Exposé für die Bachelorarbeit: Anwendung von Ensembles auf die Zeitreihenklassifikatoren WEASEL und BOSS

Florian Heinrichs

Juli 2020

Inhaltsverzeichnis

1 Einleitung

Als Zeitreihe bezeichnet man eine Folge von zeitlich geordneten Werten. Sie werden unter anderem in EKG's oder bei Temperaturaufzeichnungen gemessen. Über die Klassifizierung von Zeitreihen kann man wichtige Erkenntnisse gewinnen. Zum Beispiel lässt sich anhand des EKG's eines Patientens eine Vorhersage zu dessen Gesundheitszustand treffen, indem wir aus historischen Daten gesunder und kranker Patienten stochastische Verteilungen berechnen. Dabei kann man Klassifikations-Algorithmen grob danach unterscheiden, ob sie die gesamte Zeitreihe oder nur Teilsequenzen für ihre Vorhersage verwenden [1]. Ich möchte mich in dieser Arbeit nur auf sogenannte Wörterbuch-Zeitreihenklassifikatoren wie BOSS [14] und WEASEL [16] konzentrieren, welche der zweiten Kategorie

zugeordnet werden. Wörterbuchklassifikatoren transformieren reelwertige Teilsequenzen in ein Wort über einem Alphabet Σ und lernen dann auf der Menge der Wörter ihren entsprechenden Klassifikator.

Klassifikatoren werden meist anhand des Verhältnisses von richtig und falsch klassifizierten Samples bewertet, oft repräsentiert durch die Anzahl der Fehler in Prozent. Dabei betrachten wir die Trainings- und Testmenge getrennt voneinander. Auf der Trainingsmenge wird das Modell gelernt, während die Testmenge die Generalisierbarkeit des Modells bewertet. Dabei interessiert uns der Zusammenhang zwischen der Fehlerrate auf den Test- und Trainingsdaten, da dieser Rückschlüsse über die Stärken und Schwächen eines Modells, hinsichtlich seiner Generalisierbarkeit (Over- und Underfitting), zulässt.

Eine Möglichkeit um die Genauigkeit von Klassifikatoren zu verbessern, ist die Verwendung von sogenannten *Ensembles* [19]. Das sind Methoden, die die Vorhersagen mehrerer Klassifikatoren zu einer einzelnen kombinieren. Dabei kann man zwischen den Methoden *Stacking* [18], *Boosting* [6] und *Bagging* [2] unterscheiden. Je nachdem ob der Klassifikator einen hohen Bias oder eine hohe Varianz hat, empfiehlt sich die Anwendung der entsprechenden Methoden.

Bei Zeitreihen kombiniert man häufig verschiedene Klassifikatoren, die unterschiedlichen Repräsentationen von Zeitreihen benutzen, mithilfe einer gewichteten Mehrheitswahl [8]. Das Ziel dieser Arbeit ist es, ein Ensemble über mehrere Instanzen des BOSS- oder WEASEL-Modells zu bilden, wobei diese jeweils mit verschiedenen Hyperparametern trainiert werden. Diese werden dem Modell bei dem Aufruf übergeben und beeinflussen unter anderem das Over- oder Underfitting des Modells. Hyperparameter für BOSS und WEASEL sind zum Beispiel die Wortlänge oderr die Alphabetgröße.

Die Arbeit soll hauptsächlich in Python realisiert werden. Der Vorteil von Python ist, dass bereits Bibliotheken existieren, die die hier zu untersuchenden Ensemble-Methoden implementiert haben. Die wichtigste ML-Bibliothek ist *scikit-learn* [13], mit der ich bereits Erfahrung gesammelt habe, weshalb ich diese Bibliothek für die Anwendung von Ensembles benutzen möchte. Im ersten Schritt sollen verschiedene Python-Implementierungen von BOSS und WEASEL mit der Java-Implementierung hinsichtlich ihrer Genauigkeit und Laufzeit verglichen werden. Drei Kandidaten hierfür sind *sktime* [9], *pyts* [5], und *SFA_Python* [17]. Hierbei genügt es, eine Python-Implementierung zu finden, die ähnliche Genauigkeiten liefert, da ich überprüfen möchte, ob die Anwendung von Ensembles grundsätzlich zu einer Verbesserung führt. Für alle Tests benutze ich das *UCR-Archiv* [4], welches 85 Datensätze, bestehend aus Trainings- und Testdaten, zur Verfügung stellt. Im zweiten Schritt identifiziere ich für BOSS und WEASEL ob sie ein High-Bias oder High-Variance Modell sind, da davon die Wahl der Ensemble-Methode abhängt, welche ich danach anwenden werde. Bei einem High-Bias-Modell könnte man zum Beispiel Boosting verwenden, bei einem High-Variance-Modell unter anderem Bagging. Zuletzt möchte ich die Idee hinter einer komplexeren Ensemble-Methode wie dem *Proximity Forest* [10] anwenden und seinen Einfluss auf die Genauigkeit beurteilen. Hierbei wäre vorher zu evaluieren, ob die Python- oder Java-Implementierung besser geeignet ist. Der Vorteil von Java wäre, dass wir hier sowohl die ursprüngliche Implementie-

rung vom Proximity Forest, als auch von BOSS/WEASEL verwenden könnten. Allerdings lassen sich die Ergebnisse des Proximity Forest dann nicht mit denen der anderen, in Python implementierten, Methoden vergleichen.

2 Grundlagen

Im folgenden erkläre ich die in dieser Arbeit verwendeten Methoden und Algorithmen.

2.1 SFA

Die *Symbolic Fourier Approximation* [15] ist ein symbolisches Dimensionsreduktionsverfahren. Bei der SFA reduzieren wir die Anzahl der Features beziehungsweise Messwerte der einzelnen Zeitreihen und diskretisieren den Wertebereich. Dafür benutzen wir die diskrete Fourier-Transformation, bei der das Ursprungssignal durch eine Linearkombination der ersten sinusoidalen Folgen angenähert wird, wobei die Folgen von den entsprechenden Fourierkoeffizienten repräsentiert werden. Danach diskretisieren wir die Fourierkoeffizienten mithilfe eines festen Alphabets Σ. Jedem Buchstaben aus diesem Alphabet wird ein Wertebereich zugewiesen. Am Ende erhalten wir ein Wort über dem Alphabet Σ, dessen Länge geringer ist als die Länge der ursprünglichen Zeitreihen. Hyperparameter für SFA sind die Wortlänge, Alphabetgröße und Diskretisierungsmethode.

2.2 BOSS

BOSS ist eine Wörterbuchrepräsentation einer Zeitreihe. Bei dem *Bag-Of-Sfa-Symbols* [14] Modell wird die Repräsentation durch SFA mit dem bag-of-words Modell verbunden. Zuerst schieben wir ein Fenster mit Länge w über die Zeitreihe und transfomieren jede enthaltene Teilsequenz mittels SFA in ein Wort. Dabei zählen wir wie oft jedes Wort vorkommt und erstellen ein Histogramm über alle Worthäufigkeiten. Als finales Modell verwenden wir Histogramme über verschiedene Fenstergrößen w. Neue Samples werden zur Klassifikation ebenfalls in Histogramme transformiert und ihre Klasse mittels 1-Nearest-Neighbor vorhergesagt, wobei wir als Distanz die Differenz der Worthäufigkeiten benutzen. BOSS führt zusätzlich die Teilsequenzlänge und Teilsequenznormalisierung als Hyperparameter ein.

2.3 WEASEL

Das *Word ExtrAction for timeSEries cLassification* [16] Modell ist ebenfalls eine Wörterbuchrepräsentation von Zeitreihen, welche auf dem *bag-of-patterns* [7] Modell basiert. Allerdings extrahieren wir hier Fenster mit mehreren unterschiedlichen Längen und beachten aufeinanderfolgende Fenster (Bigramme). Im Anschluss vereinigen wir die Histogramme und wenden noch eine statistische Feature-Wahl an. Diese sorgt für eine stark diskriminierende Featuremenge, indem irrelevante Features entfernt werden, sodass Logistische Regression mit

Ridge Regularisierung als Klassifikator benutzt werden kann. WEASEL führt drei zusätzliche Hyperparameter ein: Teilsequenzlänge, Normierung und Bigramme.

2.4 Ensembles

Unter Ensembles versteht man Methoden die eine Vorhersage erstellen, indem die Vorhersagen mehrerer Klassifikatoren kombiniert werden. Das verbessert im Allgemeinen die Genauigkeit auf Kosten von Laufzeit und, je nach Methode, Arbeitsspeicher. Dabei unterscheidet man die Methoden Stacking, Bagging und Boosting.

Stacking und Boosting eignen sich für Klassifikatoren, die zum Underfitting neigen, also einen relativ hohen Fehler auf den Test- und Trainingsdaten erzeugen beziehungsweise hohen Bias haben. Beim Stacking lernen wir mehrere verschiedene Modelle parallel und trainieren dann ein Meta-Modell auf den berechneten Vorhersagen. Eine aktuelle Veröffentlichung von Stacking auf Wörterbuchklassifikatoren ist zum Beispiel TDE [11]. Im Gegensatz dazu lernen wir beim Boosting mehrere Modelle sequenziell. Jedes neue Modell versucht dabei besonders die Fehler des vorherigen zu korrigieren. Boosting wurde bisher noch nicht auf Wörterbuchklassifikatoren angewendet.

Haben wir ein Modell das overfittet, also einen niedrigen Fehler auf den Trainingsdaten, aber einen hohen Fehler auf den Testdaten aufweist (High Variance), bietet sich Bagging (oder Stacking) an. Hierbei wird mehrmals, parallel, das selbe Modell gelernt, jedoch jeweils mit einer zufälligen Teilmenge der Trainingsdaten und eventuell sogar der Featuremenge. Diese werden dann kombiniert, indem der Mittelwert oder die Mehrheit der Vorhersagen berechnet wird. Der bekannteste Vertreter von Bagging mit Entscheidungsbäumen ist der *Random Forest* [3]. Bei diesem wird nicht nur jeder Entscheidungsbaum auf einer zufälligen Teilmenge der Trainingsdaten aufgebaut, sondern wir schränken auch die Features, die an jedem Knoten zur Auswertung benutzt werden, auf eine zufällige Teilmenge ein. Auf Zeitreihen wird zum Beispiel der Proximity Forest [10] verwendet, welchen ich weiter unten genauer erkläre.

Möchte man Ensembles auf Zeitreihen anwenden, werden meistens verschiedene Klassifikatoren verknüpft, die auch unterschiedliche Repräsentationen von Zeitreihen verwenden. Das macht man wie beim Bagging über die Berechnung eines Mittelwertes oder einer Mehrheit. Der State-of-the-art Zeitreihen-Klassifikator *HIVE-COTE* [8] kombiniert insgesamt 38 Klassifikatoren mittels Stacking, unter denen 5 verschiedenen Repräsentationen von Zeitreihen vertreten sind. Eine Besonderheit ist, dass alle Klassifikatoren, die dieselbe Repräsentation verwenden, zu einem Modul zusammengefasst werden. Jedes Modul produziert am Ende nur eine Vorhersage, die für die gewichtete Verknüpfung benutzt wird.

2.5 Ensemble mittels verschiedener Hyperparameter

Ziel der Arbeit ist es mehrere BOSS- oder WEASEL-Modelle, jeweils mit unterschiedlichen Hyperparametern, aufzubauen und diese miteinander zu verknüpfen. Das Verändern der Hyperparameter kann das Over- beziehungsweise Underfitting des Modells beeinflussen. Die Idee ist, dass das kombinierte Modell, im Vergleich zu einem Modell mit nur einem Hyperparameter, robuster gegenüber Over- und Underfitting auf den Trainingsdaten ist. In der Tabelle 1 sind die anpassbaren Hyperparameter mit ihren möglichen Wertebereichen dargestellt. Das m in der Zeile für die Fensterlänge steht dabei für die Länge der Zeitreihen.

Parameter	Wertebereich
Wortlänge	$l = \{16, 14, 12, 10, 8, 6, 4\}$
Fensterlänge	$w = \{10...m\}$
Normalisierung	$p = \{true, false\}$
Alphabetgröße	$\alpha = \{4\}$
Bigramme	True, False

Tabelle 1: Wertebereiche der Hyperparameter

2.6 Proximity Forest

In Proximity Forest [10] wird beschrieben, wie man die Idee des Random Forest auf Zeitreihen erweitert. Bei diesem entscheiden wir in einem Knoten nicht anhand von Features, in welche Richtung wir branchen, sondern haben im Fall von Wörterbuchklassifikatoren in jedem Knoten ein Histogramm mit Wörtern der Länge l über dem Alphabet Σ gegeben. Dann transformieren wir unsere derzeit betrachtete Zeitreihe in ein Histogramm mit Wörtern derselben Länge l über diesem Alphabet Σ und berechnen die Distanz zu dem gegebenen Histogramm im Knoten. Anhand dieses Distanzwertes wird bestimmt, in welche Richtung wir branchen. Dabei wählen wir zuerst für jeden Baum eine Menge von Hyperparametern, aus der wir in jedem Knoten den Besten für einen möglichst reinen Split bestimmen, so dass in einem Teilbaum nur Zeitreihen einer Klasse auftreten. Über diesem Parameter wird dann das im Knoten repräsentative Histogramm erstellt.

3 Motivation und Ziele

Ich möchte in dieser Bachelorarbeit den Einfluss von Ensembles über Hyperparameter auf WEASEL und BOSS untersuchen. Dafür habe ich folgende Ziele:

1. Vergleich verschiedener existierender Python-Implementierungen von BOSS und WEASEL mit der Java-Implementierung hinsichtlich Laufzeit und Genauigkeit.

   (a) sktime [9]: liefert eine Implementierung des BOSS- und WEASEL-Klassifikators.
   (b) pyts [5]: bietet Funktion für die WEASEL- und BOSS-Transformationen, muss noch mit entsprechenden Klassifikatoren kombiniert werden.
   (c) SFA_Python [17]: „inoffizielle“ 1-zu-1 Portierung von der Java-Implementierung.

2. Identifikation von High-Bias oder High-Variance der Implementierung und abhängig davon Wahl der Ensemble-Methoden.

3. Anwendung von einfachen Ensemble-Strategien, basierend auf scikit-learn [13], wie Bagging, Boosting und Stacking über mehrere Modelle von BOSS oder WEASEL mit jeweils unterschiedlichen Hyperparametern.

4. Evaluierung der Java- und Python-Implementierung vom Proximity Forest [10] und Anwendung der entsprechenden Implementierung auf mehrere BOSS- oder WEASEL-Modelle mit jeweils verschiedenen Hyperparametern.

4 Technische Realisierung

Da ich einige Funktionen der scikit-learn-Bibliothek benutzen werde, schreibe ich meine eigenen Implementierungen auch in Python. Zu diesen Funktionen gehören *BaggingClassifier*, *AdaBoostClassifier* und *StackingClassifier* [13]. Der Proximity Forest wurde im Rahmen einer wissenschaftlichen Arbeit in Java entwickelt. Für eine Python-Implementierung war die erste Idee den Random Forest zu benutzen und eine eigene Distanzfunktion zu übergeben. Das ist leider nicht möglich, jedoch gibt es bereits eine Python-Implementierungen des Proximity Forest in der scikit-learn-Bibliothek und eine weitere [12], die im Rahmen einer weiteren Arbeit entstanden ist, die beide auf der ursprünglichen Arbeit basieren.

5 Erste Ergebnisse

In der folgenden Tabelle 2 sind die Ergebnisse von ersten Test auf ausgewählten Datensätzen des UCR-Archivs zu sehen. Dabei habe ich die BOSS-Implementierung in Java mit den sktime und pyts Implementierungen in Python verglichen. Da pyts nur die Transformierung bereitstellt, habe ich diese mit dem 1-Nearest Neighbour-Klassifikator kombiniert. Die Ergebnisse sind dabei als Accuracy auf den Testdaten dargestellt.

Datensatz	Java BOSS	sktime BOSS	pyts BOSS + 1-NN
Coffee	1	1	0,893
Beef	0,8	0,73	0,6
CBF	0,999	0,99	0,649
ECG200	0,85	0,86	0,66
FaceFour	1	1	0,591
OliveOil	0,9	0,87	0,633
Gun_Point	1	1	0,833
DiatomSizeReduction	0,931	0,92	0,657
ECGFiveDays	1	1	0,695
TwoLeadECG	0,981	0,98	0,826
SonyAIBORobotSur	0,905	0,876	0,744
MoteStrain	0,921	0,897	0,727
ItalyPowerDemand	0,924	0,92	0,89
SonyAIBORobotSurface	0,719	0,667	0,616

Tabelle 2: Erster Vergleich verschiedener Implementierungen auf ausgewählten Datensätzen des UCR-Archivs

Man sieht, dass pyts mit 1-Nearest Neighbour bei den meisten Datensätzen sehr stark von der Java-Implementierung abweicht. sktime hingegen erreicht sehr ähnliche Ergebnisse, jedoch nicht genau die gleichen, scheint also in der Implementierung leicht abzuweichen.

Literatur

[1] Anthony Bagnall u. a. „The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances“. In: *Data Mining and Knowledge Discovery* 31.3 (2017), S. 606–660.

[2] Leo Breiman. „Bagging predictors“. In: *Machine learning* 24.2 (1996), S. 123–140.

[3] Leo Breiman. „Random forests“. In: *UC Berkeley TR567* (1999).

[4] Yanping Chen u. a. *The UCR Time Series Classification Archive*. `www.cs.ucr.edu/~eamonn/time_series_data/`. Juli 2015.

[5] Johann Faouzi und Hicham Janati. „pyts: A Python Package for Time Series Classification“. In: *Journal of Machine Learning Research* 21.46 (2020), S. 1–6. URL: `http://jmlr.org/papers/v21/19-763.html`.

[6] Yoav Freund, Robert Schapire und Naoki Abe. „A short introduction to boosting“. In: *Journal-Japanese Society For Artificial Intelligence* 14.771-780 (1999), S. 1612.

[7] Jessica Lin, Rohan Khade und Yuan Li. „Rotation-invariant similarity in time series using bag-of-patterns representation“. In: *Journal of Intelligent Information Systems* 39.2 (2012), S. 287–315.

[8] J. Lines, S. Taylor und A. Bagnall. „HIVE-COTE: The Hierarchical Vote Collective of Transformation-Based Ensembles for Time Series Classification“. In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. 2016, S. 1041–1046.

[9] Markus Löning u. a. „sktime: A Unified Interface for Machine Learning with Time Series“. In: *Workshop on Systems for ML at NeurIPS 2019*. 2019.

[10] Benjamin Lucas u. a. „Proximity Forest: an effective and scalable distance-based classifier for time series“. In: *Data Mining and Knowledge Discovery* 33.3 (Mai 2019), S. 607–635. ISSN: 1573-756X. DOI: 10.1007/s10618-019-00617-3. URL: https://doi.org/10.1007/s10618-019-00617-3.

[11] Matthew Middlehurst u. a. „The Temporal Dictionary Ensemble (TDE) Classifier for Time Series Classification“. In: *The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*. 2020.

[12] moradisten. *ProximityForests-Python*. https://github.com/moradisten/ProximityForests-python. (Besucht am 13. 09. 2020).

[13] F. Pedregosa u. a. „Scikit-learn: Machine Learning in Python“. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830.

[14] Patrick Schäfer. „The BOSS is concerned with time series classification in the presence of noise“. In: *Data Mining and Knowledge Discovery* 29.6 (Nov. 2015), S. 1505–1530. ISSN: 1573-756X. DOI: 10.1007/s10618-014-0377-7. URL: https://doi.org/10.1007/s10618-014-0377-7.

[15] Patrick Schäfer und Mikael Högqvist. „SFA: A Symbolic Fourier Approximation and Index for Similarity Search in High Dimensional Datasets“. In: *Proceedings of the 15th International Conference on Extending Database Technology*. EDBT '12. Berlin, Germany: Association for Computing Machinery, 2012, S. 516–527. ISBN: 9781450307901. DOI: 10.1145/2247596.2247656. URL: https://doi.org/10.1145/2247596.2247656.

[16] Patrick Schäfer und Ulf Leser. „Fast and Accurate Time Series Classification with WEASEL“. In: *CoRR* abs/1701.07681 (2017). arXiv: 1701.07681. URL: http://arxiv.org/abs/1701.07681.

[17] sharford5. *SFA$_P$ython*. https://github.com/sharford5/SFA_Python. (Besucht am 24. 09. 2020).

[18] David H. Wolpert. „Stacked generalization“. In: *Neural Networks* 5.2 (1992), S. 241–259. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(05)80023-1. URL: http://www.sciencedirect.com/science/article/pii/S0893608005800231.

[19] Zhi-Hua Zhou. „Ensemble Learning.“ In: *Encyclopedia of biometrics* 1 (2009), S. 270–273.