ABSCHLUSSPROJEKT

TEMPERATURENTWICKLUNGEN ANALYSIEREN MIT PYTHON

JULIA SCHÄFER
KITZ24 T-INFT-3/4
25. JUNI 2025

Inhaltsverzeichnis

1 Einleitung

1.1 Aufgabenstellung

Ziel des Projektes war es, den Umgang mit KI-Methoden (Polynomielle Regression, Entscheidungsbaum) in Python zu vertiefen. Dafür galt es Temperaturverläufe verschiedenere Städte weltweit an historischen Daten zu analysieren, Temperaturprognosen für die Zukunft abzugeben und Temperaturtrends für noch neue Werte einer bisher umklassifizierten Stadt zu klassifizieren.

1.2 Verwendete Bibliotheken

- Bibliotheken für Kernfunktionen:
  - numpy: Arbeiten mit numerischen Berechnungen und Array-Formaten
  - pandas: Datenmanipulation und -analyse
  - scikit-learn: Machine Learning Algorithmen und Datenvorbereitung dafür
  - matplotlib: Datenvisualisierung
  - os: Systemfunktionen z.B. für Dateien/Ordner

- Bibliotheken für zusätzliche Funktionen
  - PyQt5: Für ein GUI
  - Sys: Systemfunktionen, die mit PyQt5 nützlich sind

1.3 Ziel des Programms

Das Programm soll am Ende der Aufgabenstellung entsprechend die Daten aus einer gegebenen CSV-Datei auslesen, diese analysieren, Prognosen erstellen und Klassifizierungen zurückgeben, wofür die entsprechenden Modelle trainiert werden und die Benutzerinteraktion über ein GUI auf Basis von PyQt5 gehändelt wird.

2 Klassenübersicht

2.1 Allgemein

Insgesamt besteht das Projekt aus 5 verschiedenen Hauptmodulen.

1) main_gui_qt.py
2) stadt_verwaltung.py
3) stadt.py
4) regression_model.py
5) classification_model.py

Dabei stellen die Module 2 – 5 jeweils eine Klasse mit Attributen und Methoden, die für das Programm benötigt werden, während das Modul „main_gui_qt.py“ das Hauptprogramm darstellt, welches auch die Nutzerführung im GUI beinhaltet.

2.2 main_gui_qt.py

Hier werden die Seiten des GUI erstellt und auf die entsprechenden Module zugegriffen, um die Aufgaben ausführen zu können.
Für das GUI werden drei Klassen angelegt:

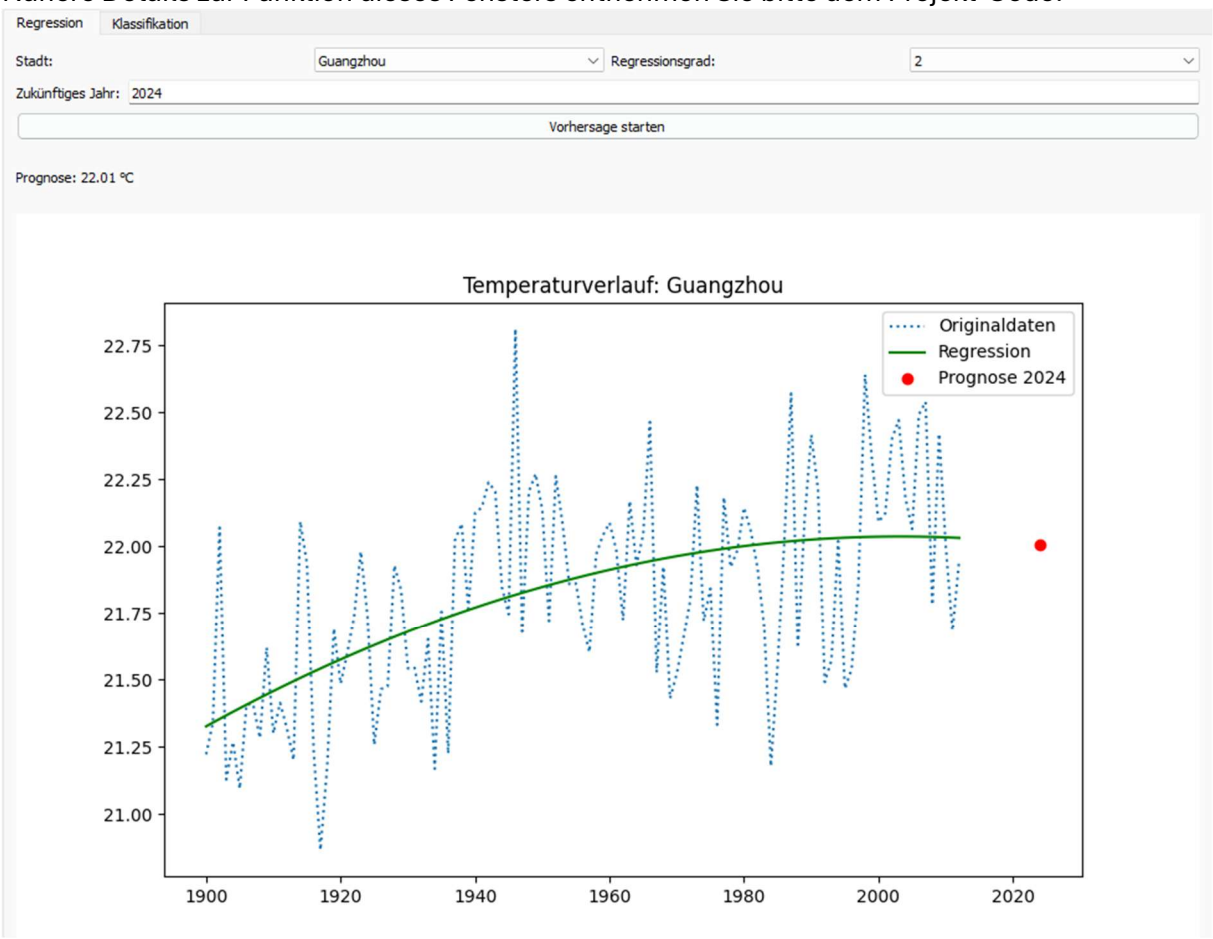
- **MainWindow**
  Das Hauptfenster, in dem sich dann alle weiteren Elemente aufhalten und welches auch Informationen hinterlegen kann, die einen Wechsel zwischen Unterfenstern überdauern.
  Eine dieser Informationen wäre der StadtManager, welcher initialisiert wird und den durch „os“ modifizierten Dateipfad übergeben bekommt. Die Modifikation des Pfads ermöglicht es zuverlässig die CSV-Datei zu übergeben, wenn sie sich im selben Ordner wie das Skript befindet.

```
class MainWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Temperaturanalyse")
        base_path = os.path.dirname(os.path.abspath(__file__))
        csv_path = os.path.join(base_path, "GlobalLandTemperaturesByMajorCity_yearly_1900-2012_sigma_classified.csv")
        self.manager = StadtManager(csv_path)
}
```

  Außerdem werden die Tabs/Unterfenster innerhalb der MainWindow-Klasse initialisiert und bekommen dann diesen StadtManager mitgegeben.

```
self.regression_tab = RegressionTab(self.manager)
self.classification_tab = ClassificationTab(self.manager)
```

- **RegressionTab**
  In diesem Tab wird die polynomielle Regression ausgeführt. Die gewünschte Stadt und der Grad der Regression können hier vom Nutzer über Dropdown-Menüs ausgewählt werden und per Tastatureingabe dann das Jahr angegeben werden, bevor die Auswahl dann zur Ausführung mit einem Button bestätigt werden muss. Der Grad der Regression wir jedoch automatisch beim Initialisieren des Tabs auf 3 gesetzt. Grad 3 wurde als Empfehlung präferiert, da Grad 1 sich in einigen Fällen als unpräzise herausgestellt hat, aber oft auch brauchbar bleibt, und Grad 2 sich zum Beispiel bei Guangzhou als eher unwahrscheinlich gezeigt hat, da hier die Prognose nach unten geht.
  Als Ausgabe werden der Wert der Prognose und ein entsprechendes Diagramm angezeigt.
  Nähere Details zur Funktion dieses Fensters entnehmen Sie bitte dem Projekt-Code.



- **ClassificationTab**
  Dieser Tab bietet die Möglichkeit die relevanten Werte für eine noch nicht klassifizierte Stadt per Tastatureingabe an das Programm zu übergeben, welches dann mit Hilfe eines Entscheidungsbaums eine entsprechende Einordnung trifft. Ausgegeben werden die Einordnung, die Genauigkeit des Modells (anhand von Testwerten bestimmt) und eine grafische Darstellung des Entscheidungsbaums.

2.3 stadt_verwaltung.py

Diese Datei beinhaltet die StadtManager-Klasse, welche für das Auslesen der Daten aus der CSV-Datei verantwortlich ist und daraus dann eine Liste aller Städte erzeugt und bereitstellt. Dabei wird darauf geachtet, dass Werte nicht redundant abgelegt werden sondern jede Stadt nur die Jahre und zugehörigen Temperaturen als DataFrame erhält, während die anderen Informationen als einzelne Werte entsprechend für eine neue Instanz der Stadt-Klasse verwendet werden.

```
        stadt_liste = []
        for city in self.df["City"].unique():
            stadt_df = self.df[self.df["City"] == city].copy()
            temp_df = stadt_df[["Year", "Temperature"]].sort_values("Year")
            first_row = stadt_df.iloc[0]

            stadt = Stadt(
                _name=first_row["City"],
                _country=first_row["Country"],
                _latitude=first_row["Latitude"],
                _longitude=first_row["Longitude"],
                _temp_df=temp_df,
                _temp_change_25y=first_row["TempChange25y"],
                _temp_std=first_row["TempStd"],
                _temp_trend=first_row["TempTrend"]
            )
```

Dass jede Stadt einmal bearbeitet wird, wird durch „unique()“ bedingt. Und die Variable „first_row“ stellt alle Daten, die nicht als DataFrame benötigt werden bereit, dass nur noch der column-Name beim Initialisieren angefügt werden muss.

2.4 stadt.py

Hier wird die Struktur für die Stadtobjekte bereitgestellt und auch Methoden, die der Übersichtlichkeit in übergeordneten Programmabschnitten dienen, da sie dort das Abrufen von Informationen verständlicher lesbar machen.

```
class Stadt:
    def __init__(self, _name, _country, _latitude, _longitude, _temp_df, _temp_change_25y, _temp_std, _temp_trend):
        self.name = _name
        self.country = _country
        self.latitude = _latitude
        self.longitude = _longitude
        self.temp_df = _temp_df
        self.temp_change_25y = _temp_change_25y
        self.temp_std = _temp_std
        self.temp_trend = _temp_trend

    def get_years(self):
        return self.temp_df["Year"].values

    def get_temperatures(self):
        return self.temp_df["Temperature"].valuesd=first_row["TempTrend"]
```

2.5 regression_model.py

In diesem Skript wird die Klasse „CityTemperatureAnalyzer“ bereitgestellt, welche numpy und scikit-learn als externe Bibliotheken benutzt. Die Klasse erzeugt beim Initialisieren als Attribute einen Platzhalter für das Modell und eine Instanz von „PolynomiaFeatures“ mit dem entsprechenden Grad, der übergeben wird aus dem Hauptprogramm, damit die Werte für das Modell passend vorbereitet werden können.
In dieser Klasse gibt es drei Methoden:

- **train**
  Training des Modells mit entsprechenden Übergabewerten
- **predict**
  Vorhersage des Wertes für das eingegebene Jahr
- **get_regression_curve**
  Stellt die Werte für die Regressionskurve bereit, dass diese dann im Hauptprogramm für den Plot verwendet werden kann

2.6 classification_model.py

„TemperatureTrendClassifier“ ist entsprechend die Klasse, welche für den Entscheidungsbaum verantwortlich ist. Die beinhaltet als eins ihrer Attribute eine Instanz des „DecisionTreeClassifiers“, welcher aus Gründen der Reproduzierbarkeit mit einem random_state versehe wurde.
Auch in diesem Modul werden wieder drei Methoden bereitgestellt:

- **train**
  Hier werden die übergebenen Daten erst einmal (erneut mit einem random_state für Reproduzierbarkeit) in 80% Trainigs- und 20% Testdaten aufgeteilt, bevor das Modell mit „fit“ trainiert und die Genauigkeit mit Hilfe der Trainingsdaten bestimmt wird.
- **predict**
  Entsprechend dem Namen wird hier die Vorhersage der Klassifikation erzeugt und als Ausgabe abgegeben.
- **plot**
  Die Darstellung des Entscheidungsbaums für Ausgabe im GUI wird hier bereitgestellt und in plot_tree parametriert.

3 Fazit

Dieses Projekt stellte eine sehr gute Möglichkeit dar die gelernten Inhalte noch einmal zu vertiefen und anzuwenden.
Das Handling der Daten hat zu Beginn etwas an Denkarbeit benötigt, da die Klassifizierung nicht 100mal pro Stadt berücksichtigt werden sollte, sondern nur einmal, was sich dann abschließend durch die Kombination aus der Stadt-Klasse und dem StadtManager bewerkstelligen ließ ohne diese Daten immer wieder irgendwo neu bereit stellen zu müssen nach dem Initialisieren des Hauptfensters.
Durch die Sammlung aller Daten bezüglich der Städte wäre es weiterführend möglich noch andere Daten auszuwerten, ob sich beispielsweise Beziehungen zwischen globaler Lage der Stadt und Temperaturentwicklung ableiten ließen. Es wäre auch möglich die Städte auf einer Karte anzuzeigen und die Trends entsprechend zum Beispiel mit Farben darzustellen, um eine visuelle Darstellung der Informationen für den Nutzer bereit zu stellen.