Lehrstuhlversuch E5b: Datenanalyse mit IceCube-Monte-Carlo-Simulationsdaten

Jan Appelt
(Jan.Appelt@udo.edu)

Thomas Jung
(thomas3.jung@udo.edu)

05.09.2017, Dortmund
Fakultät Physik, TU Dortmund

Inhaltsverzeichnis

1 Zielsetzung

In diesem Versuch soll mithilfe von maschinellen Lernern Signal vom Untergrund getrennt werden, dazu werden simulierte Daten des Ice-Cube Experimentes verwendet.

2 Theorie

2.1 Physikalische Grundlagen

Viktor Heß hat im Jahre 1912 die Höhenstrahlung entdeckt, seit dem ist bekannt das nicht nur Photonen sondern auch Protonen und andere Teilchen auf die Erde treffen. Das Energiespektrum der geladene kosmische Strahlung (Protonen, etc.) erstreckt sich bis zu einer Energie von 10^{20}eV und sie folgt annähernd einem Potenzgesetz:

$$\frac{\mathrm{d}\theta}{\mathrm{d}E} = \theta_0 E^{\gamma}$$

dabei ist $\gamma \approx -2,7$ und wird spektral Index genannt, bei Neutronen ist der spektral Index ungefähr 2.
Diese geladenen Teilchen werden durch (extra)galaktische Magnetfelder abgelenkt, somit ist die Quelle der Teilchen nicht zurückzuführen. Die Quellen die Hadronen beschleunigen sollten auch Neutrinos und hoch energetische Photonen emittieren. Da die Neutrinos ungeladen sind und nur über die schwache Wechselwirkung mit Materie interagieren, werden sie nicht durch Magnetfelder abgelenkt und können dichte Staubwolken durchdringen. Somit zeigen die Neutrinos fast direkt auf ihren Uhrsprungort. Neutrinos entstehen auch in der Atmosphäre und zwar durch Wechselwirkung von kosmischer Strahlung mit Atomen der Luft, dadurch entstehen schwere Teilchen die eine geringe Lebensdauer besitzen, die dann in Myonen und Myonneutrinos zerfallen. Bei dem Zerfall erben si den spektralen Index der kosmischen Strahlung, dieser Untergrund wird als prompte atmosphärische Komponente bezeichnet.

2.2 Das IceCube-Experiment

Das IceCube-Experiment befindet sich in einer Eistiefe von 1400m bis 2450m am geographischen Südpol. Es dient zur Identifikation von hoch energetischen Neutrinos und Myonen, und besteht aus drei Stationen: IceTop, In-Ice-Array und dem DeepCore. Zu dem DeepCore und dem In-Ice-Array gehören 86 Kabel mit insgesamt 5160 Photoelektronenvervielfachern, davon gehören sieben Kabel zum DeepCore die einen kleineren Abstand zueinander besitzen als die restlichen. Daher hat der DeepCore eine untere Energieschwelle von 10GeV und die anderen von 100GeV. Das IceTop Experiment befindet sich an der Eis Oberfläche. Hier wird die kosmische Strahlung mittels lichtdichten Eistanks detektiert. Es dient als Veto für die Neutrinodetektion da vor allem geladene kosmische Strahlung detektiert wird.
Die Teilchen werden durch ihr Tscherenkowlicht detektiert. Tscherenkowlicht entsteht, wenn geladene Teilchen sich schneller als das Licht in diesem Medium bewegt. Neutrinos interagieren mit dem Eis auf zwei grundlegenden Arten:

$$\bar{\nu_\ell}\,(\nu_\ell) + A \to \ell^{\pm} + X$$
$$\nu_\ell + A \to \nu_\ell + X$$

Die Sekundärteilchen aus diesen Wechselwirkungen werden dann detektiert.
Die Myonen haben eine lange Lebensdauer im Vergleich zu den Elektronen und Tau-Leptonen, da sie nur langsam Energie abgeben daher haben sie auch die längste Reichweite. Da die Reichweite so groß ist können sie auch viele Kilometer außerhalb des Detektors erzeugt werden und trotzdem gemessen werden.
Es entstehen auch in der Atmosphäre eine große Anzahl an Myonen und zwar durch zwei wesentlichen Wechselwirkungen:

$$\begin{aligned} p + A &\rightarrow \pi^+/K^+ + X, \\ \pi^+/K^+ &\rightarrow \mu + \nu_\mu, \end{aligned}$$

diese Myonen bilden einen Untergrund.
Da Myonen aus Luftschauern 10^6 mal öfter auftreten als Myonen aus Myonneutrinos werden nur Schauer verwendet die nicht vom Südhimmel kommen. Da Neutrinos die Erde durchqueren können aber Myonen nicht. Somit bleibt nur noch 10^3 mal mehr Untergrundergebnisse als Signalergebnisse übrig. Diese werden dann durch maschinellen Lernens voneinander getrennt.

3 Signal-Untergrund Trennung

Bei dem Verfahren des Machinellen Lernens wird ein typischer Ablauf befolgt der jetzt beschrieben wird, an diesem Ablauf haben wir uns auch gehalten.

3.1 Vorbereitung der Daten

Da die Signal und Untergrund Daten getrennt voneinander simuliert werden, werden zuerst alle Attribute entfernt die nur in einem Datensatz vorkommen. Bei manchen Attributen kann es vorkommen das NaN oder Inf vorkommt diese Werten müssen dann sinnvoll ersetzt werden oder entfernt werden. Zum Schluss bekommen die Daten noch ein Label ob es sich um ein Signal handelt oder um ein Untergrund Ereignis und dann werden sie zusammengelegt zu einem Datensatz.

3.2 Attributselektion mit mRMR

Bei der Attributselektion werden die Attribute x_k herausgesucht welche den größten Informationsgehalt haben. Bei mRMR (minimum Redundancy, Maximum Relevance) wird die Wahrscheinlichkeitsverteilungen der Variablen betrachtet und hängt somit nicht vom Lerner ab. Hierzu wird der Informationsgehalt zweier Variablen x, y benutzt:

$$I(x,y) = \int p(x,y) \log\left(\frac{p(x,y)}{p(x)\,p(y)}\right) \mathrm{d}x\mathrm{d}y,$$

dabei ist $p(x), p(y), p(x,y)$ die Wahrscheinlichkeitsdichtefunktionen der betreffenden Variablen. Die ausgewählten Attribute korrelieren stark mit den Zielattributen aber wenig untereinander. Für den gesuchten Attributsatz S_k soll folgendes gelten:

$$\max\left[D(S,c)\right],$$
$$\min\left[R(S)\right],$$

dabei ist c die Zielgröße und D und R sind wie folgt definiert:

$$D = \frac{1}{|S|}\sum_{x_i \in S} I(x_i, c), \quad R = \frac{1}{|S|^2}\sum_{x_i, x_j \in S} I(x_i, x_j).$$

Diese Bedingungen werden dann oft zu $\max \Phi(D,R)$ verbunden, dabei kann Φ wie folgt definiert werden:

$$\Phi = D - R \text{ oder als} \qquad (3.2.1)$$
$$\Phi = \frac{D}{R}$$

3.3 Stabilitätsanalyse mit hilfe des Jaccard-Index

Beim Jaccard-Index handelt es sich um ein Maß für die Ähnlichkeit zweier Mengen und wird wiefolgt berechnet:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Dies wird n-mal durchgeführt und jedes mal ein andere Teilmenge des Datensatzes verwendet. Wenn $J = 1$ ist wird eine maximale stabile Attributauswahl gegen statistischen Schwankungen beschrieben.

$$\hat{J} = \frac{2}{n\,(n-1)} \sum_{i=1}^{n} \sum_{i=1+1}^{n} J\left(F_i, F_j\right) \tag{3.3.1}$$

3.4 Lerner

3.4.1 Naive Bayes-Algorithmus

Beim Naive Bayes-Algorithmus handelt es sich um ein Algorithmus der auf dem Satz von Bayes basiert. Der Satz von Bayes sagt folgendes aus:

$$p\,(A|B) = \frac{p\,(B|A)\,p\,(A)}{p\,(B)}$$

dabei ist A die Klassenzugehörigkeit (Signal A oder Hintergrund $\bar{A}$) und B ein Attribut, und $p\,(A|B)$ ist die bedingte Wahrscheinlichkeit die die Wahrscheinlichkeit angibt das A eintrifft wenn B bereits eingetroffen ist.
Der folgende Ausdruck:

$$Q = \frac{p\,(A|B)}{p\left(\bar{A}|B\right)} = \frac{p\,(B|A)\,p\,(A)}{p\left(B|\bar{A}\right)p\left(\bar{A}\right)}, \tag{3.4.1}$$

ist größer als eins wenn es zu einer höheren Wahrscheinlichkeit ein Signalereignis ist als ein Untergrundereignis. Bei mehr als einem Attribut wird Gleichung 3.4.1 zu:

$$Q = \frac{p\,(A|B_1, \ddots, B_n)}{p\left(\bar{A}|B_1, \ddots, B_n\right)} = \frac{p\,(A)}{p\left(\bar{A}\right)} \prod_{i=1}^{n} \frac{p\,(B_i|A)}{p\left(B_i|\bar{A}\right)}$$

3.5 Random Forest

Beim Random Forest werden verschiedene binäre Entscheidungsbäume erstellt. Bei jedem Entscheidungsbaum werden unterschiedliche Attribute verwendet um die Korrelation zwischen den Bäumen zu verkleinern. Beim erstellen eines Baumes wird bei jedem Attribut geguckt wie viel Informationsgehalt er hat und danach getrennt, dies wird solange durchgeführt bis die maximale Tiefe eines Baumes erlangt ist. Beim Random Forest werden mehrere Bäume zusammen getan und eine Konfidenz c gezogen, da manche Bäume das Ereignis als Signal sehen und die anderen als Untergrund. Die Konfidenz ist eine Mittlung über alle Entscheidungsbäume.

$$c = \frac{1}{N} \sum_{i=1}^{N} P_i, \tag{3.5.1}$$

dabei kann P_i nur 1(Signal) oder 0(Untergrund) sein.

3.6 kNN

Beim kNN werden bei einem Attribut die k naheliegende Werte betrachtet und geschaut ob es sich um ein Signal oder Untergrund handelt. Da es sich bei unterschiedlich vielen Werten um ein Signal bzw. Untergrund handelt wird auch ihr eine Konfidenz c eingeführt.

$$c = \frac{1}{N} \sum_{i=1}^{N} P_i.$$

3.7 Qualitätsparameter

Um die Güte der Lerner zu klassifizieren wird die Reinheit p und die Effizienz r benutzt.

$$p = \frac{\mathrm{tp}}{\mathrm{tp} + \mathrm{fp}} \tag{3.7.1}$$

$$r = \frac{\mathrm{tp}}{\mathrm{tp} + \mathrm{fn}} \tag{3.7.2}$$

Die Anzahl der korrekt als Signal klassifizierten Ereignisse sind tp(true posetiv) und die richtigen Untergrund Ereignisse als tn(true negativ), und die fälschlich als Signal klassifizierten als fp(false posetiv) und die fälschlich als Untergrund als fn(false negative).

4 Auswertung

Für die Auswertung der Daten wurden das Programm Python [3], sowie die entsprechenden Bibliotheken Numpy [2], pandas [8], matplotlib [9], uncertainties [7] sowie scikit leanrn [6] verwendet. Für die Featureselektion wurde außerdem die Bibliothek RPy2 [4] verwendet, um das R-Packages mRMRe [1] zu verwenden.

4.1 Datenaufbereitung

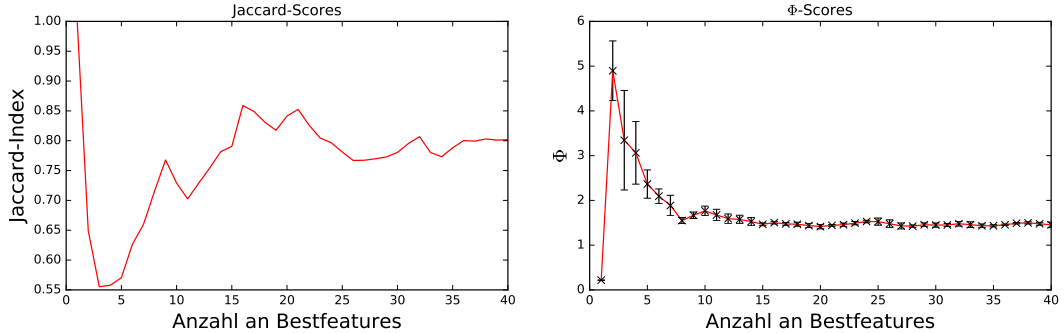
Zur Datenaufbereitung werden zunächst alle Features der beiden Datensätze *background.csv* und *signal.csv* entfernt, welche zu mehr als 75 % unvollständig sind, einen konstanten Wert beinhalten, oder nur in einem der beiden Datensätze vorkommen.

Anschließend werden die beiden Datensätze zusammengefügt, nachdem jeweils das Label „IsSignal“ hinzugefügt wurde, welches für Daten aus den Signaldaten den Wert 2 bekommt, und für Daten aus den Untergrunddaten den Wert 1.

Zum Schluss werden nicht numerische Werte in den Datensätzen durch numerische ersetzt, und fehlende, bzw. falsche Werte, für Daten von Features, welche nicht numerisch oder vom Datentyp *Integer* sind durch den Wert 0 ersetzt, die restlichen durch den Mittelwert der restlichen Daten des entsprechenden Features.

4.2 Featureselektion

Für die Featureselektion wird der MRMR-Algorithmus verwendet, wobei für eine beste Anzahl von bis zu 40 zu selektierenden Features geprüft wurde, welche Anzahl an zu selektierenden Features den besten Wert für den Jaccard-Index (3.3.1), abgebildet in Abbildung 1a, und den Φ-Score (3.2.1), abgebildet in Abbildung 1b erreicht. Um die entsprechenden Scores zu berechnen werden 20 Samples aus je 30 % zufällig gezogener Werten der original Daten erstellt. Aus den verschieden Samplen werden dann pro Anzahl an Bestfeatrues sowohl der Jaccard-Index, wie in Abbildung 1a zu sehen, als auch aus dem Mittelwert und der Abweichung der Φ-Werte für jeden Sampel der Φ-Score, wie in Abbildung 1b zu sehen, berechnet.



1a Jaccard-Index.

1b Mittelwert und Abweichung des Φ-Scores.

Abbildung 1: Aus 20 verschiedenen, zufällig gezogenen Samples berechneter Jaccard-Index (links) und berechneter Mittelwert sowie Abweichung der Φ-Scores (rechts) für verschiedene Anzahlen an Bestfeatrues.

Dies war für eine Anzahl von 16 zu selektierenden Featuren, für folgende Best-Features der Fall:
„LineFit_TTParams.lf_vel_z“, „SplineMPEDirectHitsA.n_dir_strings“, „SplineMPEFitParams.rlogl“, „SplineMPECharacteristics.track_hits_separation_length“, „SplineMPECharacteristics.avg_dom_dist_q_tot_dom“, „NewAtt.SplineVerRadius“, „SPEFit2Bayesian.z“, „HitStatisticsValues.z_travel“, „MuEXAngular4_Sigma.value“, „SPEFit2_TT.zenith“, „SplineMPEDirectHitsA.n_dir_doms“, „MPEFitParaboloidFitParams.err1“, „NewAtt.DeltaZd“, „SPEFit2Bayesian.zenith“, „SplineMPEDirectHitsA.dir_track_length“ und „NewAtt.AbsSmooth“.

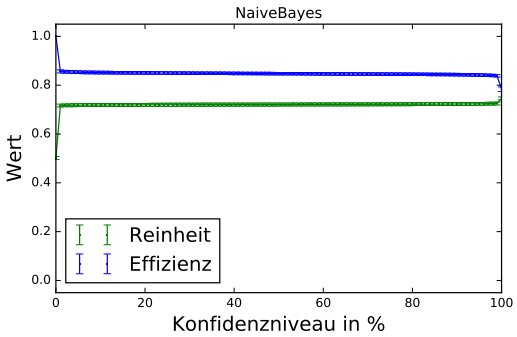
Nun werden die aufbereiteten Daten auf die selektierten Features, sowie das Label-Feature reduziert.

4.3 Lernalgorithmen

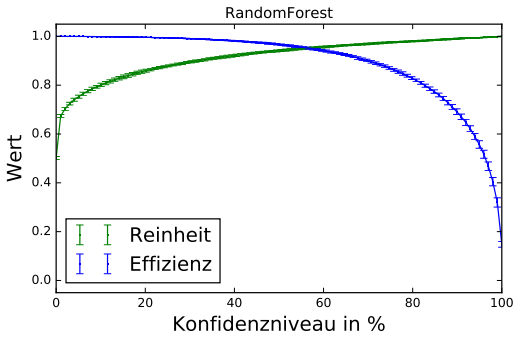
Zunächst werden nun aus den Daten zufällig 10 Sample gezogen, wobei jedes Sample aus jeweils 50 % der original Daten besteht, wobei $^3/_5$ aus Trainingsdaten und $^2/_5$ aus Testdaten bestehen. Für jedes Sample wurde hierbei ein eigener Key für den Zufallsgenerator benutzt, welcher für das erste dem Wert 2407 entspricht, und für jedes weitere Sample um je 100 erhöht wird.

Anschließend wird für jedes dieser Sample je ein Lerne basierend auf den *NaiveBayes*, dem *RandomForest*, mit je 300 Bäumen und einer maximalen Tiefe von 6 Knoten, und dem *k-NearestNeighbors*, für jeweils 6 Nachbarn, trainiert und anschließend für die Test-Daten eine wahrscheinlichkeits- vorhersage getroffen, mit welcher Wahrscheinlichkeit das jeweilige Ereignis einem Signal zuzuordnen wäre. Dann wird jeweils für 100 gleichmäßig verteilte Konfidenzniveaus von $0\,\% - 100\,\%$ nach der Definition (3.7.1) und (3.7.2) für jedes Sample die Reinheit und die Effizienz der Vorsage berechnet.

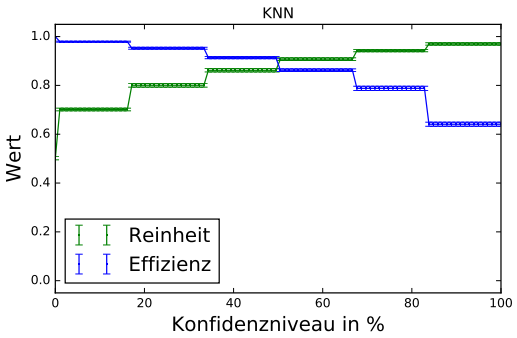
Dann wird der Mittelwert und die Standardabweichung für jeden Lerner und für jedes Konfidenzniveau über alle Reinheit- und Effizienz-Werte der Sample bestimmt und für den *NaiveBayes* in die Abbildung 2a, für den *RandomForest* in die Abbildung 2b und für den *k-NearestNeighbors* in die Abbildung 2c eingetragen.



2a *NaiveBayes* Lerner.



2b *RandomForest* Lerner.



2c *k-NearestNeighbors* Lerner.

Abbildung 2: Mittelwert und Standartabweichung für die Reinheit und die Effizienz in Abhängigkeit des gewählten Konfidenzniveaus für den *NaiveBayes* Lerner (links), den *RandomForest* Lerner (rechts) und en *k-NearestNeighbors* Lerner (unten).

5 Diskussion

5.1 NaiveBayes

Wie anhand der Abbildung 2a zu erkennen ist, liegen die Werte für Reinheit und Effizienz, bis auf kleine Konfizenzniveaus bis ca. 3 % und große Konfidenzniveaus ab 97 %, in etwa auf einem konstanten Wert von etwa $0,72$ für die Reinheit und etwa $0,85$ für die Effizienz. Für kleine und große Konfidenzniveaus steigt der Wert der Reinheit mit steigendem Konfidenzniveau an, für die Effizienz ab.

Wie zuerkennen ist, liegt kein Schnittpunkt zwischen den Werten der Reinheit und der Effizienz vor, weshalb der beste Schnitt im Konfidenzniveau bei 100 % liegt.

5.2 RandomForest

Wie in Abbildung 2b zu erkennen ist, nimmt der Wert der der Reinheit in Abhängigkeit des Konfidenzniveaus in etwa logarithmisch zu und für die Effizienz in etwa logarithmisch ab.

Der beste Schnitt im Konfidenzniveau liegt etwa bei 57 %.

5.3 k-NearestNeighbors

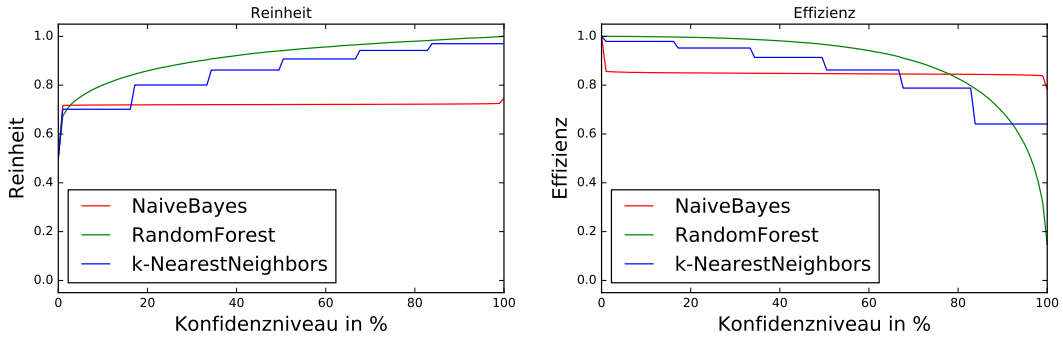
Wie in der Abbildung 2c zu erkennen ist, weisen hier sowohl die Reinheit als auch die Effizienz eine Abhängigkeit vom Konfidenzniveau auf. Die dabei jeweils zu zuerkennen Abstufungen lassen sich dadurch durch die aus jeweils 6 Nachbarn bestehende Clusterung bei der Klassifizierung zurückführen. Während die Reinheit mit steigendem Konfidenzniveau zunimmt, nimmt die Effizienz mit steigendem Konfidenzniveau ab.

Der beste Schnitt im Konfidenzniveau liegt hier in etwa bei 50 %.

5.4 Vergleich

Wie anhand der Abbildungen 3a und 3b zu erkennen ist, zeigt der *RandomForest* sowohl bei der Reinheit als auch bei der Effizienz die höchsten Werte, über das größte Konfidenzniveauintervall.

Beim Vergleich der Effizienzen in Abhängigkeit vom gewählten Konfidenzniveau ist zu erkennen, dass der *k-NearestNeighbors* einen stärkeren Abstieg der Effizienz zu verzeichnen hat als der *NaiveBayes*, aber für den größeren Konfidenzniveausintervall bis etwas 65 % eine höhere Effizienz aufweist. Für die Werte der Reinheit ist zu erkennen, dass der *k-NearestNeighbors* einen stärkeren Anstieg der Effizienz zu verzeichnen hat als der *NaiveBayes*, und für den größeren Konfidenzniveausintervall ab etwas 17 % eine höhere Reinheit aufweist.



3a Reinheit.

3b Effizienz.

Abbildung 3: Der Wert der Reinheit (links) un der Effizienz (rechts) aufgetragen gegen das Konfidenzniveau für den *NaiveBayes*, den *RandomForest* und den *k-NearestNeighbors*.

5.5 Resultat

Wie anhand der in den vorherigen Abschnitten präsentierten Ergebnissen zu erkennen seien sollte, erweist sich der *RandomForest* als der geeignetste Lerner für die Klassifizierung von atmosphärischen Neutinoereignissen, im Vergleich mit dem *NaiveBayes* und dem *k-NearestNeighbors*. Anschließend folgt der *k-NearestNeighbors* und als am ungeeignesten erweist sich der *NaiveBayes*.

Literatur

[1] Nicolas De Jay et al. *mRMRe*. URL: https://cran.r-project.org/web/packages/mRMRe/mRMRe.pdf. Version 2.0.5.

[2] Numpy developers. *Numpy*. URL: http://www.numpy.org/. Version 1.9.1.

[3] Python Software Foundation. *Python:* URL: https://www.python.org/. Version 3.4.1.

[4] Laurent Gautier. *RPy2*. URL: http://rpy.sourceforge.net/rpy2/doc-2.1/html/index.html. Version 2.1.9.

[5] Hans Hagen, Hartmut Henkel, Taco Hoekwater u. a. *LuaTeX*. URL: http://www.luatex.org/. Version beta-0.79.1.

[6] INRIA. *scikit learn*. URL: http://scikit-learn.org/stable/index.html. Version 0.19.0.

[7] Eric O. Lebigot. *Uncertainties: a Python package for calculations with uncertainties*. URL: http://pythonhosted.org/uncertainties/. Version 2.4.5.

[8] Wes McKinney. *pandas*. URL: http://pandas.pydata.org/index.html. Version 0.20.3.

[9] The Matplotlib development team. *matplotlib*. URL: http://matplotlib.org/contents.html#. Version 2.0.2.