ORIGINAL ARTICLE
DOI: https://doi.org/10.30932/1992-3252-2024-22-2-2

World of Transport and Transportation, 2024,
Vol. 22, Iss. 2 (111), pp. 156–168

Traffic Flow Intensity Research Based on Deep Learning

Igor N. PUGACHEV

Nikolay G. SHESHERA

Denis E. GRIGOROV

Igor N. Pugachev[1], Nikolay G. Sheshera[2], Denis E. Grigorov[3]

[1] Khabarovsk Federal Research Centre of the Far Eastern Branch of the Russian Academy of Sciences, Khabarovsk, Russia.
[2, 3] Far Eastern Law Institute of the Ministry of Internal Affairs of the Russian Federation, Khabarovsk, Russia.
[1] ORCID 0000-0003-0345-4350; Web of Science Researcher ID: ABY-8399-2022; Scopus Author ID: 56386223400; Russian Science Citation Index SPIN-code: 1856-1556; Russian Science Citation Index Author ID: 416392.
[2] ORCID 0009-0006-3302-5572; Web of Science Researcher ID: LCI-5197-2024; Scopus Author ID: 57209470019; Russian Science Citation Index SPIN-code: 9869-8822; Russian Science Citation Index Author ID: 1033608.
[3] ORCID 0009-0005-4049-9488; Scopus Author ID: 57209470019; Russian Science Citation Index SPIN-code: 6146-4152; Russian Science Citation Index Author ID: 1084181.
✉ *[2] kolyaka239@mail.ru.*

ABSTRACT

In a harmonious transport system, traffic flows are rationally distributed depending on the capacity of roads and streets to ensure transit capacity, considering the traffic light control systems. At the same time, due attention is not paid to changes in weather and natural conditions, which in turn significantly adjusts driving regimes, taking them out of a stable, predictable state. Modern software and hardware systems and information resources of large cities have a wide range of recorded indicators that affect distribution of traffic flows. Their automated processing using algorithmic machine learning tools has formed a comprehensive understanding of the patterns of change in the traffic intensity indicator, which is a new stage of improving road traffic safety, striving for zero mortality.

The scientific novelty of the study refers to the techniques and approaches to studying the weather and climate characteristics and factors of the street-and-road network, their preliminary processing using modern statistical and logical methods of normalisation and eliminating random outliers.

The deep learning method opens wide opportunities for analysing the intensity of the road traffic flow. By processing large amounts of data, such algorithms are able to identify complex patterns and relationships, which improves traffic forecasting and optimises traffic management. For correct operation of the neural network for training the model and studying the road traffic flow intensity, a set of software tools for preliminary data processing has been developed, which includes a step-by-step analysis of array structures with subsequent replacement of values or elimination of errors.

Preliminary data cleaning in accordance with the syntax of the program logic and the rules of statistical analysis is followed by application of a method for searching and eliminating anomalies was used, i.e. the isolation forest method.

This research direction was part of a large study on road traffic flow intensity, and the described results are a set of solutions based on the system interaction of software and methods of statistical and analytical transformations developed by the authors.

Keywords: transport systems, traffic flow intensity, deep learning method, weather and natural conditions, neural networks, model.

For citation: Pugachev, I. N., Sheshera, N. G., Grigorov, D. E. Traffic Flow Intensity Research Based on Deep Learning. World of Transport and Transportation, 2024, Vol. 22, Iss. 2 (111), pp. 156–168. DOI: https://doi.org/10.30932/1992-3252-2024-22-2-2.

The original text of the article in Russian is published in the first part of the issue.
Текст статьи на русском языке публикуется в первой части данного выпуска.

INTRODUCTION

Modern trends in development of information technology offer a wide range of data analysis capabilities. The presence of large number of features make it possible to form various patterns of transport environment distribution and consider weather conditions in each respective environment[1] [1].

Artificial intelligence (AI) is one of the promising areas in modern science and technology. In recent decades, systems have been created that are capable with the use of AI to perform complex tasks requiring intellectual skills [2; 3].

Modern AI is able to analyse the environment and interact with it, interpret and process data, learn and adapt itself while evolving. That is why the main task of the work is to prepare data for an objective assessment of the phenomenon under study.

AI is used in many areas, including computer vision, natural language processing, robotics, autonomous vehicles, medicine, financial analytics, etc. In each of these areas, AI makes it possible to automate processes, improve forecasts and make decisions based on large amounts of data and analysis of complex models.

The principles of AI are based on various approaches and methods for creating computer systems that can analyse data, learn and make decisions based on the information received. Machine learning methods, neural networks and deep learning are the main tools in achieving this goal.

In this paper, to predict the intensity of road traffic flow, the properties of the street-and-road network (SRN) are considered to be a classifier for categorising transport facilities, and weather conditions are assumed to be independent variables by which changes in the dependent will be predicted [4].

The work contains complex programming elements. The initial operations are explained, and for the rest of this kind, to reduce duplicating information, comments are omitted.

The working window of the PyCharm development environment has numbered lines. The commands are explained by referring to the line number of the provided screenshot of the workspace.

[1] ODM 218.4.005–2010. Industry road methodological document. Recommendations for ensuring road traffic safety on motorways.

RESULTS

Backpropagation Neural Network Architecture

To confirm the influence of weather conditions, road geometry and traffic characteristics on traffic flow intensity, as well as to test the predictive capabilities, a Backpropagation neural network was used[2].

A Backpropagation neural network is a type of neural network that uses the backward propagation of errors algorithm for training. The basic idea is that the network goes through two stages: forward propagation, where the input data is transferred from the input layer through the hidden layers to the output layer, and backpropagation, where the network weights are adjusted [5; 6].

The backward propagation of errors process begins with calculating the error at the output layer. This error is then propagated back through the layers of the network, proportional to the weights of the connections between neurons, and the error at each layer is calculated [8; 9]. Next, the neuron weights are adjusted in the direction opposite to the error gradient to reduce the error on subsequent passes through the network.

For the backpropagation algorithm, it is necessary to define the error function. It is measured by the difference between the expected values and the predicted ones. The mean squared error is the most common for regression problems and cross entropy for classification problems [10].

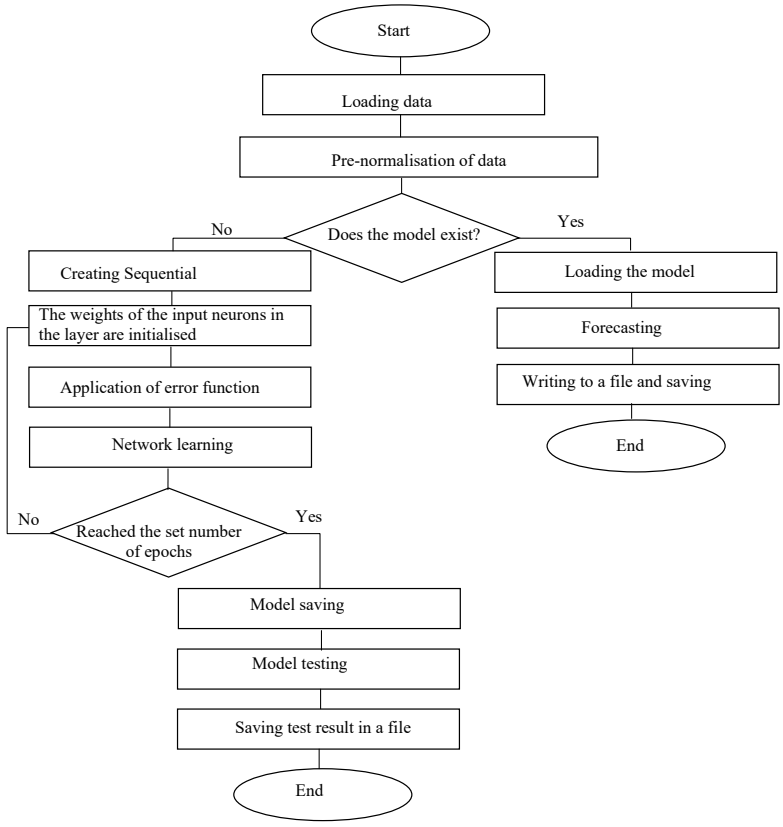
During training, the network goes through several epochs, where each epoch is one pass through the entire training set, i.e., all training examples [11; 12]. The network weights are adjusted in such a way as to minimise the error relative to the training data.

It was decided to form the architecture of the neural network in the Python programming language, using the Keras add-on of the low-level TensorFlow library [13].

Keras was developed to facilitate and speed up the process of developing deep learning models. It provides high-level abstractions for building neural networks, hiding most of the complexity and details of low-level libraries from the developer.

The architecture of a backpropagation neural network consists of several main components.

[2] Operation of motorways: Study guide. Comp. by: I. N. Pugachev, A. V. Kamenchukov, N. S. Nesterova. Khabarovsk, Publishing house of FESTU, 2022, 168 p.



Pic. 1. Block diagram of a backpropagation neural network [performed by the authors].

1. *Input layer* receives as input data that is fed to the input of the neural network. In this layer, the number of neurons corresponds to the dimensionality of the incoming data.

2. *Hidden layers* perform intermediate computational operations. Depending on the specific task or architecture of the neural network, the number of hidden layers and neurons in them changes. Each neuron in the hidden layer takes the activation values of the previous layer as input and calculates its activation value based on the weights that connect it to the previous layer.

3. *Output layer* performs a forecast. The number of neurons in the output layer depends on the classification or regression problem. For example, if a binary classification problem is being solved, then the output layer will have one neuron responsible for the probability of belonging to a class.

4. *Network activation function* is used for nonlinear transformation of the input data. The most common activation functions are sigmoid, tanh and ReLU.

5. *Backward propagation of errors* is an algorithm for updating the weights in a neural network. It calculates the gradient of the error function over all the weights and uses it to update the weights to minimise the prediction error.

6. *Optimiser* applies an optimisation algorithm to update the weights in a neural network based on the error gradient. Some popular optimisers include stochastic gradient descent (SGD), Adam, and RMSprop.

7. *Loss function* determines how well the neural network performs on the task at hand. It calculates the difference between the predicted values and the true values and is used in backpropagation to calculate the gradient.

The main components of the backpropagation neural network architecture can be changed and modified depending on the specific task and requirements [14; 15].

The block diagram of the neural network is shown in Pic. 1.

Initially, the neural network structure was based on data analysis without preliminary processing and validation [16]. Testing the model after training showed the ineffectiveness of the forecast. This is due to the large number of

```
import os
import pandas as pd

df = pd.read_csv(os.getcwd() + '/dataset.csv', sep=";")
n = df.groupby('frontier') #Сортруем по рубежам
grp_idx = n.groups
for key in grp_idx.keys():
    r = n.get_group(key)
    #r.drop(columns=r.columns[0], axis=1, inplace=True)  # убираем лишний столбец
    #r.drop(columns=r.columns[16], axis=1, inplace=True)  # убираем лишний столбец
    r.to_csv(os.getcwd() + '/output/r/' + str(key) + '.csv', sep=";", encoding="utf-8-sig")
```

Pic. 2. Explode_start module. Sorting data by reference points and generating files [performed by the authors].

incoming neurons. Therefore, it was decided to group quantitative features. As a result, it was necessary to change the approaches to creating a neural network [17].

To implement the program according to the above algorithm, a virtual environment was created in the PyCharm integrated development environment. This had to be done primarily for more convenient management of project dependencies, and to make the development process more convenient and focused.

The created virtual environment contains modules (files with the extension «py»). They are used to manage project dependencies, create an isolated development environment, and to work with different versions of packages and libraries.

The project consists of three stages:

1. Data generation.
2. Data validation.
3. Model learning.

Data Generation

The statistical data were imported into the project's virtual environment directory in the *csv* file format. This extension is used for convenience of storing data (in text form) so that it is easy to process and transfer it [18].

To study the influence of weather, natural conditions and road geometry on the intensity of road traffic flow, 562612 hours with different characteristics from 107 reference points were collected and grouped. Preliminary processing of the entire array of data showed poor results for the influence of independent features on the dependent one. Explaining this by the insufficient list of analysed factors, it was decided to analyse each reference point separately, with the prospect of creating not one, but 107 models for forecasting. Thus, the influence of unaccounted static independent variables was to be excluded [19; 20].

To achieve the above goal, considering the rules for processing by a neural network of a single file with all the data, it was necessary to form 107 files with the *csv* extension. In each of them, the data of a specific boundary.

To sort the general population by reference points with subsequent formation of files, the *explode_start.py* algorithm (Pic. 2), located in the root of the application, was created and launched.

In the *explode_start* module, the operating system functions were imported – *OS* (*import os*) to ensure reading of files and the *pandas* library (*import pandas as pd*) to work with the data (Pic. 2). Additionally, an *alias* is applied to the library (renaming for easy access to the library), simplifying its name to *pd*.

In the fourth line, the program «reads» the file, the a semicolon separator type was additionally specified.

In the fifth line, grouping is done by reference points.

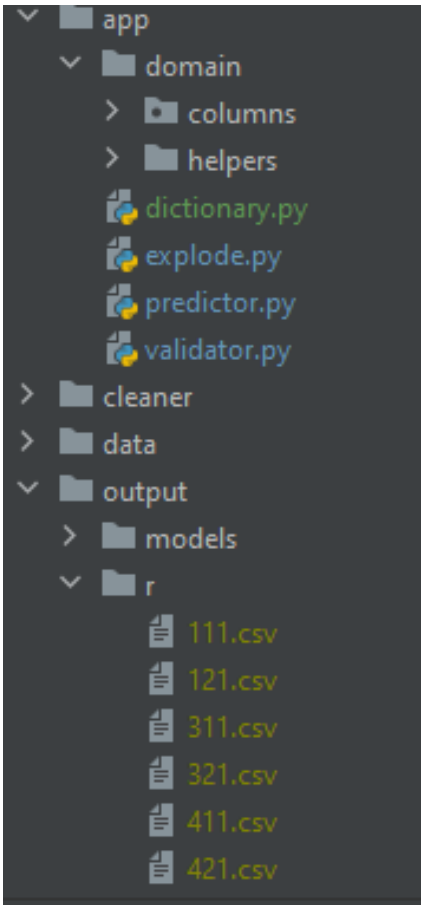
In the sixth line, the program receives a list of reference points.

From the seventh line, the program cyclically alternately searches reference points and saves them in separate files, writing their number in the file name.

After executing the *explode_start* script and following the dynamic route *'/output/r/'* (Pic. 2), files with data appear in the *r* directory (Pic. 3).

Data Preparation

After forming files with reference points, to eliminate errors during training, it is necessary to prepare the data, which will include checking for missing values in a position (warn the operator, replace with 0, delete the entire line, etc.) and for the range of acceptable values, validation, etc. [21].



Pic. 3. Generated files with reference points in the project's virtual environment directory [performed by the authors].

To prepare the data, the *validate_start* algorithm was created and launched (Pic. 4).

The command «*from app.validator import start as validate*» imports the validation function «*start*» from the «*validator*» module of the «*app*» directory (Pic. 5).

In the *validate_start* algorithm cycle (Pic. 4), the program gets a list of files from the */output/r directory*, going through them one by one. In the sixth line, the file name is separated by a period separator. As a result, an array of two parts is obtained. The first part (zero) is the file name, the second part (one) is the extension. Only the file name is considered in this function.

In the eighth line, using the *start* function (renamed to *predict*), validation is launched from the *validator* module (Pic. 5).

```
1  import os
2
3  from app.validator import start as validate
4
5  for file in os.listdir(os.getcwd() + '/output/r'):
6      split = os.path.splitext(file)
7      filename = split[0]
8      validate(True, filename)
```

Pic. 4. Validate_start script [performed by the authors].

```python
import os
from app.domain.columns import names_input
from app.domain.columns import names_output
from app.domain.columns import validators as get_validators
from app.domain.helpers import logger
import pandas as pd
NSheshera *
def start(need_train, filename):
    # 1-й шаг - готовим данные
    data = pd.read_csv(os.getcwd() + '/output/r/'+filename+'.csv', sep=';')
    validators = get_validators()
    new_array = dict()
    for data_name, data_values in data.items():
        if data_name in validators:
            l = validators[data_name]
            s = data_values
            encoded = list(map(l, s))
            new_array[data_name] = encoded
            logger.success(''' + str(data_name) + ''' + ' validated success!')
    df = pd.DataFrame(new_array)
    if need_train:
        d = names_input() + names_output()
    else:
        d = names_input()
    df.to_csv(os.getcwd() + '/output/validated/'+filename+'.csv', sep=";",
              columns=d, encoding='utf-8-sig')
```

Pic. 5. The start validation function from the validator module in the app directory [performed by the authors].

```python
# Колонки names_input которые использовать в анализе данных
5 usages  NSheshera
def names_input():
    return _columns_inputs
2 usages  NSheshera
def names_output():
    return _columns_outputs
# Получить список колонка-валидатор
NSheshera *
def validators():
    return dict(zip(_columns_inputs+_columns_outputs,
                    _validators_input+_validators_output))
```

Pic. 6. Method names_input of the module _init_ [performed by the authors].

When executing the algorithm of the *validator* module program, auxiliary functions and variables are imported from the *columns* directory (*app/domain/columns*) and the *logger* module is imported from the *helpers* directory (*app/domain/helpers*). The task of the *logger* module is to change the font colour of the message that is output to the console (the design details are not important for the purpose of the work, so the details of the algorithm of this module are omitted).

```
# Колонки по порядку
_columns_inputs = [
    WEEK_COLUMN_NAME,# День недели
    TIME_COLUMN_NAME,#   Время суток
    T_AIR_COLUMN_NAME,#   Температура воздуха
    T_SOIL_COLUMN_NAME,#   Температура почвы
    T_DEW_COLUMN_NAME,#   Температура точки росы
    PRESSURE_COLUMN_NAME,# Парциальное давление водяного пара (pa), Па
    F_PERCENT_COLUMN_NAME,# Относительная влажность воздуха (φ), %
    VISIBILITY_CIPHER_COLUMN_NAME,# Видимость, шифр (VV):
    SATURATION_DEFICIT_COLUMN_NAME,# Дефицит насыщения (d), г/м³
    P_LEVEL_STARION_COLUMN_NAME,# Атмосферное давление на уровне станции (P станции), г.Па
    P_SEA_LEVEL_COLUMN_NAME,# Атмосферное давление на уровне моря (P моря), г.Па
    WEATHER_CODE_WW_COLUMN_NAME,# Погода, шифр (ww)
    DIRECTION_WIND_COLUMN_NAME,#направление ветра, градусы
    WIND_SPEED_COLUMN_NAME,# Скорость ветра, м/с
    PRECIPITATION_COLUMN_NAME,# Осадки, мм
    DAYLIGHT_COLUMN_NAME,# Естественное освещение...
]
_columns_outputs = [INTENSIVE_COLUMN_NAME]
```

Pic. 7. List of column names of input and output parameters in the _init_ module [performed by the authors].

```
# Валидаторы по порядку
_validators_input = [
    week_validator,# День недели
    time_validator,#   Время суток
    t_air_validator,#   Температура воздуха
    t_soil_validator,#   Температура почвы
    t_dew_validator,#   Температура точки росы
    pressure_validator,# Парциальное давление водяного пара (pa), Па
    f_percent_validator,# Относительная влажность воздуха (φ), %
    visibility_cipher_validator,# Видимость, шифр (VV):
    saturation_deficit_validator,# Дефицит насыщения (d), г/м³
    p_level_station_validator,# Атмосферное давление на уровне станции (P станции), г.Па
    p_sea_level_validator,# Атмосферное давление на уровне моря (P моря), г.Па
    weather_code_ww_validator,# Погода, шифр (ww)
    direction_wind_validator,#направление ветра, градусы
    wind_speed_validator,# Скорость ветра, м/с
    precipitation_validator,# Осадки, мм
    daylight_validator,# Естественное освещение...]
_validators_output = [intensive_validator]
```

Pic. 8. List of column names of input and output parameters in the _init_ module [performed by the authors].

Considering that *columns* is a directory and not an executable file, the *_init_* module is initialised in it by default. Accordingly, line 2 (Pic. 5) imports the *names_input* and *names_output* methods from the *_init_* module into the *validator* module – this is encapsulation, one of the principles of object-oriented programming, hiding the implementation to ensure the possibility of introducing additional logic into the formation of *names_input* and *names_output* (a list of column names of input parameters that are used in the data header) (Pics. 6, 7).

The *validators* method in the *_init_* module returns a list of all validators, which are within two constants that are data arrays (Pic. 8):

1. *_columns_inputs* + *_columns_outputs*;
2. *_validators_input* + *_validators_output*.

After data processing, these arrays are concatenated and a correspondence is established between the column name and its validator. This means that the order of the validator in the constant – *validators_input* must correspond to the order of its column.

Each column from the lists of columns *(_columns_inputs, _columns_outputs, _validators_input, _validators_output*) is described in the columns directory and processed in the *_init_* module using the methods shown in Pic. 4. To ensure this, the modules and methods for data validation were imported into the *_init_* module using the command *from app.domain.columns*:

1. Day of the week:

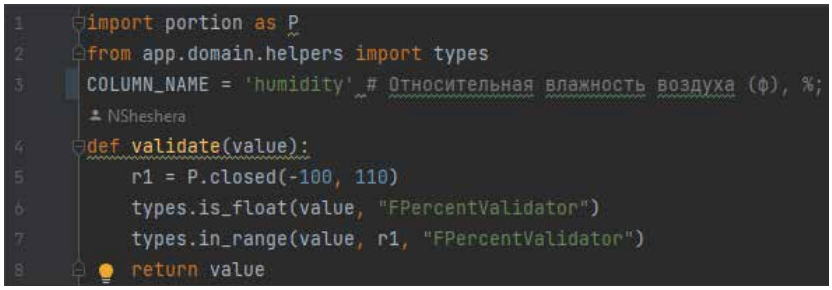
– week_day import validate as week_validator;

– week_day import COLUMN_NAME as WEEK_COLUMN_NAME.

2. Time of the day:

– time import COLUMN_NAME as TIME_COLUMN_NAME;

– time import validate as time_validator



Pic. 9. Structure of the error control module of the f_percent module [performed by the authors].

3. Road traffic intensity:

– intensity import COLUMN_NAME as INTENSIVE_COLUMN_NAME;

– intensity import validate as intensive_validator.

4. Air temperature:

– t_air import COLUMN_NAME as T_AIR_COLUMN_NAME;

– t_air import validate as t_air_validator.

5. Soil temperature:

– t_soil import COLUMN_NAME as T_SOIL_COLUMN_NAME;

– t_soil import validate as t_soil_validator.

6. Dew point temperature:

– t_dew import COLUMN_NAME as T_DEW_COLUMN_NAME;

– t_dew import validate as t_dew_validator.

7. Partial pressure of water vapor (ps), Pa:

– pressure import COLUMN_NAME as PRESSURE_COLUMN_NAME;

– columns.pressure import validate as pressure_validator.

8. Relative humidity (ϕ), %:

– f_percent import COLUMN_NAME as F_PERCENT_COLUMN_NAME;

– f_percent import validate as f_percent_validator.

9. Visibility, cipher (VV):

– from app.domain.columns.visibility_cipher import COLUMN_NAME as VISIBILITY_CIPHER_COLUMN_NAME;

– visibility_cipher import validate as visibility_cipher_validator.

10. Saturation deficit (d), g/m³:

– columns.saturation_deficit import COLUMN_NAME as SATURATION_DEFICIT_COLUMN_NAME;

– saturation_deficit import validate as saturation_deficit_validator.

11. Atmospheric pressure at the station level (P station), g.Pa

– p_level_station import COLUMN_NAME as P_LEVEL_STARION_COLUMN_NAME;

– p_level_station import validate as p_level_station_validator.

12. Atmospheric pressure at the sea level (P sea), g.Pa:

– p_sea_level import COLUMN_NAME as P_SEA_LEVEL_COLUMN_NAME;

– p_sea_level import validate as p_sea_level_validator.

13. Weather, cipher (ww):

– weather_code_ww import COLUMN_NAME as WEATHER_CODE_WW_COLUMN_NAME;

– weather_code_ww import validate as weather_code_ww_validator.

14. Wind direction, degrees:

– direction_wind import COLUMN_NAME as DIRECTION_WIND_COLUMN_NAME;

– direction_wind import validate as direction_wind_validator.

15. Wind speed, m/s:

– wind_speed import COLUMN_NAME as WIND_SPEED_COLUMN_NAME;

– wind_speed import validate as wind_speed_validator.

16. Precipitation, mm:

– precipitation import COLUMN_NAME as PRECIPITATION_COLUMN_NAME;

– precipitation import validate as precipitation_validator.

17. Natural light:

– daylight import COLUMN_NAME as DAYLIGHT_COLUMN_NAME;

– daylight import validate as daylight_validator.

In the ninth line, the *start* function of the validator module of the *app/domain/columns* directory reads the *csv* file (Pic. 5). The path to the file is formed according to the template line (*os.getcwd() + '/output/r/'*).

The *filename* variable is the name of the file, it is passed as an argument to this function, and *need_train* is an indicator that validation is performed at the model training level or at the

forecasting level (Pic. 5). The fact is that validation is used both at the model training stage and at the forecasting stage.

Using validators in the *start* function of the validator module (Pic. 5), new verified data is formed, which is written to the directory – */utput/validated*.

The structure of the module itself, reflecting information about the column using the example of the variable – «Relative air humidity» is shown in Pic. 9.

Line 2 imports *helpers*, which denotes custom project functions for checking the types of the *csv* table column value.

The constant *COLUMN_NAME* in the fourth line binds the name of the column header. In this case, it is «humidity».

From the fifth to the eighth lines, the *validate* function checks the values for compliance with the established requirements, otherwise the program reports an error. In this specific example (Pic. 9), the values in the humidity column should not be lower than 100 or higher than 110. Other values will be considered an error.

The rules for data control in the Python programming language for *csv* file columns at the preparation stage are given below.

Natural light, daylight module

```
import math
import app.domain.helpers.utils as f
COLUMN_NAME = 'daylight' #Natural light
def validate(value):
if 0.5 < value < 1:
value = float(f.toFixed(value, 2))
types.in_range(value, [0, 0.5, 0.85, 1],
«Daylight_Validator»)
return value
```

Wind direction, degrees, direction_wind module

```
import portion as P
from app.domain.helpers import types COLUMN_NAME
= 'wind_direction' #Wind direction, degrees
def validate(value):
r1 = P.closed(0, 360)
types.is_float(value, «direction_windValidator»)
types.in_range(value, r1, «direction_windValidator»)
return value
```

Relative humidity, f_percent module

```
import portion as P
from app.domain.helpers import types
COLUMN_NAME = 'humidity' # Relative humidity (ϕ),
%;
def validate(value):
r1 = P.closed(–100, 110)
types.is_float(value, «FPercentValidator»)
types.in_range(value, r1, «FPercentValidator»)
return value
```

Traffic flow intensity, intensity module

```
import portion as P
from app.domain.helpers import types
COLUMN_NAME = 'intensity' # Intensity
def validate(value):
r1 = P.closed(0, float('inf'))
types.is_int(value, «IntensivityValidator»)
types.in_range(value, r1, «IntensivityValidator»)
return value
```

Atmospheric pressure at the station level, p_level_station module

```
import portion as P
from app.domain.helpers import types
COLUMN_NAME = 'pressure_station' # Atmospheric
pressure at the station level (P station), g.Pa
def validate(value):
r1 = P.closed(900, 1100)
types.is_float(value, «p_level_stationValidator»)
types.in_range(value, r1, «p_level_stationValidator»)
return value
```

Atmospheric pressure at the sea level, p_ sea_level module

```
import portion as P
from app.domain.helpers import types
COLUMN_NAME = 'pressure_sea' # Atmospheric
pressure at the sea level (P sea), g.Pa
def validate(value):
r1 = P.closed(900, 1100)
types.is_float(value, «p_sea_levelValidator»)
types.in_range(value, r1, «p_sea_leveValidator»)
return value
```

Precipitation, precipitation module

```
import portion as P
from app.domain.helpers import types
import math
COLUMN_NAME = 'precipitation' # Precipitation, mm
def validate(value):
if math.isnan(value):
value = float(0)
r1 = P.closed(0, 100)
types.is_float(value, «Precipitation_Validator»)
types.in_range(value, r1, «Precipitation_Validator»)
return value
```

Partial pressure of water vapour, pressure module

```
import portion as P
from app.domain.helpers import types
COLUMN_NAME = 'partial_pressure' # Partial pressure
of water vapor (ps), Pa
def validate(value):
r1 = P.closed(0, float('inf'))
types.is_float(value, «PressureValidator»)
types.in_range(value, r1, «PressureValidator»)
return value
```

Saturation deficit, saturation_deficit module

```
import portion as P
from app.domain.helpers import types
COLUMN_NAME = 'saturation_deficit' # Saturation
deficit (d), g/m³
def validate(value):
r1 = P.closed(0, 40)
```

Table 1

Fragment of data from model assessment using test loss and test acc methods before and after excluding anomalies from the traffic flow intensity indicator [compiled by the authors]

Reference point	With anomalies		Without anomalies	
	test loss	test acc	test loss	test acc
1011	0.21	0.44	0.36	0.69
1021	0.26	0.38	0.30	0.67
1031	0.25	0.31	0.62	0.52
111	0.52	0.26	0.89	0.43
1111	0.42	0.24	0.74	0.20
1121	0.25	0.35	0.33	0.65
1131	0.23	0.40	0.32	0.62
1141	0.25	0.47	0.40	0.60

```
types.is_float(value, «SaturationValidator»)
types.in_range(value, r1, «SaturationValidator»)
return value
```

Air temperature (t air), t_air module

```
import portion as P
from app.domain.helpers import types
COLUMN_NAME = 't_air' #Air temperature (t air), C°
def validate(value):
r1 = P.closed(–100, 100)
types.is_float(value, «TAirValidator»)
types.in_range(value, r1, «TAirValidator»)
return value
```

Dew point (DP) temperature, t_dew module

```
import portion as P
from app.domain.helpers import types
COLUMN_NAME = 't_dew_point' # Dew point
temperature (DP), C°
def validate(value):
r1 = P.closed(–100, 100)
types.is_float(value, «TDewValidator»)
types.in_range(value, r1, «TDewValidator»)
return value
```

Soil temperature (t soil), t_soil module

```
import portion as P
from app.domain.helpers import types
COLUMN_NAME = 't_soil' # Soil temperature (t soil), C°
def validate(value):
r1 = P.closed(–100, 100)
types.is_float(value, «TSoilValidator»)
types.in_range(value, r1, «TSoilValidator»)
return value
```

Time of the day, time module

```
import portion as P
from app.domain.helpers import types
COLUMN_NAME = 'day_time' # Time of the day
def validate(value):
r1 = P.closed(0, 23)
types.is_int(value, «TimeValidator»)
types.in_range(value, r1, «TimeValidator»)
return value
```

Visibility, cipher (VV), visibility_cipher module

```
from app.domain.helpers import types
import math
COLUMN_NAME = 'visibility_VV' # Visibility, cipher (VV);
def validate(value):
if math.isnan(value):
value = 0
types.is_int(int(value), «VisibilityValidator»)
types.in_range(value, [90, 91, 92, 93, 94, 95, 96, 97, 98,
99, 0], «VisibilityValidator»)
return value
```

Weather, cipher (ww), weather_code_ww module

```
from app.domain.helpers import types
import math
COLUMN_NAME = 'weather_WW' # Weather, cipher
(ww)
def validate(value):
if math.isnan(value):
value = 0
types.is_int(int(value), «Weather_code_ww_Validator»)
types.in_range(value, [0, 1, 2, 3, … 99], «Weather_code_
ww_Validator»)
return value
```

Day of the week, week_day module

```
import portion as P
from app.domain.helpers import types
COLUMN_NAME = 'week_day' # Day of the week
def validate(value):
r1 = P.closed(1, 7)
types.is_int(value, «WeekValidator»)
types.in_range(value, r1, «WeekValidator»)
return value
```

Wind speed, wind_speed module

```
import portion as P
from app.domain.helpers import types
import math
COLUMN_NAME = 'wind_speed' # Wind speed, m/s
def validate(value):
if math.isnan(value):
value = float(0)
r1 = P.closed(0, 40)
types.is_float(value, «Wind_speedValidator»)
types.in_range(value, r1, «Wind_speedValidator»)
return value
```

Outliers Removal Using the Isolation Forest Method

Looking ahead, it should be noted that the decision to clean the data from outliers was

Pic. 10. Cleaning traffic flow data using the isolation forest method [performed by the authors].

a consequence of the low accuracy of the test data forecast (Table 1).

Test loss and test acc are metrics used to evaluate the performance of a machine learning model on test data.

Test loss is a measure of the error, or difference between the predicted values and the actual values on the test data. Test loss is usually calculated using some loss function (such as Mean Squared Error or Cross-Entropy Loss) and shows how well the model does at predicting the correct values on the test data. The lower is the test loss, the better is the model's performance.

Test accuracy is the percentage of correct predictions by the model on the test data. It shows how well the model classifies the correct classes on the test data. Test accuracy is calculated by comparing the predicted values with the actual class labels and calculating the proportion of correct answers. The higher is the test accuracy, the better is the model's performance.

Both test loss and test accuracy are used to assess the performance of a model on data that was not used for learning. They make it possible to check how well the model generalises knowledge and is able to predict the correct values on new data.

All factors were checked for anomalies, but only the traffic flow intensity showed significant contamination of the data with random outliers. These are consequences of road traffic accidents, maintenance of road itself or of adjacent structures with temporary restraints of traffic, either of a breakdown of the recording device, etc.

The Isolation Forest method turned to be the best choice for finding anomalies because the traffic flow metric does not have a clearly defined normal distribution.

Isolation Forest is a machine learning algorithm for detecting anomalies in data. Unlike other algorithms, such as clustering or density, Isolation Forest uses decision trees to find anomalies.

The basic idea behind Isolation Forest is that anomalies in data tend to have fewer connections and shorter distances to other features. The algorithm builds multiple decision trees by randomly selecting features and splitting the data at each step. It then estimates how quickly an anomaly can be isolated in these trees.

When learning an isolation forest:

1. The maximum tree depth and maximum number of splits are specified.

2. The data is split into random subsamples and a decision tree is built for each subsample.

3. Each decision tree splits the data into two parts by choosing a random feature and a random split.

4. Steps 2–3 are repeated until the maximum depth or number of splits is reached.

5. At the end of learning and for a new feature, the validation score is used to determine anomaly.

When using isolation forest for anomaly detection:

1. A «path» in the tree is computed for each feature, representing the number of splits needed to isolate the feature.

2. The average path length is computed for all features and used to determine anomalies. Features with shorter paths are usually considered more anomalous.

Advantages of using isolation forest:

1. It can be well scaled to handle large amounts of data.

2. It is powerful and efficient in detecting anomalies.

3. It does not require preliminary data preparation, as it is insensitive to outliers and multicollinearity.

The existing method of implementing the random forest method in the Python programming language requires certain versions of special libraries of the program itself, which are not compatible with this project, so an additional virtual environment with the key module *clean_start* was created in the PyCharm development environment (Pic. 10).

When the *clean_start* module is launched, the isolation forest method of the *sklearn* library ensemble add-in is imported in the third line. A cyclic function is launched for each reference point from the */output/validated* directory and the *cleaner* function is processed (Pic. 10), where:

– Data is loaded in the sixth line.

– The first index column is deleted in the seventh line.

– The model is formed, and parameters are set in the tenth line.

– Data on traffic flow intensity is transferred to the model in the eleventh line.

– Anomalies are identified and marked in lines 12–13. Two additional columns are created. One reflects the scores of values (*scores_intensity*), the second (*anomaly_intensity*) marks all anomalous values with the number «-1» and normal values with «1» according to the scores.

At the end of each cycle, the transformed data is written back to the */output/validated* directory.

To eliminate anomalies, before learning, the data was sorted by the value «1» of the column *anomaly_intensity*, using the command *data_frame = data_frame.loc[data_frame['anomaly_intensity'] == 1]*.

CONCLUSION

Preliminary data transformation in order to prepare it for processing by machine learning methods is an important component of achieving the accuracy of the obtained results, objective comparison of features and identification of dependencies.

The use of information and analytical resources of large cities is becoming promising for the analysis of road traffic flow intensity. For this purpose, the study has used the algorithmic tools based on the obtained theoretical results and software prototypes of their main components, methods and algorithms for predictive analytics of the Python programming language, and the PyCharm development environment.

Further implementation of the research results is aimed at building a deep learning neural network, model learning and forecasting, therefore, pre-processing is an important component of the program logic, and allows improving data quality, simplifying the model and making learning more effective.

Due to the high dynamism of road traffic flow intensity and a large number of unstable parameters affecting it, unpredictable (anomalous) values are observed, which include accidents, construction work, failures of recording equipment, etc. To exclude them, preliminary data processing was followed by processing the data by the Isolation Forest method.

The Isolation Forest was used to find anomalies by constructing an ensemble of isolated trees, in each of which partitions were randomly selected by random features and data points for constructing the tree. Then, anomalies were eliminated by reducing the height of anomalous points in the tree.

This work became a platform for training models on the dependence of traffic flow intensity on weather and climate characteristics and road network factors using the deep learning method. Table 1 demonstrates the quality of the neural network model before and after data processing, including with the Isolation Forest method. The research results have been tested and scientifically weighed, indicating an increase in the quality of the predictive model on test data.

The work is deemed to be relevant from theoretical and practical points of view, and already has application examples that will be published in future works.

REFERENCES

1. Babkov, V. F. Road conditions and traffic safety [*Dorozhnie usloviya i bezopasnost dvizheniya*]. Moscow, Transport publ., 1993, 271 p. ISBN 5-277-01402-0.

2. Pavlyuk, D. Robust and Responsive Learning of Spatiotemporal Urban Traffic Flow Relationships. *IEEE Transactions on Intelligent Transportation Systems,* 2022, Vol. 23, Iss. 9, pp. 14524–14541. DOI: 10.1109/TITS.2021.3130146.

3. Oktarina, Yu., Sastiani, D. Z., Dewi, T. Simulation Design of Artificial Intelligence Controlled Goods Transport Robot. *Computer Engineering and Applications Journal*, 2022, Vol. 11, Iss. 2, pp. 155–165. DOI: 10.18495/COMENGAPP.V11I2.411.

4. Pugachev, I. N., Shcheglov, V. I. Implementation of programs for integrated development of transport infrastructures of agglomerations and neighbouring entities of the Russian Federation based on creation of an information system [*Realizatsiya program kompleksnogo razvitiya*

transportnykh infrastruktur aglomeratsii i sosedstvuyushchikh subektov Rossiiskoi Federatsii na osnove sozdaniya informatsionnoi sistemy]. *Transport and service*, 2021, Iss. 9, pp. 7–16. EDN: AMDQXT.

5. Lugbade, S., Ojo, S., Imoize, A. L., Isabona, J., Alaba, M. O. A Review of Artificial Intelligence and Machine Learning for Incident Detectors in Road Transport Systems. *Mathematical and Computational Applications*, 2022, Vol. 27, Iss 5, 77. DOI: 10.3390/mca27050077.

6. Moskvitin, V. M., Semenova, N. И. Noise influence on recurrent neural network with nonlinear neurons. *Izvestiya VUZ. Applied Nonlinear Dynamics*, 2023, Vol. 31, Iss. 4, pp. 484–500. DOI: https://doi.org/10.18500/0869-6632-003052.

7. Pugachev, I. N., Tormozov, V. S. Development of a new method for detection and classification of vehicles using satellite images [*Razrabotka novogo metoda detektirovaniya i klassifikatsiya transportnykh sredstv po sputnikovym izobrazheniyam*]. *Dorogi i mosty*, 2023, Iss. 49–1, pp. 199–221. [Electronic resource]: https://rosdornii.ru/upload/iblock/de5/4m83hirrtm29nzdyiknu9vg4rse3q7me/11.-Pugachev-Razrabotka-novogo-metoda.pdf. Last accessed 29.12.2023.

8. Pugachev, I. N., Skripko, P. B., Sheshera, N. G. A software approach to the integrated collection and preparation of data on vehicle traffic intensity, weather conditions and natural light in hourly intervals [*Programmniy podkhod k kompleksnomu sboru i podgotovki dannykh ob intensivnosti dvizheniya transportnykh sredstv, pogodnykh uslovii i estestvennoi osveshchennosti v chasovykh intervalakh*]. *T-Comm: Telecommunications and transport*, 2023, Vol. 17, Iss. 10, pp. 43–51. DOI: 10.36724/2072-8735-2023-17-10-43-51.

9. Kopp, T., Weitemeyer, R., Beyer, J., Ziegler, D., Hess, R. Entscheidungsunterstützung in Leitstellen des Personennahverkehrs – Technische und sozio-technische Herausforderungen [A*rtificial Intelligence for Decision Support in Local Public Transport Control Centers–Technical and Socio-technical Challenges*]. *HMD Praxis der Wirtschaftsinformatik*, 2023, Vol. 60, Iss. 6, pp. 1156–1173. DOI: https://doi.org/10.1365/s40702-023-00996-8.

10. Pugachev, I. N., Sheshera, N. G. Application of statistical analysis methods for assessing the parameters of traffic flows and characteristics of the street and road network [*Primenenie metodov statisticheskogo analiza dlya otsenki parametrov transportnykh potokov i kharakteristik ulichno-dorozhnoi seti*]. *Far Eastern Law Institute of the Ministry of Internal Affairs of Russia*. Khabarovsk, RIO DVUI MVD of Russia, 2020, 108 p. ISBN 978-5-9753-0313-4.

11. Chernih, V. S., Zhikharev, A. G., Fedoseev, A. D., Marton, N. A. Comparison of efficiency of different methods of training neural networks. *Research Result Information Technologies*, 2023, Vol. 8, Iss. 1, pp. 83–93. DOI: 10.18413/2518-1092-2022-8-1-0-8.

12. Mikhalev, O. N., Yanyushkin, A. S. Automation of technological processes based on neural network [*Avtomatizatsiya tekhnologicheskikh protsessov na osnove neironnoi seti*]. *Automation. Modern technologies*, 2022, Vol. 76, Iss. 4, pp. 147–152. EDN: DIEMHO.

13. Nguyen, H. T., Nguyen, L. T., Afanasiev, A. D., Pham, L. T. Classification of Road Pavement Defects Based on Convolution Neural Network in Keras. *Automatic Control and Computer Sciences*, 2022, Vol. 56, Iss. 1, pp. 17–25. DOI: https://doi.org/10.3103/S0146411622010084.

14. Sokerin, D. D. Introduction to Artificial Neural Networks [*Vvedenie v iskusstvennie neironnie seti*]. *Information and Education: The Frontiers of Communication*, 2023, Iss. 15 (23), pp. 284–286. EDN: WKHNMZ.

15. Pugachev, I. N., Sheshera, N. G., Shcheglov, V. I. Analysis of geometric elements of roads when assessing their accident rate by means of modern geoinformational systems. *Bulletin of civil engineers*, 2021, Iss. 3 (86), pp. 127–133. DOI: 10.23968/1999-5571-2021-18-3-127-133.

16. Arrykova, G. K., Ezizova, S. E., Garayev, G. B., Khodzhakaeva, D. M. Artificial intelligence and neural networks: modern technologies in solving key problems [*Iskusstvenniy intellect i neironnie seti: sovremennie tekhnologii v reshenii klyuchevykh problem*]. Ceteris Paribus, 2023, Iss. 12, pp. 16–18. [Electronic resource]: https://sciartel.ru/arhiv-journal/CP-2023–12.pdf?ysclid=m06mw02bpt288436451 [full text of the issue]. Last accessed 29.12.2023.

17. Shashev, D. V., Shatravin, V. V. Implementation of the sigmoid activation function using the concept of reconfigurable computing environments [*Realizatsiya sigmoidnoi funktsii aktivatsii s pomoshchyu kontseptsii perestraivaemykh vychislitelnykh sred*]. *Bulletin of Tomsk State University. Management, Computer Science and Information Technology*, 2022, Iss. 61, pp. 117–127. EDN: PDIJZM.

18. Dulesov, A. S., Baishev, A. V., Karandeev, D. Yu., Dulesova, N. V., Karandeeva, I. Yu. Preliminary processing of statistical data on the state of homogeneous technical objects [*Predvaritelnaya obrabotka statisticheskikh dannykh o sostoyanii odnorodnykh tekhnicheskikh obektov*]. *Scientific and Technical Bulletin of Volga Region,* 2023, Iss. 4, pp. 80–83. EDN: EFIDGU.

19. Rezova, N. L., Kazakovtsev, L. A., Shkaberina, G. Sh., Tsepkova, M. I. Preliminary data processing for analysing the behaviour of complex systems [*Predvaritelnaya obrabotka dannykh dlya analiza povedeniya slozhnykh system*]. *Control Systems and Information Technologies*, 2022, Iss. 2 (88), pp. 40–45. EDN: BYGESB.

20. Akimov, A. A., Valitov, D. R., Kubryak, A. I. Preliminary data processing for machine learning [*Predvaritelnaya obrabotka dannykh dlya mashinnogo obucheniya*]. *Scientific review. Technical sciences*, 2022, Iss. 2, pp. 26–31. EDN: GWGJSK.

21. Bezmenov, I. V. Method of cleaning measurement data from outliers: search for an optimal solution with a minimum number of rejected measurement results [*Metod ochistki izmeritelnykh dannykh ot vybrosov: poisk optimalnogo resheniya s minimalnym kolichestvom otbrakovanykh resultatov izmerenii*]. *Measuring equipment*, 2023, Iss. 1, pp. 16–23. EDN: KISWIN. ●

Information about the authors:

Pugachev, Igor N., *D.Sc. (Eng), Associate Professor, Deputy Director for Scientific Work of Khabarovsk Federal Research Centre of the Far Eastern Branch of the Russian Academy of Sciences (KhFRC FEB RAS), Khabarovsk, Russia, ipugachev64@mail.ru.*

Sheshera, Nikolay G., *Ph.D. (Eng), Associate Professor at the Department of Information and Technical Support of Internal Affairs Bodies of Far Eastern Law Institute of the Ministry of Internal Affairs of Russia, Khabarovsk, Russia, kolyaka239@mail.ru.*

Grigorov, Denis E., *Head of the Office of Special Disciplines of the Department of Information and Technical Support of Internal Affairs Bodies of Far Eastern Law Institute of the Ministry of Internal Affairs of Russia, Khabarovsk, Russia, glowfisch8lan@gmail.com.*

Article received 29.12.2023, approved 24.04.2024, accepted 15.05.2024.