Санкт-Петербургский политехнический университет
Высшая школа прикладной математики
и вычислительной физики, ФизМех

Направление подготовки
«Прикладная математика и информатика»

Отчет по работе
«Построение модели для предсказания поведения курсов валют»
Дисциплина: «Машинное обучение»

Выполнил:
Ольшанский М.Д.
Группа:
5040102/40201

Преподаватель:
Иванов Денис Юрьевич

Санкт-Петербург 2025

1. Постановка задачи

Необходимо реализовать модель для предсказания курсов валют стран по данным о их ВВП и импорту/экспорту.

2. Используемые материалы

Датасеты для обучения и тестирования были взяты с порталов kaggle.com и databank.worldbank.org.
Данные курсов валют: Exchange rate
Данные ВВП: GDP
Данные импорта/экспорта: Import/Export

3. Алгоритм решения

Для обучения модели были выбраны язык Python и библиотека tensorflow.
В качестве модели обучения была выбрана полносвязная нейронная сеть.

Конфигурация приложения

Конфигурация приложения указывается в файле config.py. Здесь необходимо указать пути до файлов с датасетами, пути до папок сохранения обработанных датасетов и обученных моделей, а также границы дат данных в датасетах. Все данные агрегируются в класс Settings

```python
enable_console_mode = True

dataset_dir = "datasets/"
currency_file_name = "exchange_rates_2021_2025.csv"
import_export_file_name = "export_import_2003_2024.csv"
gdp_file_name = "gdp_2020_2025.csv"

prepared_data_dir = "by_country/"
train_country = "Russia"
file_format = ".csv"
save_model_dir = "model/"
model_file_format = ".keras"

exchange_rates_year_from = 2021
exchange_rates_year_to = 2025

gdp_year_from = 2020
gdp_year_to = 2025

import_export_year_from = 2003
import_export_year_to = 2024

# ================================ Automatic settings aggregator ==================================
class Settings:  1 usage  new *
    def __init__(self):  new *
        self.enable_console_mode = enable_console_mode

        self.dataset_dir = dataset_dir
        self.currency_file_name = currency_file_name
        self.import_export_file_name = import_export_file_name
        self.gdp_file_name = gdp_file_name

        self.prepared_data_dir = prepared_data_dir
        self.train_country = train_country
        self.file_format = file_format
```

Обработка данных

Обработка данных проходит в 3 этапа:

1. Загрузка и генерация данных из датасетов
2. Объединение данных по странам
3. Сохранение данных по странам в отдельные датасеты

Всю эту работу проделывает метод dataset_prepare:

```python
def dataset_prepare():  2 usages  Mihail *
    export_import = get_export_import()
    print("Export/import loaded")
    gdp = get_gdp()
    print("GDP loaded")
    exchange_rates = get_exchange_rates()
    print("Exchange rates loaded")

    min_date = max(get_min_index(export_import), get_min_index(gdp), get_min_index(exchange_rates))
    max_date = min(get_max_index(export_import), get_max_index(gdp), get_max_index(exchange_rates))

    export_import = bound_df_by_index(export_import, min_date, max_date)
    gdp = bound_df_by_index(gdp, min_date, max_date)
    exchange_rates = bound_df_by_index(exchange_rates, min_date, max_date)

    countries = list(filter(lambda x: x.strip()
                                      and any(x in s for s in export_import.columns)
                                      and any(x in s for s in gdp.columns), exchange_rates.columns.tolist()))

    for country, progress in zip(countries, range(1, len(countries) + 1)):
        df = DataFrame(index=exchange_rates.index)
        df['Rate'] = exchange_rates[country]
        df['Gdp'] = gdp[country]
        df['Import'] = export_import[f"{country} Import"]
        df['Export'] = export_import[f"{country} Export"]
        df.to_csv(f"{settings.prepared_data_dir}{country}{settings.file_format}")
        LogProgressInfo( process_name: "Prepare data", progress, len(countries))
```

Методы get_export_import, get_gdp и get_exchange_rates загружают, обрабатывают и генерируют необходимые данные из датасетов, указанных в конфигурации программы:

```
def get_export_import() -> DataFrame:  1 usage  Mihail *
    export_import = pd.read_csv(settings.import_export_filepath)
    keys = ["Country", "Series Name"] + [str(i) for i in range(settings.import_export_year_from, settings.import_export_year_to + 1)]
    export_import = export_import[keys]
    export_import = export_import[export_import["Series Name"].isin(['Imports of goods and services (current US$)', 'Exports of goods and servic
    export_import["Series Name"] = export_import["Series Name"].apply(lambda x: "Import" if "Import" in x else "Export")
    export_import.reset_index(inplace=True, drop=True)
    export_import.set_index(["Country", "Series Name"], inplace=True, drop=True)
    for ind in range(settings.import_export_year_from, settings.import_export_year_to + 1):
        export_import[str(ind)] = export_import[str(ind)].map(convert_to_float_if_possible)
    date_range = pd.date_range(start=f"{settings.import_export_year_from + 1}-01-01", end=f"{settings.import_export_year_to + 1}-12-31", freq='D

    data = DataFrame({f"{ind[0]} {ind[1]}": generate_data(
        export_import.loc[ind,export_import.loc[ind].notna().tolist()],
        date_range.shape[0],
        LogProgressInfo( process_name: "Export/Import", progress, len(export_import.index)))
        for ind, progress in zip(export_import.index, range(1, len(export_import.index) + 1))}, index=date_range)

    print()
    return data
```

```
def get_gdp() -> DataFrame:  1 usage  Mihail *
    gdp = pd.read_csv(settings.gdp_filepath)
    gdp = gdp.set_index("Country")
    for ind in range(settings.gdp_year_from, settings.gdp_year_to + 1):
        gdp[str(ind)] = gdp[str(ind)].map(convert_to_float_if_possible)
    date_range = pd.date_range(start=f"{settings.gdp_year_from + 1}-01-01", end=f"{settings.gdp_year_to + 1}-12-31", freq='D')
    data = DataFrame({country: generate_data(
        gdp.loc[country, gdp.loc[country].notna().tolist()],
        date_range.shape[0],
        LogProgressInfo( process_name: "GDP", progress, len(gdp.index)))
        for country, progress in zip(gdp.index, range(1, len(gdp.index) + 1))}, index=date_range)

    print()
    return data
```

```
def get_exchange_rates() -> DataFrame:  1 usage  Mihail *
    exchange_rates = pd.read_csv(settings.currency_filepath)
    exchange_rates.drop( labels: "Unnamed: 0", axis=1, inplace=True)
    exchange_rates["Country"] = exchange_rates["Country"].apply(lambda x: " ".join(x.split()[:-1]))
    exchange_rates["date"] = pd.to_datetime(exchange_rates["date"], format="%d/%m/%Y")
    countries = exchange_rates["Country"].unique()
    date_range = exchange_rates["date"].unique()
    data = DataFrame({country: get_country_exchange_rates(
        exchange_rates[exchange_rates["Country"] == country],
        date_range.shape[0],
        LogProgressInfo( process_name: "Exchange rates", progress, len(countries)))
        for country, progress in zip(countries, range(1, len(countries) + 1))}, index=date_range)

    print()
    return data
```

Как можно заметить, методы get_export_import, get_gdp в процессе работы вызывают метод generate_data. Это необходимо, т.к. в датасетах для импорта/экспорта и ВВП указанны суммарные данные за год. Метод generate_data по данным о годовой сумме значений генерирует данные по линейной зависимости. Также в этом методе делается сдвиг данных на год назад, т.к. в результате хочется получить модель, которая предсказывает курс валюты по данным импорта/экспорта и ВВП прошлого года:

```
def generate_data(df, target_size, progress_info: LogProgressInfo = None):  2 usa
    if progress_info is not None:
        progress_info.print()
    years = list(map(int, df.index.tolist()))
    if len(years) == 0:
        return [nan] * target_size
    years.append(years[-1] + 1)
    generated = generate_linear_data(years, df.tolist())
    if len(generated) < target_size:
        generated = [*generated, *([nan] * (target_size - generated.shape[0]))]
    elif len(generated) > target_size:
        generated= generated[:target_size]
    return generated
```

Обучение модели

Обучение модели проходит в 4 этапа:

1. Выгрузка данных по стране
2. Обучение модели
3. Сохранение обученной модели
4. Тестирование предсказания модели

Все эти этапы делает метод do_all:

```
def do_all(country: str):  2 usages  new *
    settings.train_country = country
    X_train, X_test, y_train, y_test = load_and_split_df()
    if not os.path.isfile(settings.saved_model_filepath()):
        print_separate_message("Training model")
        model = train_and_save(X_train.values, y_train)
        print_separate_message("Save model")
        print(f"Model saved to {settings.saved_model_filepath()}")
    else:
        print_separate_message("Load Model")
        model = models.load_model(settings.saved_model_filepath())
        print(f"Model loaded from file {settings.saved_model_filepath()}")

    print_separate_message("Prediction")
    predict_and_compare(model, X_test, y_test)
```

Данный метод вызывает 3 ключевых метода: load_and_split, train_and_save и predict_and_compare.

Метод load_and_split загружает данные по стране, добавляет лаги для учёта временной зависимости и разделяет их на обучающий и тестовый датасеты:

```
def load_and_split_df():  1 usage  new *
    # Загрузка данных
    df = pd.read_csv(settings.filepath(), index_col=0, parse_dates=True)

    # Создание лагов для курса валюты
    for lag in range(1, 4):
        df[f'Rate_lag_{lag}'] = df['Rate'].shift(lag)

    # Удаление строк с NaN (из-за лагов)
    df = df.dropna()

    # Разделение на признаки и целевую переменную
    X = df[['Gdp', 'Import', 'Export', 'Rate_lag_1', 'Rate_lag_2', 'Rate_lag_3']]
    y = df['Rate'].values

    # Нормализация
    scaler = MinMaxScaler()
    X_scaled = scaler.fit_transform(X)
    X.loc[:, ['Gdp', 'Import', 'Export', 'Rate_lag_1', 'Rate_lag_2', 'Rate_lag_3']] = X_scaled

    # Разделение на обучающую и тестовую выборки
    return train_test_split( *arrays: X, y, test_size=0.3, shuffle=False, random_state=42)
```

Метод train_and_save создаёт, компилирует, обучает и сохраняет модель в формате keras:

```python
def train_and_save(X_train, y_train):  1 usage  new *
    # Создание модели с учётом временных признаков
    model = Sequential([
        Dense( units: 128, activation='relu', input_shape=(X_train.shape[1],)),
        Dense( units: 64, activation='relu'),
        Dense( units: 32, activation='relu'),
        Dense(1)
    ])

    # Компиляция модели
    model.compile(optimizer='adam', loss='mse', metrics=['mae'])

    # Обучение
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2, verbose=1)

    # Сохранение
    if not os.path.isdir(settings.save_model_dir):
        print(f"Saved models directory {settings.save_model_dir} not found")
        print(f"Creating directory {settings.save_model_dir}")
        os.mkdir(settings.save_model_dir)

    if not os.path.isfile(settings.saved_model_filepath()):
        with open(settings.saved_model_filepath(), 'x') as file:
            pass

    model.save(settings.saved_model_filepath())

    return model
```

Метод predict_and_compare строит предсказание по тестовым данным, считает MSE и визуализирует полученный результат для наглядного сравнения с реальными данными

```python
def predict_and_compare(model, X_test, y_test):  1 usage  new *
    y_pred = model.predict(X_test.values)

    # Расчёт MSE
    mse = mean_squared_error(y_test, y_pred)
    print(f"Test MSE: {mse:.4f}")

    # Визуализация
    plt.figure(figsize=(12, 6))
    plt.plot( *args: X_test.index.to_numpy(), y_test, label='Real')
    plt.plot( *args: X_test.index.to_numpy(), y_pred, label='Predicted')
    plt.xlabel("Date")
    plt.ylabel("Exchange rate")
    plt.title(f"{settings.train_country} exchange rate")
    plt.legend()
    plt.tight_layout()
    plt.show()
```