sich als erfolgreicher, zeitpunktspezifische Charakteristika des gesamten Marktes für die Vorhersage heranzuziehen.

Doch trotz dieser Berücksichtigungen blieb das Vermeiden von Overfitting eine permanente Aufgabe. Im Rahmen des Wettbewerbs erwiesen sich hier insbesondere sehr einfache Modelle als überlegen gegenüber Komplexeren. Je flexibler ein Modell, desto eher neigte es zu Überanpassung und dem Lernen von zufälligen Umständen in den Daten. Daraus ergab sich somit auch, dass viele Prognosen auf den Leaderboards keine konsistenten Ergebnisse

abliefern. Gelöst werden konnte diese Problematik letztendlich am besten durch eine starke Regularisierung innerhalb der Ridge-Regression, zugunsten einer geringen Varianz.

Für weitere Kaggle-Wettbewerbe und andere Anwendungsgebiete lässt sich daher mitnehmen, dass unter gewissen Umständen einfache Modelle bei schwachen Signalen und viel Noise, komplexere Systeme übertreffen können. Wichtiger als ausgefeilte Algorithmen kann es sein, die Funktionsweise der Modelle zu verstehen und so jeden Subprozess von Datenaufbereitung bis Feature Selektion und Modelloptimierung an die Umstände anpassen zu können.

6 References

- Anderson, J. (2015). New Holiday Data from Winton. Retrieved from <https://www.kaggle.com/c/the-winton-stock-market-challenge/discussion/18006>
- Anderson, J. (2016). *The Winton Stock Market Challenge: Congratulations, Thoughts on the Problem*. Retrieved from <https://www.kaggle.com/c/the-winton-stock-market-challenge/discussion/18645>
- Bellman, R. (1961). *Adaptive control processes*. Princeton, New Jersey: Princeton University Press.
- Chen, T., & Guestrin, C. (2016). XGBoost. In B. Krishnapuram, M. Shah, A. Smola, C. Aggarwal, D. Shen, & R. Rastogi (Eds.), *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16* (pp. 785–794). New York, New York, USA: ACM Press. <https://doi.org/10.1145/2939672.2939785>
- Clarke, B., Fokoue, E., & Zhang, H. H. (2009). *Principles and Theory for Data Mining and Machine Learning*. New York, NY: Springer New York.
- Cont, R. (2001). Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative Finance*, 1(2), 223–236. <https://doi.org/10.1080/713665670>
- Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, 55(10), 78. <https://doi.org/10.1145/2347736.2347755>
- Forumsdiskussion. (2016). Solution Sharing. Retrieved from <https://www.kaggle.com/c/the-winton-stock-market-challenge/discussion/18584>
- Guo, Y. (2017). Introduction to Kaggle Kernels. Retrieved from <https://towardsdatascience.com/introduction-to-kaggle-kernels-2ad754ebf77>
- Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3, 1157–1182. Retrieved from <https://dl.acm.org/citation.cfm?id=944968>
- Hastie, T., Tibshirani, R., & Friedman, J. H. (2017). *The elements of statistical learning: Data mining, inference, and prediction* (Second edition, corrected at 12th printing). *Springer series in statistics*. New York, NY: Springer.
- Hoerl, A. E., & Kennard, R. W. (1970). Ridge Regression: Applications to Nonorthogonal Problems. *Technometrics*, 12(1), 69–82. <https://doi.org/10.1080/00401706.1970.10488635>
- Huijskens, T. (2018). *Why giving your algorithm ALL THE FEATURES does not always work*. PyData, London. Retrieved from https://www.youtube.com/watch?v=JsArBz46_3s&t=1848s
- Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3), 90–95. <https://doi.org/10.1109/MCSE.2007.55>

- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2017). *An introduction to statistical learning: With applications in R* (Corrected at 8th printing). *Springer texts in statistics*. New York, Heidelberg, Dordrecht, London: Springer.
- Kaggle Team. (2016). Winton Stock Market Challenge, Winner's Interview: 3rd place, Mendrika Ramarlina. Retrieved from <http://blog.kaggle.com/2016/02/12/winton-stock-market-challenge-winners-interview-3rd-place-mendrika-ramarlina/>
- Kaggle Team. (2017a). Two Sigma Financial Modeling Challenge, Winner's Interview: 2nd Place, Nima Shahbazi, Chahhou Mohamed. Retrieved from <http://blog.kaggle.com/2017/05/25/two-sigma-financial-modeling-challenge-winners-interview-2nd-place-nima-shahbazi-chahhou-mohamed/>
- Kaggle Team. (2017b). Two Sigma Financial Modeling Code Competition, 5th Place Winners' Interview: Team Best Fitting | Bestfitting, Zero, & CircleCircle. Retrieved from <http://blog.kaggle.com/2017/05/11/two-sigma-financial-modeling-code-competition-5th-place-winners-interview-team-best-fitting-bestfitting-zero-circlecircle/>
- Kaiser, J. (2014). Dealing with Missing Values in Data. *Journal of Systems Integration*, 5(1), 42–51. Retrieved from <http://si-journal.org/index.php/JSI/article/viewFile/178/255>
- Kakushadze, Z. (2016). 101 Formulaic Alphas. Retrieved from <http://arxiv.org/pdf/1601.00991v3>
- Mansfield, E. R., & Helms, B. P. (1982). Detecting Multicollinearity. *The American Statistician*, 36(3), 158. <https://doi.org/10.2307/2683167>
- Marr, B. (2016). *Big data in practice: How 45 successful companies used big data analytics to deliver extraordinary results*. Chichester, West Sussex: Wiley.
- Matsuki, K., Kuperman, V., & van Dyke, J. A. (2016). The Random Forests statistical technique: An examination of its value for the study of reading. *Scientific Studies of Reading : the Official Journal of the Society for the Scientific Study of Reading*, 20(1), 20–33. <https://doi.org/10.1080/10888438.2015.1107073>
- McDonald, G. C. (2009). Ridge regression. *Wiley Interdisciplinary Reviews: Computational Statistics*, 1(1), 93–100. <https://doi.org/10.1002/wics.14>
- McKinney, W. Data Structures for Statistical Computing in Python. In *Proc. of the 9th python in science conf. (scipy 2010)* (pp. 51–56). Retrieved from <https://conference.scipy.org/proceedings/scipy2010/pdfs/mckinney.pdf>
- Melkumova, L. E., & Shatskikh, S.Y. (2017). Comparing Ridge and LASSO estimators for data analysis. *Procedia Engineering*, 201, 746–755. <https://doi.org/10.1016/j.proeng.2017.09.615>
- Nanny, W. (1975). Beobachtungen zur Ridge-Regression. *Jahrbücher Für Nationalökonomie Und Statistik*, 189(3-4). <https://doi.org/10.1515/jbnst-1975-3-411>

- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . .
 Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.
- Perez, F., & Granger, B. E. (2007). IPython: A System for Interactive Scientific Computing. *Computing in Science & Engineering*, 9(3), 21–29. <https://doi.org/10.1109/MCSE.2007.53>
- Sarkar, D., Bali, R., & Sharma, T. (2018a). Feature Engineering and Selection. In D. Sarkar, R. Bali, & T. Sharma (Eds.), *Practical Machine Learning with Python* (pp. 177–253). Berkeley, CA: Apress. https://doi.org/10.1007/978-1-4842-3207-1_4
- Sarkar, D., Bali, R., & Sharma, T. (Eds.). (2018b). *Practical Machine Learning with Python*. Berkeley, CA: Apress.
- Schapire, R. E. (2013). Explaining AdaBoost. In B. Schölkopf, Z. Luo, & V. Vovk (Eds.), *Empirical Inference* (pp. 37–52). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-41136-6_5
- Schmid, F., & Trede, M. (2006). *Finanzmarktstatistik*. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg. Retrieved from <http://dx.doi.org/10.1007/3-540-29795-2>
- Shah, S., & Goldbloom, A. (2013). Kaggle's Anthony Goldbloom Is Building A New Kind Of Marketplace. Retrieved from <https://techcrunch.com/2013/01/10/in-the-studio-kaggles-anthony-goldbloom-is-building-a-new-kind-of-marketplace/?guccounter=1>
- Swamynathan, M. (2017). *Mastering Machine Learning with Python in Six Steps: A Practical Implementation Guide to Predictive Data Analytics Using Python*. Berkeley, CA: Apress; Imprint.
- Thomas, K., Benjamin, R.-K., Fernando, P., Brian, G., Matthias, B., Jonathan, F., . . .
 Duchesnay, E. (2016). Jupyter Notebooks: a publishing format for reproducible computational workflows. In F. Loizides & B. Schmidt (Eds.), *Positioning and power in academic publishing: Players, agents and agendas : proceedings of the 20th International Conference on Electronic Publishing* (pp. 87–90). Amsterdam: Ios Press, Inc.
- Winton Capital. (2015a). Data. Retrieved from <https://www.kaggle.com/c/the-winton-stock-market-challenge/data>
- Winton Capital. (2015b). Overview. Retrieved from <https://www.kaggle.com/c/the-winton-stock-market-challenge>
- Winton Capital. (2016). Leaderboard. Retrieved from <https://www.kaggle.com/c/the-winton-stock-market-challenge/leaderboard>

7 Anhang

7.1 Anhang 1: Das Winton Statement

Das abschließende Statement von Winton zu Stock-Market-Challenge:

„Hi all. Thanks for all your efforts competing in what was a hard data challenge, we hope you found the time you spent on the problem interesting. We're currently waiting for and looking at code to verify the top solutions, but in the meantime for your interest we thought we would say a little about the problem preparation and some remarks on approaches to solutions.

But first we would like to congratulate those who finished in the prize money. We're excited to see what the best solutions are as the write-ups and code comes in. For some of the top solutions we have a good idea of their approach; no doubt these methods will be revealed in time. In any case it is a notable achievement to have beaten out all the others, and all the hard work should be lauded.

In explaining what successful modelling strategies might have looked like, it is interesting to start by noting that a number of people remarked that cross-validation "didn't work". It's not obvious, but it does work, just not in a classical manner. Feature_7 is the key, as it is a date identifier. Stocks tend to be correlated on the same day, and so the classical approach to cross-validation would have simply allowed your machine learning method to learn patterns of exactly what happened on certain days. These would not generalise well to other days, hence many commenting of good scores in the training set which do poorly on the public leaderboard. By splitting up or re-sampling the training set by days you would have ended up with useful cross-validation sets.

It is slightly tricky to find this given the features were not identified, but for this date feature there are a number of clues. Firstly, it is one of only two features with no missing values. Secondly, if you run the data through some out-of-the-box machine learning method on the training set, it will likely come up as the most important feature by far. Thirdly, the feature is a categorical variable with different categories in training and test sets. It is then reasonable to infer that this is a "group" identifier and that the training and test samples may have been obtained by sampling those groups, instead of single cases. Indeed, somebody found this last clue and posted it on the forum, but unfortunately their conclusion was to throw it out as a feature.

Finding the day feature would have notably helped you in your prediction, but there was plenty to find without that. For obvious reasons we don't want to say exactly what we think is there, but a few insights were needed to do useful things with the data. It is easy to see that trying to predict each one-minute return is very hard, and some approach to binning or trying to predict overall features is required. Watching the forums through the competition

suggested a few people had issues with this. With generalisation very difficult, most things you could add to deal with outliers and make things more robust would end up helping. Being able to predict temporal and cross-sectional volatility, which are both easier to predict than actual returns, would also give you a clue as to where the big returns might be, which is where the largest reward is in terms of score improvement.

With respect to the other features, we won't say exactly what they were, but they are a mix of fundamental and technical financial measures, past performance, and a few irrelevant features. We kept them anonymised because we didn't want anyone with a financial background to have an a priori advantage, and "know" what to do with them. Even still, we expect that only a few would have been of help and only very slightly; Feature_5, the other feature with no missing data, would have been a good one to look at.

A few quick answers to other questions that have come up. Firstly, the data is real financial data, obviously transformed carefully and slightly noised, but for confidentiality we can't say any more about where or when it is from. The weights were, among other things, meant to give some appreciation of trading costs in a very simple way. Similarly, the WMAE evaluation metric was not intended to be perfect, but since the modelling part of the challenge is so hard, we wanted to make the rest of the framework conceptually as easy as possible. In the same theme, the returns were simple logarithmic returns.

A number of people have asked what our own score on the competition is, but we don't think it is useful to give a precise number. Although we did work on it considerably when preparing the problem, it hasn't had the attention we would give a usual research problem, so we wouldn't want to call our score our "best" effort. We also knew how the problem was constructed, what the data sources were, and have years of collective experience with these types of problems, so it might be hard to convince you all our score is fair as a benchmark (or, indeed, at all). Looking at the community's scores, it looks like the top scorers have made some progress, but we still think there is some mileage that can be gotten from the data.

As we said in the problem description, we hope this problem should give a good flavour of the kind of signal-to-noise and overfitting issues we deal with on a consistent basis. We typically work in a relatively standard back-testing framework, but currently it is not really possible to put this kind of problem up on Kaggle, as the training and test set are essentially the same. Data work is also a big part of what we do at Winton. In the end we thought the contest was difficult enough without asking you to comb through unclean data and find all the errors, so there were not too many data quality issues, but training your model on bad data will likely mean bad predictions. It's teasing that little bit of signal out from all of the noise, as

well as trying to determine if you believe what you have found, that makes the problem interesting and compelling to work on.“ (Anderson, 2016)

7.2 Anhang 2: Tabelle der trainierten Modelle

Eine Übersicht über die Modelle, die innerhalb des Trainings-Sets trainiert und ausgewertet wurden:

Algorithm	Feature-Set	Target	Train-Time [s]	Zero-Score	Train-Score (WMAE)	Validation-Score (MAE)
ExtraTreesRegressor	Core	Ret_MinutesFut	10,373	1773,9244	1774,08599	0,0048563
ExtraTreesRegressor	corr10	Ret_MinutesFut	9,930	1773,9244	1774,08572	0,0048543
ExtraTreesRegressor	corr20	Ret_MinutesFut	11,199	1773,9244	1774,07304	0,0048549
ExtraTreesRegressor	corr30	Ret_MinutesFut	10,262	1773,9244	1774,07725	0,0048551
ExtraTreesRegressor	mic10	Ret_MinutesFut	9,879	1773,9244	1774,07687	0,0048561
ExtraTreesRegressor	mic20	Ret_MinutesFut	11,461	1773,9244	1774,07424	0,0048560
ExtraTreesRegressor	mic30	Ret_MinutesFut	9,741	1773,9244	1774,07706	0,0048552
ExtraTreesRegressor	micpoly15	Ret_MinutesFut	9,778	1773,9244	1774,07918	0,0048563
ExtraTreesRegressor	polycorr15_1	Ret_MinutesFut	9,893	1773,9244	1774,08335	0,0048563
ExtraTreesRegressor	polycorr15_2	Ret_MinutesFut	10,995	1773,9244	1774,0798	0,0048555
ExtraTreesRegressor	Core	Ret_PlusOne	10,195	1773,9244	1773,80549	0,0157703
ExtraTreesRegressor	corr10	Ret_PlusOne	9,797	1773,9244	1773,77388	0,0157694
ExtraTreesRegressor	corr20	Ret_PlusOne	9,681	1773,9244	1773,76236	0,0157693
ExtraTreesRegressor	corr30	Ret_PlusOne	9,871	1773,9244	1773,67233	0,0157690
ExtraTreesRegressor	mic10	Ret_PlusOne	9,540	1773,9244	1773,83925	0,0157701
ExtraTreesRegressor	mic20	Ret_PlusOne	9,587	1773,9244	1773,82747	0,0157704
ExtraTreesRegressor	mic30	Ret_PlusOne	9,856	1773,9244	1773,75181	0,0157705
ExtraTreesRegressor	micpoly15	Ret_PlusOne	9,756	1773,9244	1773,81635	0,0157705
ExtraTreesRegressor	polycorr15_1	Ret_PlusOne	9,609	1773,9244	1773,8421	0,0157704
ExtraTreesRegressor	polycorr15_2	Ret_PlusOne	9,670	1773,9244	1773,69519	0,0157677
ExtraTreesRegressor	Core	Ret_PlusTwo	11,086	1773,9244	1773,91493	0,0151928
ExtraTreesRegressor	corr10	Ret_PlusTwo	9,656	1773,9244	1773,81565	0,0151918
ExtraTreesRegressor	corr20	Ret_PlusTwo	9,609	1773,9244	1773,8467	0,0151914
ExtraTreesRegressor	corr30	Ret_PlusTwo	9,794	1773,9244	1773,82202	0,0151917
ExtraTreesRegressor	mic10	Ret_PlusTwo	9,594	1773,9244	1773,88892	0,0151917
ExtraTreesRegressor	mic20	Ret_PlusTwo	9,787	1773,9244	1773,81667	0,0151916
ExtraTreesRegressor	mic30	Ret_PlusTwo	9,756	1773,9244	1773,83446	0,0151921
ExtraTreesRegressor	micpoly15	Ret_PlusTwo	9,609	1773,9244	1773,81637	0,0151917
ExtraTreesRegressor	polycorr15_1	Ret_PlusTwo	9,877	1773,9244	1773,92339	0,0151923
ExtraTreesRegressor	polycorr15_2	Ret_PlusTwo	9,903	1773,9244	1773,68146	0,0151919
GradientBoostingRegressor	Core	Ret_MinutesFut	372,628	1773,9244	1773,9226	0,0048448
GradientBoostingRegressor	corr10	Ret_MinutesFut	179,818	1773,9244	1773,78802	0,0048366
GradientBoostingRegressor	corr20	Ret_MinutesFut	226,894	1773,9244	1773,78439	0,0048389
GradientBoostingRegressor	corr30	Ret_MinutesFut	284,804	1773,9244	1773,80666	0,0048411
GradientBoostingRegressor	mic10	Ret_MinutesFut	166,582	1773,9244	1773,94474	0,0048456
GradientBoostingRegressor	mic20	Ret_MinutesFut	257,163	1773,9244	1773,80861	0,0048411

Algorithm	Feature-Set	Target	Train-Time [s]	Zero-Score	Train-Score (WMAE)	Validation-Score (MAE)
GradientBoostingRegressor	mic30	Ret_MinutesFut	361,501	1773,9244	1773,77841	0,0048366
GradientBoostingRegressor	micpoly15	Ret_MinutesFut	184,599	1773,9244	1773,8016	0,0048588
GradientBoostingRegressor	polycorr15_1	Ret_MinutesFut	209,974	1773,9244	1773,92552	0,0048438
GradientBoostingRegressor	polycorr15_2	Ret_MinutesFut	203,433	1773,9244	1773,82858	0,0048459
GradientBoostingRegressor	Core	Ret_PlusOne	337,349	1773,9244	1773,58159	0,0157561
GradientBoostingRegressor	corr10	Ret_PlusOne	164,507	1773,9244	1772,58223	0,0157576
GradientBoostingRegressor	corr20	Ret_PlusOne	239,980	1773,9244	1772,54671	0,0157576
GradientBoostingRegressor	corr30	Ret_PlusOne	307,323	1773,9244	1772,55325	0,0157566
GradientBoostingRegressor	mic10	Ret_PlusOne	164,784	1773,9244	1773,59517	0,0157705
GradientBoostingRegressor	mic20	Ret_PlusOne	228,828	1773,9244	1771,06627	0,0157638
GradientBoostingRegressor	mic30	Ret_PlusOne	275,670	1773,9244	1770,81437	0,0157643
GradientBoostingRegressor	micpoly15	Ret_PlusOne	186,180	1773,9244	1771,02874	0,0157600
GradientBoostingRegressor	polycorr15_1	Ret_PlusOne	196,244	1773,9244	1773,30275	0,0157551
GradientBoostingRegressor	polycorr15_2	Ret_PlusOne	173,725	1773,9244	1771,79362	0,0157693
GradientBoostingRegressor	Core	Ret_PlusTwo	341,887	1773,9244	1773,62753	0,0151748
GradientBoostingRegressor	corr10	Ret_PlusTwo	176,695	1773,9244	1771,50038	0,0151814
GradientBoostingRegressor	corr20	Ret_PlusTwo	245,882	1773,9244	1771,88612	0,0151806
GradientBoostingRegressor	corr30	Ret_PlusTwo	311,545	1773,9244	1771,81927	0,0151823
GradientBoostingRegressor	mic10	Ret_PlusTwo	169,021	1773,9244	1773,52352	0,0151808
GradientBoostingRegressor	mic20	Ret_PlusTwo	221,505	1773,9244	1772,12563	0,0151889
GradientBoostingRegressor	mic30	Ret_PlusTwo	274,095	1773,9244	1771,59204	0,0151945
GradientBoostingRegressor	micpoly15	Ret_PlusTwo	192,493	1773,9244	1771,63038	0,0151774
GradientBoostingRegressor	polycorr15_1	Ret_PlusTwo	199,222	1773,9244	1773,29241	0,0151701
GradientBoostingRegressor	polycorr15_2	Ret_PlusTwo	184,834	1773,9244	1770,71726	0,0151609
HuberRegressor	corr10	Ret_MinutesFut	37,410	1773,9244	1774,76017	0,0048367
HuberRegressor	corr20	Ret_MinutesFut	69,870	1773,9244	1774,81313	0,0048458
HuberRegressor	corr30	Ret_MinutesFut	211,435	1773,9244	1774,85679	0,0048468
HuberRegressor	mic10	Ret_MinutesFut	5,566	1773,9244	1773,92562	0,0048472
HuberRegressor	mic20	Ret_MinutesFut	10,413	1773,9244	1774,25283	0,0048459
HuberRegressor	mic30	Ret_MinutesFut	57,215	1773,9244	1774,73117	0,0048603
HuberRegressor	micpoly15	Ret_MinutesFut	4,921	1773,9244	1773,9256	0,0048491
HuberRegressor	polycorr15_1	Ret_MinutesFut	13,140	1773,9244	1774,09345	0,0048493
HuberRegressor	polycorr15_2	Ret_MinutesFut	14,388	1773,9244	1774,30927	0,0048409
HuberRegressor	Core	Ret_PlusOne	883,253	1773,9244	1773,2304	0,0157955
HuberRegressor	corr10	Ret_PlusOne	29,544	1773,9244	1773,40607	0,0157518
HuberRegressor	corr20	Ret_PlusOne	53,920	1773,9244	1773,37159	0,0157522
HuberRegressor	corr30	Ret_PlusOne	95,658	1773,9244	1773,19515	0,0157512
HuberRegressor	mic10	Ret_PlusOne	15,431	1773,9244	1773,84101	0,0157579
HuberRegressor	mic20	Ret_PlusOne	43,130	1773,9244	1773,71676	0,0157670
HuberRegressor	mic30	Ret_PlusOne	88,512	1773,9244	1773,473	0,0158023
HuberRegressor	micpoly15	Ret_PlusOne	27,346	1773,9244	1773,72944	0,0157596
HuberRegressor	polycorr15_1	Ret_PlusOne	38,396	1773,9244	1773,80348	0,0157651
HuberRegressor	polycorr15_2	Ret_PlusOne	19,865	1773,9244	1773,43832	0,0157581
HuberRegressor	Core	Ret_PlusTwo	888,553	1773,9244	1773,67349	0,0152222

Algorithm	Feature-Set	Target	Train-Time [s]	Zero-Score	Train-Score (WMAE)	Validation-Score (MAE)
HuberRegressor	corr10	Ret_PlusTwo	42,699	1773,9244	1773,45409	0,0151849
HuberRegressor	corr20	Ret_PlusTwo	79,234	1773,9244	1773,39851	0,0151913
HuberRegressor	corr30	Ret_PlusTwo	52,544	1773,9244	1773,62349	0,0152012
HuberRegressor	mic10	Ret_PlusTwo	15,160	1773,9244	1773,83897	0,0151762
HuberRegressor	mic20	Ret_PlusTwo	39,660	1773,9244	1773,69437	0,0151873
HuberRegressor	mic30	Ret_PlusTwo	83,228	1773,9244	1773,60616	0,0152265
HuberRegressor	micpoly15	Ret_PlusTwo	28,802	1773,9244	1773,61454	0,0151767
HuberRegressor	polycorr15_1	Ret_PlusTwo	35,614	1773,9244	1773,74594	0,0151833
HuberRegressor	polycorr15_2	Ret_PlusTwo	22,096	1773,9244	1773,26443	0,0151685
Ridge	Core	Ret_MinutesFut	3,155	1773,9244	1774,10135	0,0048542
Ridge	corr10	Ret_MinutesFut	1,146	1773,9244	1774,15334	0,0048467
Ridge	corr20	Ret_MinutesFut	2,163	1773,9244	1774,18341	0,0048462
Ridge	corr30	Ret_MinutesFut	2,159	1773,9244	1774,19347	0,0048457
Ridge	mic10	Ret_MinutesFut	1,144	1773,9244	1774,08995	0,0048541
Ridge	mic20	Ret_MinutesFut	1,745	1773,9244	1774,07488	0,0048545
Ridge	mic30	Ret_MinutesFut	2,134	1773,9244	1774,10412	0,0048511
Ridge	micpoly15	Ret_MinutesFut	1,404	1773,9244	1774,07238	0,0048554
Ridge	polycorr15_1	Ret_MinutesFut	1,303	1773,9244	1774,082	0,0048564
Ridge	polycorr15_2	Ret_MinutesFut	1,628	1773,9244	1774,0817	0,0048592
Ridge	rfe10	Ret_MinutesFut	1,125	1773,9244	1774,08021	0,0048549
Ridge	rfe20	Ret_MinutesFut	1,473	1773,9244	1774,07789	0,0048547
Ridge	rfe30	Ret_MinutesFut	2,103	1773,9244	1774,07556	0,0048545
Ridge	Core	Ret_PlusOne	6,616	1773,9244	1773,60775	0,0157670
Ridge	corr10	Ret_PlusOne	2,963	1773,9244	1773,15082	0,0157564
Ridge	corr20	Ret_PlusOne	4,222	1773,9244	1773,23601	0,0157571
Ridge	corr30	Ret_PlusOne	5,659	1773,9244	1773,07735	0,0157546
Ridge	mic10	Ret_PlusOne	2,918	1773,9244	1773,80984	0,0157875
Ridge	mic20	Ret_PlusOne	4,276	1773,9244	1773,79498	0,0157982
Ridge	mic30	Ret_PlusOne	5,644	1773,9244	1773,504	0,0157962
Ridge	micpoly15	Ret_PlusOne	3,752	1773,9244	1773,82098	0,0157995
Ridge	polycorr15_1	Ret_PlusOne	3,520	1773,9244	1773,57497	0,0157686
Ridge	polycorr15_2	Ret_PlusOne	4,057	1773,9244	1772,99521	0,0157564
Ridge	rfe10	Ret_PlusOne	2,824	1773,9244	1773,47915	0,0157685
Ridge	rfe20	Ret_PlusOne	4,305	1773,9244	1773,37371	0,0157717
Ridge	rfe30	Ret_PlusOne	5,596	1773,9244	1773,16997	0,0157739
Ridge	Core	Ret_PlusTwo	6,964	1773,9244	1773,91217	0,0151942
Ridge	corr10	Ret_PlusTwo	3,003	1773,9244	1773,79768	0,0151961
Ridge	corr20	Ret_PlusTwo	4,493	1773,9244	1773,82685	0,0152107
Ridge	corr30	Ret_PlusTwo	5,884	1773,9244	1773,79754	0,0152180
Ridge	mic10	Ret_PlusTwo	3,065	1773,9244	1773,9824	0,0152085
Ridge	mic20	Ret_PlusTwo	4,347	1773,9244	1773,89573	0,0152187
Ridge	mic30	Ret_PlusTwo	5,914	1773,9244	1773,8726	0,0152278
Ridge	micpoly15	Ret_PlusTwo	3,841	1773,9244	1773,61524	0,0152076
Ridge	polycorr15_1	Ret_PlusTwo	4,502	1773,9244	1773,86816	0,0151985

Algorithm	Feature-Set	Target	Train-Time [s]	Zero-Score	Train-Score (WMAE)	Validation-Score (MAE)
Ridge	polycorr15_2	Ret_PlusTwo	3,620	1773,9244	1772,99782	0,0151819
Ridge	rfe10	Ret_PlusTwo	3,122	1773,9244	1773,82924	0,0152028
Ridge	rfe20	Ret_PlusTwo	4,438	1773,9244	1773,84563	0,0152125
Ridge	rfe30	Ret_PlusTwo	6,115	1773,9244	1773,84138	0,0152205

7.3 Anhang 3: Feature Bedeutung

Eine Übersicht über die Merkmale, die im finalen Modell enthalten waren mit entsprechendem Koeffizienten.

	Name	Koeffizient	Inhalt
Ret_PlusOne	interaction_1	-5,10	Interaktion zwischen dem MAD der 120 Minutenrenditen und der Periodenrendite der letzten 5 Minuten
	interaction_2	-4,98	Interaktion zwischen dem MAD der letzten 15 Minutenrenditen und der Periodenrendite der letzten 10 Minuten
	MAD_120	0,00	Mittlere absolute Abweichung der Minutenrenditen
	MAD_last_60	0,01	Mittlere absolute Abweichung der Minutenrenditen eins bis 60
	MAD_first_60	-0,03	Mittlere absolute Abweichung der Minutenrenditen eins bis 60
	MAD_last_30	0,06	Mittlere absolute Abweichung der Minutenrenditen 91 bis 120
	MAD_dif_60_120	0,25	Differenz der MAD zwischen den 120 Minuten und den letzten 60 Minuten
	MAD_dif_15_120	0,23	Differenz der MAD zwischen den 120 Minuten und den letzten 15 Minuten
	smoothed_minute_mean	-0,83	Mittelwert der geglätteten Minutenrenditen
	smoothed_minute_mad	0,07	Mittlere absolute Abweichung der geglätteten Minutenrenditen
	smoothed_minute_median	0,07	Median der geglätteten Minutenrenditen
	smoothed_minute_std	0,08	Standardabweichung der geglätteten Minutenrenditen
	smoothed_minute_var	21,89	Varianz der geglätteten Minutenrenditen
	smoothed_minute_sem	0,82	Standard error of mean der geglätteten Minutenrenditen
	smoothed_minute_25_quantile	-0,12	25%-Quantil der geglätteten Minutenrenditen
	smoothed_minute_75_quantile	-0,14	75%-Quantil der geglätteten Minutenrenditen
	minute_mean	-0,83	Mittelwert der Minutenrenditen
	minute_mad	0,00	Mittlere absolute Abweichung der Minutenrenditen
	minute_median	0,00	Median der Minutenrenditen
	minute_var	5,98	Varianz der Minutenrenditen
	minute_sem	0,20	Standard error of mean der Minutenrenditen
	minute_25_quantile	0,04	25%-Quantil der Minutenrenditen
	minute_75_quantile	-0,09	75%-Quantil der Minutenrenditen
	fet_7_Minute_Mean	-1,92	Nach Feature_7 gruppierter Mittelwert der Minutenrenditen
	fet_7_Minute_MAD	-0,90	Nach Feature_7 gruppierte mittlere absolute Abweichung der Minutenrenditen
	interaction_3	-0,45	Interaktion zwischen der Periodenrendite über die 120 Minuten und dem MAD der Minutenrenditen gruppiert nach Feature_7
	interaction_5	-1,98	Interaktion zwischen der Periodenrendite über die 120 Minuten und dem MAD der Minutenrenditen

	Name	Koeffizient	Inhalt
	interaction_6	-49,18	Interaktion zwischen dem Mittelwert der geglätteten Minutenrenditen und der nach Feature_5 gruppierten MAD $X.smoothed_minute_mean * X.grouped_mad_fet_5$
	interaction_7	-95,12	Interaktion zwischen der Periodenrendite über 120 Minuten und dem MAD der Minutenrenditen gruppiert nach Feature_7
	interaction_8	-20292,61	Interaktion der nach Feature_7 gruppierten MAD und der nach Feature_7 gruppierten Mittelwerte der Minutenrenditen
Ret_PlusTwo	x0 x20	-552,21	Interaktion zwischen dem MAE der 120 Minutenrenditen, der Periodenrendite der letzten 5 Minuten und dem Mittelwert der Periodenrendite gruppiert nach Zeitpunkten (Feature_7)
	x0 x18	-62951,93	Interaktion zwischen dem MAD der 120 Minutenrenditen, der Periodenrendite der letzten 5 Minuten und dem Mittelwert der Minuten gruppiert nach Zeitpunkten
	x1 x8	97,67	Interaktion zwischen dem MAD der letzten 15 Minutenrenditen, der Periodenrendite der letzten 10 Minuten und dem Mittelwert der vergangenen Tagesrendite Ret_MinusOne gruppiert nach Zeitpunkten
	x11 x19	134,53	Interaktion zwischen der nach Feature_5 gruppierte MAD von Ret_MinusOne und nach Zeitpunkt gruppierte MAD der Minutenrenditen
	x11 x21	1,18	Interaktion zwischen der MAD der nach Feature_5 gruppierten Ret_MinusOne und der MAD der Periodenrendite der Minuten zu einem Zeitpunkt
	x3 x21	0,00	Interaktion zwischen der absoluten Differenz der vergangenen Tagesrenditen und der MAD der Periodenrendite der Minuten zu einem Zeitpunkt
	x3 x19	0,50	Interaktion zwischen der absoluten Differenz der vergangenen Tagesrenditen und der MAD der Minutenrenditen zu einem Zeitpunkt
	x19 x21	141,31	Interaktion zwischen der MAD der Periodenrendite der Minuten zu einem Zeitpunkt und der MAD der Minutenrenditen zu dem gleichen Zeitpunkt
	x19	2,39	MAD der Minutenrenditen an einem Zeitpunkt
	x21^2	1,24	MAD der Periodenrendite der Minuten zu einem Zeitpunkt (quadriert)
	x21	0,02	MAD der Periodenrendite der Minuten zu einem Zeitpunkt
	x19^2	16109,17	Nach Zeitpunkt gruppierte MAD der Minutenrenditen quadriert
	x7 x18	-739,48	Interaktion zwischen der Summe der letzten beiden Minutenrenditen und dem Mittelwert der Minutenrenditen zu einem Zeitpunkt
	x7 x20	-6,49	Interaktion zwischen der Summe der letzten beiden Minutenrenditen und der Summer der Minutenrenditen zu einem Zeitpunkt
	x20^2	1,22	Summe der Minutenrenditen zu einem Zeitpunkt (quadriert)

8 Eidesstattliche Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe. Die Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt und von dieser als Teil einer Prüfungsleistung angenommen. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Nürnberg, den 30. August 2018

Vorname Name