Tausendmal probiert

Tobias Weyand, Christian Buck

27. Juli 2005

1 Einleitung

Im Rahmen dieses Praktikums nahmen wir am jährlich stattfindenden Data Mining Cup teil, bei dem Studenten eine Aufgabe aus dem Themengebiet selbständig lösen müssen. Obwohl das ein Einzelwettbewerb ist, haben wir die Aufgabe in Kleingruppen bearbeitet und werden im Folgenden die diesjährige Aufgabe und unsere Herangehensweise vorstellen.

2 Die Daten
unser kleines Data warehouse

Aufgabe des diesjährigen Data-Mining-Cups war es, aus den Bestell- und Kundendaten eines Online-Versandhandels Kunden mit hoher Zahlungsausfallwahrscheinlichkeit zu bestimmen. Diesen wird nur die Zahlung per Nachnahme angeboten, was den Versand allerdings verteuert. Daraus ergab sich eine Kostenmatrix, nach der zu optimieren war.

Dazu erhielten wir 30.000 Trainings- und 20.000 Test-Datensätze mit je 43 Features. Um Overfitting zu vermeiden einigten wir uns auf eine Partitionierung der Trainingsdaten in Development- und Holdoutset mit Verhältnis 4:1. Letzteres wurde bis zur letzten Woche nicht verwendet. Um eine Vergleichbarkeit der Scores mit anderen Gruppen zu gewährleisten, legten wir uns auf 5-fach Cross-Validation fest.

3 Weka
der Vogel pickt

Das Data-Mining-Toolkit unserer Wahl war Weka. Außerdem verwendeten wir einige eigene Skripte zur

- Vorverarbeitung der Daten
- Aufteilung der Daten
- Konvertierung in verschiedene Formate
- Erstellung neuer Features
- Selektion relevanter Features
- Durchführung und Auswertung von Testläufen

Wir wählten Weka aufgrund der breiten Auswahl an Classifiern und Vorverarbeitungsfiltern, der einfachen Visualisierung, guten Skriptbarkeit, der freien Verfügbarkeit für verschiedene Architekturen und Plattformen und wegen des kleinen Vogels. Leider ergaben sich im Zusammenspiel mit verschiedenen Implementierungen der Java Virtual Machine Stabilitätsprobleme, die aber leicht zu umgehen waren.

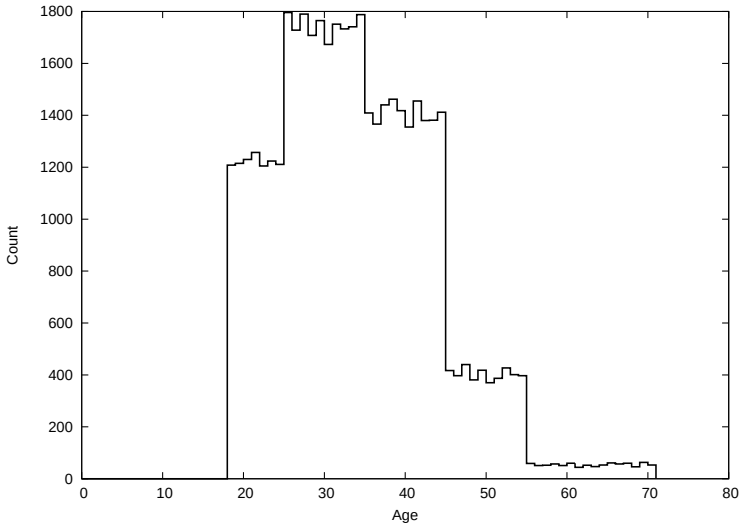

Abbildung 1: Histogramm der Altersverteilung

4 Neue Features

Widder sind bös

Die ersten Vorverarbeitungsschritte waren das Umwandeln von Datumsangaben in zeitliche Abstände. So wurden die Geburtstagsangaben in das Alter in Tagen konvertiert. Helga unterteilte die Kunden zudem nach Sternzeichen, was ein schlechtes Licht auf Widder warf. Nominale Features wie die Art der Kreditkarte wurden in mehrere binäre Features aufgeteilt.

Nach Visualisierung und eingehender Betrachtung der Daten konnten für einige Features geeignete Diskretisierungen gefunden werden. So war etwa bei der Altersverteilung eine Stufenform klar erkennbar (Abbildung 1), die eine Einteilung in Altersklassen nahe legte. Laut Aufgabenstellung handelte es sich um synthetische Daten, wir nahmen aber an, dass ausgehend von echten Datensätzen Fehlinformation hinzugefügt wurde, um die Datensätze zu anonymisieren. Dieses Rauschen versuchten wir durch Diskretisierung wieder entfernen.

Für einige besonders signifikante Ausprägungen wurden manuell zusätzliche binäre Features erstellt. Zum Beispiel wies etwa eine sehr kurze Session-Time auf unzuverlässige Kunden hin.

Anhand des Bestellwertes war es möglich den Preis der einzelnen Artikel zu errechnen, jedoch erwies sich eine Einteilung der Artikel in Produktkategorien, abhängig von der Artikelnummer, als geeignetere Alternative.

5 Feature-Selection

only the bad die young

Um herauszufinden, welche Features einer möglichst guten Klassifikation dienlich sind, implementierten wir ein automatisches Feature-Selection-Verfahren. Dieses basierte auf dem Prinzip der Backward-Elimination. Hierbei beginnt man mit dem vollständigen Feature-Set und entfernt iterativ das Feature, dessen Entfernung die größte Verbesserung im Score ergibt. Die Iteration bricht ab, sobald keine signifikante Verbesserung mehr erzielt werden kann. Den Score ermittelten wir mit Logistischer Regression bzw. Decision Trees mit 5-facher Cross-Validation. Um den Rechenaufwand zu verringern und um Overfitting erkennen zu können wählten wir hierfür ein Subset der Trainingsdaten mit 1000 bzw. 5000 Einträgen, das für jeden Durchlauf neu erstellt wurde. Durch Verwendung des Queuing-Systems im Rechnerpool konnten wir diesen Prozess parallelisieren. Ins-

gesamt führten wir 17000 Experimente durch.

Leider stand die Qualität der Ergebnisse in keinem Verhältnis zur verwendeten Rechenzeit, da sich die Verbesserungen des Scores auf dem gesamten Datensatz nicht reproduzieren ließen. Somit entschieden wir uns für manuelle Feature-Selection basierend auf theoretischen Überlegungen. Hierbei waren die Ergebnisse der automatischen Feature-Selection zumindest für ein paar Tipps zu gebrauchen, aber insgesamt sind wir nach den ernüchternden Erfahrungen mit automatischer Feature-Selection der Meinung, dass logische Überlegungen hierfür weit sinnvoller sind als automatische Verfahren.

6 Stacking
Hybris und Niedergang

Wir haben versucht, per Stacking verschiedener Classifier ein möglichst gutes Ergebnis zu erzielen. Dazu wählten wir Classifier, die auf unseren Features einen möglichst hohen Score erzeugten. In hunderten Experimenten konnten wir folgende Classifier und dazu passende Parameter ermitteln, die allesamt einen Score erzielten, der höher oder gleich dem per Logistischer Regression erreichbaren war: Multiboost, LogitBoost, SupportVectorMachine (SMO) und LogisticRegression. Wir zerlegten das Developmentset in fünf Teile, trainierten jeweils auf vieren mehrere dieser Classifier mit verschiedenen Parametern und speicherten deren Klassifikationsergebnisse auf dem verbleibenden Teil. Auf diesen neuen Features trainierten wir LogisticRegression als Meta-Classifier. Leider waren die Ergebnisse dieser Methode auf dem Holdoutset ernüchternd. Aufgrund des enormen Rechenaufwandes dieses Verfahrens waren uns leider keine weiteren Tests möglich, die aufgezeigt hätten, warum diese Methode signifikant schlechter abschnitt als die einzelnen Classifier, deren gute Ergebnisse wir auf dem Holdoutset im Nachhinein verifizieren konnten.

7 Unsere Abgabe
den Wald vor Stümpfen nicht sehen

Die per MultiBoost erzielbaren Scores waren sowohl in der Cross-Validation überzeugend als auch auf dem Holdoutset zu reproduzieren. Leider dauerte das Training sehr lange, weshalb wir keine weitere Verbesserung der Parameter erzielen konnten.

Etwa die Hälfte der Trainingsdaten bestand aus Neukunden, für die einige Features, wie etwa der Bestellwert der letzten Bestellung irrelevant waren. Durch eine Trennung der Daten in Alt- und Neukunden konnten bei einigen Classifiern Verbesserungen erzielt werden, weshalb wir diese auch für unseren finalen Datensatz verwendeten. Unsere beiden Lösungsmodelle unterschieden sich ausschließlich in den Parametern des Neukunden-Classifiers.

Zuletzt haben wir noch manuell einige Klassifikationen geändert, indem wir Kunden, die mit Kundenkarte zahlten, blindes Vertrauen schenkten.

Insgesamt bestand unser Abgabe-Classifier aus drei Schichten: Die untere Schicht bildete der Costsensitive Classifier, ein Metaclassifier, der die Ergebnisse seines Base-Classifiers anhand der Kostematrix neu gewichtet. Somit wird das Klassifikationsergebnis nicht bezüglich der Fehlerrate, sondern bezüglich der Kosten optimiert. Die mittlere Schicht war der der MultiBoost Metaclassifier, der eine Kombination aus Bagging, Wagging und AdaBoost ist. Die Oberste Schicht und damit der eigentliche Classifier war der DecisionStump. Diese Techniken sollen im Folgenden kurz erläutert werden.

7.1 Bagging

Bagging erstellt durch zufälliges Subsampling (mit Zurücklegen) aus den Trainingsdaten eine Menge von Datensätzen, die so viele Elemente wie die Trainingsdaten haben, einige Elemente aber mehrfach enthalten können. Auf diesen neuen Datensätzen wird nun jeweils ein Base Classifier

trainiert. Um nun ein Test Sample zu klassifizieren, lässt man die unterschiedlichen Classifier demokratisch abstimmen, das heißt die Klasse mit den meisten Stimmen gewinnt und wird gewählt.

7.2 Wagging

Wagging ist eine Abwandlung von Bagging. Statt aus den Trainingsdaten zu sampeln, erstellt es Training Sets, indem es jeder Trainingsinstanz ein zufälliges Gewicht zuweist. Das heißt, dass der verwendete Base-Classifier mit gewichteten Daten umgehen können muss. Der Rest (Training mehrerer Classifier und demokratisches Wählen) verläuft wie beim Bagging.

7.3 AdaBoost

AdaBoost ist eine iterative Boosting-Methode zur Bestimmung von geeigneten Gewichten für jedes Training Sample. Zunächst werden alle Training Samples mit 1 gewichtet und der Base Classifier wird auf den Daten trainiert. Dann wird das Gewicht jedes Training Samples erhöht, falls der Classifier es falsch klassifiziert, sonst wird es verringert. Dies wird über eine feste Anzahl von Schritten iteriert.

7.4 Decision Stump

Der Decision Stump ist ein Decision Tree der Tiefe 1. Das heißt, er besteht nur aus einer Regel und trifft seine Entscheidung nur aufgrund dessen, ob ein Wert des Training Samples über bzw. unter einem bestimmten Wert liegt. Allein ist dieser Classifier sehr schlecht, aber in Kombination mit MultiBoost ist er unter den besten Classifiern, die wir getestet haben.

8 Ende

Unsere Lösungsmodelle erhielten beim Data Mining Cup mit 11313 bzw 11257 Punkten die Plätze 14 und 15. Das theoretische Maximum lag laut den Veranstaltern bei etwa 12280, da sie die Daten aus statistischen Modellen generiert haben und dies der Score ist, den ein Bayes-Classifier mit den zum Generieren verwendeten Modellen erreicht.
Wir sind mit dem erreichten Ergebnis zufrieden und können uns vorstellen, nächstes Jahr erneut am Data Mining Cup teilzunehmen.

Literatur

[Witten/Frank/2000b] . Witten and E. Frank. *Data Mining – Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann, 2000.

[Duda et al., 2001] Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification (2nd ed.).* John Wiley and Sons. ISBN: 0-471-05669-3.