DOKUMENTATION CA2&CA3

[Document subtitle]

Inhalt

Dies ist eine Dokumentation der im Fach „Partikuläre Datenanalyse und maschinelles Lernen“ erstellten Systeme. Es sollten verschiedene Verfahren zum Klassifizieren der Iris Daten und ein Spamfilter mithilfe von naive Bayes implementiert werden.

Tim Schäfer
1610337@stud.hs-mannheim.de

hochschule mannheim

Contents

CA-2 Klassifizierung der Iris Daten

Ziel

Im Rahmen von CA-2 sollte ein System erstellt werden, welches Daten aus dem Iris-Dataset klassifizieren kann. Dies soll mithilfe der Verfahren kNN, random forest und svm geschehen. Darüber hinaus soll ein weiteres Modell entwickelt werden, welches diese drei Verfahren als Ensemble-Learning kombiniert. Dies soll nach einfachem Abstimmungsrecht (die Mehrheit gewinnt) funktionieren.
Neben dem System zur Klassifizierung sollte auch ein System erstellt werden (oder ein Subsystem), welches das Iris-Dataset vorbereitet. Unter der Vorbereitung sollten mindestens die folgenden Funktionen vorhanden sein:

- LinePreFilter
  Es sollen bestimme Zeilen aus dem Datensatz entfernt werden
- CalcFeature
  Es sollen neue Features aus anderes errechnet werden
- Features
  Es sollen nur bestimme Features für die Klassifizierungsverfahren verwendet werden

- Ausreißer-Analyse
  bitte noch ausfüllen

Dieses Sub-System wurde in einer ähnlichen Form bereits in einem Workshop vorgestellt und konnte mit einigen kleinen Anpassungen weiterverwendet werden.

Der Code kann auf dem im Unterricht verwendeten NAS-Server oder auch unter dem folgenden Link eingesehen werden. Der GitHub Link enthält immer die neuesten Änderungen des Codes. Der NAS-Server alle Änderungen bis Ende 2018. Das Hauptsystem ist *main.py* und das Verwendete Subsystem ins *DataAnalysis.py*.

https://github.com/1610337/PML/tree/master/SciKitLearn/MachineLearningCombined

Eingabedaten

Grundlage des Systems sind Daten des Iris-Dataset. Das Iris-Dataset besteht aus Längenangaben zu der sogenannten Iris Blume. Nähere Informationen sind zu genüge online zu finden und werden hier nicht weiter erläutert.
Die zur Verfügung gestellten .csv Dateien haben alle die Spaltennamen (sepal length, sepal width, petal length, petal width, iris) in der ersten Zeile und außerdem sind einzelne Spalten jeweils mit einem Semikolon getrennt.

Datenverarbeitung

Der Punkte der Datenverarbeitung umfasst das Einlesen der oben erläuterten Eingabedaten, das Filtern, das Berechnen neuer Features und auch dem aufteilen der Eingabedaten in einen Test und Trainingsdatensatz. In dem Hauptsystem wird dies in der Methode *main* durch den Aufruf der Methode *get_filtered_data()* getan. Diese Methode gibt direkt die Trainings und Test Daten zurück.

Methodenerklärungen:

Main.get_filtered_data()

- Aufruf von DataAnalysis.get_filtered_data() und erhalten der gefilterten Daten in einem pandas Dataframe und dem *returnStr,* welcher später als Logfile gespeichert wird.
- Erstellen von eventuell neuen Feature-Spalten im Dataframe. Diese Funktion ist i.d.R. auskommentiert.
- Splitten der Daten in Test und Trainingsdaten mithilfe der Methode *train_test_split()* aus der Library *sklearn.model_selection*. Dieser Vorgang ist im Machine-Learning weitgehend standardisiert und kann daher durch eine Library durchgeführt werden. [1]

DataAnalysis.get_filtered_data()

Die Methode dieses Subsystems verwendet Parameter welche in der Datei *parameters.py* definiert sind. Dieses Subsystem bzw. diese Methode des Subsystems stammt aus einem Code, welcher in einem Workshop der Vorlesung verwendet wurde. Da dieser hinreichend von dem Autor kommentiert wurde, werden hier nur noch die im Code vorgenommenen Änderungen dokumentiert.

- Beim Lesen des Inputfiles müssen alle Kommas durch Punkte ersetzt werden, da die *.csv* in einem anderen Format geliefert wurde. Des Weiteren musste dort die erste Zeile übersprungen werden, da dort die Headerinformationen liegen.
- Einführen der Variable *returnStr*. Viele Formatstrings, welche in ein Separates Logfile gespeichert wurden, werden nun in diesen String geschrieben.

- Cast des Dictionaries *rawdata* in ein pandas DataFrame und Löschen der Spalten, die nicht in den *featurecols* genannt wurden.

ML-Verfahren

Nachdem in der Methode *main()* die Daten eingelesen, bearbeitet und in Test und Trainingsdatensätze geteilt wurden, wird nun für jedes geforderte ML-Verfahren eine Methode aufgerufen, welche dieses Verfahren durchführt und den gegebenen Accuracy-Value des Modells zurück gibt. Die Rückgabewerte werden anschließend als Konsolenoutput und in einem Logfile ausgegeben.

Methodenerklärungen:

Main.knn()

KNN wurde standardisiert mit dem Framework sklearn implementiert. Die Dokumentation der von sklearn verwendeten Methoden ist unter [2] zu finden. Zunächst wird ein KNN-model instanziiert. Testweise sollen 5 Nachbarn eines untersuchten Elements betrachtet werden *(n_neighbors=5).* Dieses Model wird dann trainiert und getestet. Für die Tests wird ein gewisser score Wert zurückgegeben.

Main.random_forest()

Diese Methode verwendet zwei Möglichkeiten, die Irisdaten zu klassifizieren. Es wird zunächst ein einfacher DecisionTree erstellt, trainiert, getestet und bewertet und danach passiert das Gleiche mit einem RandomForestClassifier mit einem testweisen gewählten Parameter von *n_estimators=200*. Relevant ist in diesem System jedoch nur der RandomForestClassifier.

Es werden von sklearn angebotene Methoden für dem classification_report und die Modelle der Verfahren standardisiert benutzt. [3][4]. Die *weighted average precision* bietet die beste Möglichkeit die Präzision des Modells zu messen, daher wird nur dieser Wert zurückgegeben.

Main.svm()

Es werden von sklearn angebotene Methoden und Modelle des Verfahrens für SVM standardisiert benutzt. [5]. Die *weighted average precision* bietet die beste Möglichkeit die Präzision des Modells zu messen, daher wird nur dieser Wert zurückgegeben.

Main.ensemble_learning()

Diese Methode lässt die Verfahren KNN, DecisionTreeClassifier und SVC zusammen durch einen *VotingClassifier* [6] die Daten des Iris-Dataset klassifizieren. Zunächst werden die drei Modelle mit Hilfe von *sklearn* erstellt. Danach werden die drei Modelle als Parameter für den *VotingClassifier* übergeben. Ein weiterer Parameter ist *„voting='hard'"* da die Mehrheit an stimmen über die Klassifikation bestimmen kann.

Die *weighted average precision* bietet die beste Möglichkeit die Präzision des Modells zu messen, daher wird nur dieser Wert zurückgegeben.

Offene Issues

Das System ist bisher allen Anforderungen gerecht geworden. Im Folgenden werden bekannte Probleme und mögliche Verbesserungen bzw. Erweiterungen aufgelistet, welche in der Zukunft denkbar wären.

- Berechnung von neuen Features sollte in *DataAnalysis.py* stattfinden

- Jegliche Filter in *DataAnalysis.py* könnten ausschließlich in pandas stattfinden, jedoch muss auch die Berechnung der KMP Tabelle dafür umgeschrieben werden. Außerdem gibt es dadurch auch das Problem der doppelten Parameterpflege, wie z.B. bei der Auswahl von Zeilen welche nicht verwendet werden sollen.
- Das Ausgabefile sollte entweder nur als String oder nur als Logfile bearbeitet werden.

CA-3 Naive Bayes Spam-Filter

Ziel

In dieser Teilprüfung sollte ein Spamfilter programmiert werden. Dieser sollte mit Hilfe von Naive Bayes die Wahrscheinlichkeit, mit welcher eine E-Mail eine Spam-Mail ist errechnen. Darüber hinaus sollte für die endgültige Klassifikation geprüft werden, ob der Sender auf einer Blacklist oder Whitelist steht und sofern dies nicht der Fall ist eine Klassifikation nach Bayes stattfinden. Bei der Bayes-Klassifikation sollten die oberen und unteren Grenzen der Wahrscheinlichkeiten variabel gestaltet werden.
Eine Email kann am Ende entweder *spam, nospam* oder *unbestimmt* sein.

Die Logik des Spamfilters liegt in *spamfilter.py.* Die Codebasis Code kann auf dem im Unterricht verwendeten NAS-Server oder auch unter dem folgenden Link eingesehen werden. Der GitHub Link enthält immer die neuesten Änderungen des Codes. Der NAS-Server alle Änderungen bis Ende 2018.

https://github.com/1610337/PML/tree/master/SciKitLearn/NaiveBayes/SpamFilter

Eingabedaten

Im Folgenden werden die verwendeten Eingabedaten für das System kurz erklärt.

- Dir.spam
  In diesem Ordner sind einige Spamemails als Textdateien gespeichert. Jede dieser Textdateien enthält den Absender, Betreff, Empfänger und weitere Headerdateien am Anfang. Der Header Teil wird durch die erste Leerzeile beendet.
- Dir.nospam
  In diesem Ordner sind eine Reihe von normalen Emails vorhanden. Die Struktur dieser entspricht der Struktur der Spammails.
- Spamfilter_params.py
  Diese Datei macht allen variablen Parameter für das Hauptsystem leicht anpassbar. Die wichtigsten Features sind die Prioritätenreihenfolge (Ob zuerst auf Blacklist, Whitelist oder nach Bayes geprüft werden soll), die Wahrscheinlichkeitsgrenzen für Bayes und die Liste für Wörter, welche ignoriert werden sollen.
- Whitelist.txt
  Eine Liste von Emailadressen welche zu vertrauenswürdigen Personen gehört.
- Blacklist.txt
  Eine Liste von Emailadressen welche nicht zu vertrauenswürdigen Personen gehört.

Datenvorbereitung

Die Datenvorbereitung umfasst das einlesen aller Emails und Parameter und dem erstellen des Bayes Modelles. Die Hauptpunkte des Spamfilters sind in der Methode *main()* gegliedert. Die dafür verwendeten und aufgerufenen Methoden werden im Folgenden erläutert.

Methodenerklärung

Read_emails():

Diese Methode ließt alle Emails in einem Ordner. Als Parameter muss der Pfad zu dem jeweiligen Ordner und ein boolean Wert, welcher angibt ob die Mail Spam ist oder nicht, mit angegeben werden.
Jede Datei des Ordners wird erkannt und als Textdatei geöffnet. Es wird durch jede Zeile iteriert und jede Zeile wird gespeichert. Sobald die erste Leerzeile erreicht wurde werden bis zu diesem Punkt gespeicherte Zeilen als Parameter für die Methode *get_header_dic* verwendet. *Get_Header_Dic* erkennt mithilfe von String-Operationen und regulären Ausdrücken die Header Inhalte und gibt diese in einem Dictionary zurück.

Die Methode gibt am Ende ein verschachteltes Dictionary zurück. Jede E-Mail ist ein Key mit einem darunter liegenden Dictionary mit den Keys: ['Von', 'Datum', 'An', 'Betreff', 'text', 'title', 'class']

Train_model()

- Diese Methode ist verantwortlich für die Erstellung der Datei worcount.csv.
- Parameter sind die Trainingsdaten (Also spam_mails und ham_mails) und nochmal die beiden Mail Gruppen in getrennter Form
- Die Datei wird aus einem Pandas Dataframe generiert, welcher auch als Rückgabewert Verwendung findet. Die Rückgabeobjekte *classifier* und *vectorizer* werden nicht von dem Hauptsystem benötigt (dienten lediglich zum Vergleich mit dem von sklearn eingebauten Bayes Algorithmus, dieser ist noch im Code vorhanden und wird kurz unter dem Punkt „Analyseprozess“ erläutert). Dennoch werden die beiden Objekte zum Erstellen der Wörtertabelle verwendet.
- Die durch den classifier und vectorizer hinzugefügten Spalten sind „Word“ (Das Schlüsselwort), „HamCount“ (Wie oft dieses Wort in normalen Mails vorkommt) und „SpamCount“ (Wie oft dieses Wort in spam Mails vorkommt)
- Es werden außerdem die Spalten „Ratio Ham 0 – Spam1“(Der prozentuale Wert mit dem das Wort in einer spam Mail vorkommt), „WordInHams“ (In wie vielen normalen Mails das Wort vorkommt) und „WordInSpams“(In wie vielen spam Mails das Wort vorkommt)

Analyseprozess

Der Analyseprozess findet in der Methode *main()* statt. Es wird über jede Mail, welche im Inputordner liegt, iteriert und je nach Priorität eine Klassifikationsmethode (Blacklist/Whitelist/Bayes) probiert, bis eine der Methoden eine definitive Antwort liefert. Wird keine Antwort geliefert entscheidend Bayes und die Mail wird als „undetermined“ eingestuft. Da die Inputmails auch durch die Methode *Read_Emails()* aufbereitet wurden, kann die Suche in der Blacklist und Whitelist durch eine einfache Substringsuche durchgeführt werden. Anschließend werden die wichtigsten Methoden für den Prozess des Analyseprozesses erklärt.

Methodenerklärung

Final_operations()

- Löscht lediglich alle Inhalte der Outputordner, sodass keine Daten doppelt vorhanden sind.

Bayes_filter2()

- Die zweite Implementierung des Bayes Verfahrens mit dem oben erwähnten Dataframe

- Die zu untersuchende Mail und der Dataframe dienen als Parameter
- Der Dataframe durch den Parameter und der von get_word_df() werden gemerged, sodass man für jedes Wort der Mail direkt die Spamwahrscheinlichkeit auslesen kann, ohne dass andere Wörter im Dataframe sind
- Der finale Value ist der Durchschnittswert aller Spamwahrscheinlichkeiten der Wörter in der Mail

Get_word_df()

- Liefert durch die gleiche Vorgehensweise wie in der Methode Train_Model() ein Dataframe mit der Anzahl eines Wortes in dem Text (also mail), welcher durch den Parameter definiert ist.

Bayes_spam_filter()

- Eine funktionsfähige Implementierung von Bayes mit Hilfe von sklearn.
- Da die Vohersagewahrscheinlichkeiten sehr extreme Werte haben, wird diese Methode nicht mehr genutzt und daher auch nicht weiter erläutert.

Verweise

1. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
2. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
3. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
4. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
5. https://scikit-learn.org/stable/modules/svm.html
6. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html