КРАММЕРТИ

ИНСТРУКЦИЯ ПО УСТАНОВКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Программа для ЭВМ
«Краммерти. Модуль прогнозирования временных рядов»

1. Введение

Программа для ЭВМ «Краммерти. Модуль прогнозирования временных рядов» предназначена для автоматизированного краткосрочного прогнозирования временных рядов метеорологических параметров (и не только). Разработка данного модуля произведена в рамках развития проекта по прогнозированию температуры воздуха.

Программа разработана для нужд оперативного мониторинга и планирования, предоставляя как основной прогноз, так и расчетную температуру "по ощущениям" (с учетом ветро-холодового индекса). Она интегрирует данные из внутреннего API исторических наблюдений необходимых параметров (метеостанции, и иное имеющееся на балансе Заказчика измерительное оборудование и т.д.) а так же учитываются внешние прогностические данные сторонних сервисов.

Применяемая модель: LSTM (Long Short Term Memory) — рекуррентная нейронная сеть глубокого обучения, которая способна улавливать закономерности в данных временных рядов и использоваться для прогнозирования будущего тренда данных.

2. Требования к аппаратному Пользовательскому обеспечению:

Учитывая компоненты программного обеспечения включая: загрузка данных, предобработка Pandas, обучение LSTM-модели (TensorFlow) с HPO (HyperParameter Optimization - Optuna) и запуск API-сервера (Flask/Gunicorn).

Важно: Требования сильно зависят от:

- **Объема исторических данных:** Сколько лет данных Вы обрабатываете? Насколько они подробные (частота точек данных например раз в 10 минут, раз в час и т.д.)?
- **Сложности модели:** Количество слоев, юнитов в LSTM, длина последовательности (`sequence_length`).
- **Интенсивности HPO:** Количество trials (`N_TRIALS`) и параллелизм Optuna (если используется).
- **Нагрузки на API:** Как часто и сколько клиентов будут запрашивать прогноз?
- **Частоты переобучения:** Как часто запускается полный цикл обучения?

Минимальные требования:

Эти требования позволят запустить проект, но производительность (особенно обучение и HPO) может быть **медленной**. Подходят для разработки, тестирования или некритичных сценариев с не очень большими данными и редким переобучением.

- **Операционная система:**
  - Ubuntu 20.04 LTS (или новее) / Debian 10 (или новее) / CentOS 7 (или RHEL 7) с поддержкой необходимых версий Python и библиотек. *Ubuntu LTS является хорошим выбором.*
- **Процессор (CPU):**
  - **x86-64 архитектура**
  - **2+ ядра** (например, базовые VPS/VDS у облачных провайдеров). Одно ядро будет крайне медленным для обучения.
- **Оперативная память (RAM):**
  - **4 GB** (минимум). Pandas может потреблять много памяти при загрузке и обработке больших CSV. Обучение TF также требует памяти. 8 GB будет значительно комфортнее.
- **Дисковое пространство:**
  - **20-30 GB SSD** (минимум). Учитывайте:
    - ОС и системные файлы (~5-10 GB)
    - Виртуальное окружение Python с библиотеками (~1-3 GB, TF занимает много места)
    - Исторические данные CSV (могут быть гигабайты, зависит от объема)
    - Данные ОМ CSV (небольшой)
    - Сохраненные модели (`.keras`, метаданные, скейлеры) - могут занимать от десятков МБ до сотен МБ каждая.
    - База данных Optuna (`.db`) - может расти.
    - Логи (`app.log`, journald, nginx)
    - Промежуточные файлы (если есть).
  - **SSD крайне рекомендуется** для быстрой загрузки данных и моделей. HDD будет очень медленным.
- **Сеть:**
  - Стабильное подключение к Интернету для скачивания данных с API ваших метеостанций (ваших серверов где хранятся исторические и потоковые данные).
  - Доступ к портам (SSH, HTTP/HTTPS, порт API 3000).

- **GPU / CUDA:**
  - **Рекомендуется! Но т.к. речь идет о минимальной конфигурации - не требуется (для минимальной конфигурации).** Обучение будет происходить на CPU. Это будет **значительно медленнее**, чем на GPU, особенно для LSTM.

- **Программное обеспечение:**
  - Python 3.8 - 3.11+ (проверьте совместимость версий TensorFlow и других библиотек).

- pip и venv.
- Nginx.
- Git (рекомендуется).
- Базовые утилиты Linux (`curl`, `nano`/`vim`, etc.).

Рекомендуемые требования:

Эта конфигурация обеспечит **значительно лучшую производительность** обучения и HPO, а также позволит комфортно обслуживать API-запросы.

- **Операционная система:**
  - Ubuntu 20.04 / 22.04 LTS.
- **Процессор (CPU):**
  - **x86-64 архитектура**
  - **4+ ядер** (лучше 8+). Большее количество ядер ускорит некоторые операции Pandas и позволит запускать больше Gunicorn workers для API.
- **Оперативная память (RAM):**
  - **16 GB+** (лучше 32 GB, если данные очень большие или планируется сложная модель/HPO). Это даст запас для Pandas, TF и одновременной работы планировщика и API.
- **Дисковое пространство:**
  - **100 GB+ NVMe SSD.** NVMe значительно быстрее обычных SSD. Запас места важен для роста объемов данных, моделей и логов.
- **Сеть:**
  - Стабильное, быстрое подключение к Интернету.
- **GPU / CUDA:**
  - **КРАЙНЕ РЕКОМЕНДУЕТСЯ для ускорения обучения.**
  - **GPU:** NVIDIA GPU с поддержкой CUDA и достаточным объемом памяти (VRAM).
    - *Минимум для экспериментов:* GeForce GTX 1660 (6GB VRAM), RTX 2060 (6GB VRAM), RTX 3050 (8GB VRAM) или аналогичные/старше.
    - *Рекомендуется:* RTX 3060 (12GB VRAM), RTX 3070/3080/3090 (8GB+ VRAM), RTX 40xx серии, или профессиональные карты (Quadro/RTX A-серии) с большим объемом VRAM (12GB+). **Объем VRAM очень важен** для LSTM, особенно с длинными последовательностями и большими батчами.
  - **CUDA Toolkit:** Версия, совместимая с вашей версией TensorFlow и драйвером NVIDIA. (Устанавливается отдельно).
  - **cuDNN:** Библиотека NVIDIA для ускорения глубокого обучения. Версия, совместимая с CUDA Toolkit и TensorFlow. (Устанавливается отдельно).

    - **Драйвер NVIDIA:** Актуальный проприетарный драйвер NVIDIA для Linux.

- **Программное обеспечение:**
    - Python 3.9 - 3.11 (проверьте совместимость TF).
    - pip, venv.
    - Nginx.
    - Gunicorn.
    - Systemd (стандартно в Ubuntu).
    - Git.
    - CUDA Toolkit и cuDNN (если используется GPU).
    - Драйвер NVIDIA (если используется GPU).

Резюме:

- **Минимум:** 2+ CPU, 4-8 GB RAM, 30+ GB SSD, без GPU. Обучение будет долгим.
- **Рекомендуется:** 4+ CPU, 16+ GB RAM, 100+ GB NVMe SSD, **NVIDIA GPU с 8GB+ VRAM** (чем больше, тем лучше) + CUDA/cuDNN.

Не забудьте также настроить ротацию логов и мониторинг ресурсов на продакшен-сервере.

3. Развертывание программного обеспечения на сервере Заказчика:

Развертывание на продакшен-сервере Ubuntu требует внимательного подхода к настройке окружения, зависимостей, безопасности и обеспечению надежной работы ваших скриптов (`main.py` и `api_server.py`).

Вот подробная пошаговая инструкция:

Предпосылки:

- У вас есть доступ к облачному серверу Ubuntu (например, через SSH). Рекомендуется использовать LTS-версию Ubuntu (например, 20.04 или 22.04).
- У вас есть права `sudo` на сервере.
- Ваш код проекта находится в системе контроля версий (например, Git на GitHub/GitLab). Это *крайне* рекомендуется для удобства развертывания и обновлений.

Шаг 1: Подготовка сервера Ubuntu

1. **Подключитесь к серверу по SSH.**
2. **Обновите списки пакетов и сами пакеты:**

```
sudo apt update
sudo apt upgrade -y
```

3. **Установите необходимые системные пакеты:**
   - `python3`: Обычно уже установлен. Проверьте: `python3 --version`.
   - `python3-pip`: Менеджер пакетов Python.
   - `Sudo apt install 3.12-venv` : Для создания виртуальных окружений (лучшая практика).
   - `git`: Для клонирования вашего репозитория.
   - `Sudo apt install nginx`: Высокопроизводительный веб-сервер, который будет работать как реверс-прокси для вашего Flask API.

```
sudo apt install -y python3-pip python3-venv git nginx
```

Шаг 2: Развертывание кода проекта

1. **Выберите директорию для проекта:** Рекомендуется размещать веб-приложения вне домашних директорий пользователей. Популярные варианты: `/srv/` или `/opt/`. **Создадим папку приложения** `/krammerti/lstm`:

```
sudo mkdir -p /home/krammerti/lstm
# Смените владельца на вашего пользователя (чтобы не работать постоянно с sudo)
# Замените <your_user> на ваше имя пользователя на сервере
sudo chown -R <krammerti>:< krammerti >
cd /home/krammerti/lstm
```

2. **Клонируйте репозиторий если Вы хотите оперировать версионностью из Git:**
3.
```
# Замените <your_repository_url> на URL вашего Git-репозитория
git clone <your_repository_url> .
# Точка в конце команды клонирует содержимое репозитория прямо в /krammerti/lstm
```

   Если вы не используете Git, скопируйте все файлы проекта (`.py`, `config.ini`, `hyperparameters.json`, папки `models`, `weather_data`, `logs` и т.д.) на сервер в эту директорию (например, с помощью `scp` или `rsync`).

Шаг 3: Настройка окружения Python и зависимостей

1. **Создайте виртуальное окружение:**

```
# Находясь в /krammerti/lstm
python3 -m venv venv
```

2. **Активируйте виртуальное окружение:**

```
source venv/bin/activate
# Ваша командная строка должна измениться, показав (venv) в начале
```

Важно: Все последующие команды `pip` и `python` выполняйте в *активированном* окружении.

3. **Создайте `requirements.txt` (если его нет):**
   - На вашей **локальной машине** (где Вы локально работаете с проектом) активируйте то же самое окружение и выполните:

     ```
     pip freeze > requirements.txt
     ```

   - Добавьте `requirements.txt` в ваш Git-репозиторий и обновите его на сервере (`git pull`) или скопируйте файл на сервер.
   - **Убедитесь, что в `requirements.txt` есть все зависимости:** `tensorflow`, `pandas`, `scikit-learn`, `optuna`, `schedule`, `requests`, `requests-cache`, `tenacity`, `aiohttp`, `matplotlib`, `Flask`, `gunicorn` (мы установим его сейчас), `python-dateutil`.
   - Возможно, понадобятся и другие, проверьте импорты во всех `.py` файлах.

4. **Установите зависимости на сервере:**

```
# Находясь в /krammerti/lstm с активированным venv
pip install --upgrade pip # Обновить pip
pip install -r requirements.txt
# Установим Gunicorn отдельно (WSGI сервер для Flask)
pip install gunicorn
```

Шаг 4: Конфигурация приложения (`config.ini`)

1. **Отредактируйте `config.ini` на сервере:**

```
nano config.ini # или используйте другой редактор
```

2. **Проверьте и исправьте пути:** Все пути в секции `[PATHS]` должны быть **абсолютными и корректными** для сервера Ubuntu. Например:

```
[PATHS]
DATA_DIR = /krammerti/lstm/weather_data
MODELS_DIR = /krammerti/lstm/models
HYPERPARAMETERS_FILE = /krammerti/lstm/hyperparameters.json
```

```
GRAPH_DIR = /krammerti/lstm/graphs
OPEN_METEO_FORECAST_FILE =
/krammerti/lstm/weather_data/open_meteo_forecasts.csv
FORECASTS_DIR = /krammerti/lstm/lstm_forecasts
LOGS_DIR = /krammerti/lstm/logs
```

3. **Проверьте другие настройки:** Убедитесь, что токены, ID станций, параметры HPO (`N_TRIALS`), расписание (`TRAINING_TIME` и т.д.) установлены так, как нужно для продакшен сервера.
4. **Кодировка:** Убедитесь, что файлы сохранены в кодировке **UTF-8**.

Шаг 5: Настройка фонового запуска `main.py` (Планировщик)

Чтобы `main.py` (с его `schedule`) работал постоянно в фоне и перезапускался автоматически, используем `systemd`.

1. **Создайте файл сервиса `systemd`:**

   ```
   sudo nano /etc/systemd/system/lstm-scheduler.service
   ```
2. **Вставьте следующее содержимое** (замените `<your_user>` на вашего пользователя):

   ```
   [Unit]
   Description=Forecast Scheduler Service (main.py)
   After=network.target # Запускаться после поднятия сети

   [Service]
   User=<your_user> # В нашем случае: krammerti указывается без угольных
   кавычек!
   Group=<your_user> # Обычно совпадает с пользователем, указываем без
   угольных кавычек!
   WorkingDirectory=/krammerti/lstm/# Путь к вашему проекту
   # Путь к python внутри виртуального окружения
   ExecStart=/krammerti/lstm/venv/bin/python /krammerti/lstm/main.py
   Restart=always # Всегда перезапускать сервис при сбое
   RestartSec=10  # Пауза перед перезапуском
   StandardOutput=journal # Направить stdout в journald
   StandardError=journal  # Направить stderr в journald
   SyslogIdentifier=forecast-scheduler

   [Install]
   WantedBy=multi-user.target # Запускать при многопользовательском режиме
   ```

3. **Сохраните и закройте файл** (в `nano`: `Ctrl+X`, затем `Y`, затем `Enter`).

4. **Перезагрузите конфигурацию `systemd`:**

   ```
   sudo systemctl daemon-reload
   ```

5. **Включите автозапуск сервиса:**

   ```
   sudo systemctl enable lstm-scheduler.service
   ```

6. **Запустите сервис:**

   ```
   sudo systemctl start lstm-scheduler.service
   ```

7. **Проверьте статус сервиса:**

   ```
   sudo systemctl status lstm-scheduler.service
   # Вы должны увидеть "active (running)"
   # Нажмите 'q' для выхода
   ```

8. **Проверьте логи сервиса:**

   ```
   sudo journalctl -u lstm-scheduler.service -f
   # -f будет показывать новые логи в реальном времени
   # Ctrl+C для выхода
   # Посмотрите, нет ли ошибок при запуске main.py
   ```

   Логи из самого `main.py` (которые пишутся в `app.log`) также будут создаваться в `/krammerti/lstm/logs/app.log`.

Шаг 6: Настройка фонового запуска `api_server.py` (Flask API)

Аналогично `main.py`, API сервер должен работать в фоне и управляться `systemd`, но через WSGI сервер (Gunicorn).

1. **Создайте файл сервиса `systemd` для Gunicorn:**

   ```
   sudo nano /etc/systemd/system/lstm-api.service
   ```

2. **Вставьте следующее содержимое** (замените `<your_user>`, подберите количество `workers`):

   ```
   [Unit]
   Description=Forecast API Service (Gunicorn serving api_server:app)
   After=network.target

   [Service]
   User=<your_user>
   ```

```
Group=<your_user>
WorkingDirectory=/krammerti/lstm
# Указываем путь к gunicorn внутри venv
# -w 4: Количество рабочих процессов (обычно 2 * CPU cores + 1)
# -b 127.0.0.1:8000: Gunicorn будет слушать ТОЛЬКО локально на
порту 8000
# api_server:app: Запустить объект 'app' из файла 'api_server.py'

ExecStart=/krammerti/lstm/venv/bin/gunicorn --workers 4 --bind
127.0.0.1:8000 api_server:app
Restart=always
RestartSec=10
StandardOutput=journal
StandardError=journal
SyslogIdentifier=forecast-api

[Install]
WantedBy=multi-user.target
```

- **`-b 127.0.0.1:8000`**: Важно, чтобы Gunicorn слушал *только* локально. Nginx будет принимать внешние запросы и перенаправлять их на этот локальный порт. Вы можете выбрать другой порт вместо 8000, если он занят.

3. **Сохраните и закройте файл.**
4. **Перезагрузите `systemd`, включите и запустите сервис:**

```
sudo systemctl daemon-reload
sudo systemctl enable lstm-api.service
sudo systemctl start lstm-api.service
```

5. **Проверьте статус и логи:**

```
sudo systemctl status lstm-api.service
sudo journalctl -u lstm-api.service -f
```

Шаг 7: Настройка Nginx как реверс-прокси

Nginx будет принимать запросы на порт `3000` и перенаправлять их на Gunicorn, работающий на порту `8000`.

1. **Создайте файл конфигурации Nginx для вашего API:**

```
sudo nano /etc/nginx/sites-available/lstm_api
```

2. **Вставьте следующее содержимое:**

```
server {
    listen 3000; # Публичный порт, на котором Nginx слушает
```

```
    # Замените!! 192.168.0.0 на реальный IP сервера или доменное имя,
если есть
    server_name 192.168.0.0;

    access_log /var/log/nginx/forecast_api_access.log;
    error_log /var/log/nginx/forecast_api_error.log;

    location / {
        # Перенаправляем запросы на Gunicorn, работающий на порту 8000
        proxy_pass http://127.0.0.1:8000;

        # Важные заголовки для Flask/Gunicorn
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

3. **Сохраните и закройте файл.**
4. **Создайте символическую ссылку, чтобы активировать конфигурацию:**

   ```
   sudo ln -s /etc/nginx/sites-available/lstm_api /etc/nginx/sites-
   enabled/
   ```
5. **Проверьте конфигурацию Nginx на ошибки:**

   ```
   sudo nginx -t
   # Должно вывести "syntax is ok" и "test is successful"
   ```
6. **Перезапустите Nginx, чтобы применить изменения:**

   ```
   sudo systemctl restart nginx
   ```

Шаг 8: Настройка брандмауэра (`ufw`)

Нужно разрешить входящие подключения к Nginx на порту 3000 и SSH.

1. **Разрешите Nginx (он сам знает нужные порты, включая 3000 из конфига):**

   ```
   sudo ufw allow 'Nginx Full'
   # Или явно указать порт:
   # sudo ufw allow 3000/tcp
   ```
2. **Убедитесь, что SSH разрешен (ОЧЕНЬ ВАЖНО!):**

   ```
   sudo ufw allow OpenSSH
   ```

```
# Или явно порт 22:
# sudo ufw allow 22/tcp
```

3. **Включите брандмауэр (если еще не включен):**

```
sudo ufw enable
# Подтвердите действие, введя 'y'
```

4. **Проверьте статус:**

```
sudo ufw status
```

Вы должны увидеть `ALLOW` для порта 3000 (или Nginx) и 22 (или OpenSSH).

Шаг 9: Тестирование

- **Проверьте API:** Откройте браузер или используйте `curl` с *другой машины* в вашей сети (или из интернета, если сервер доступен) и обратитесь по адресу `http://192.168.0.0:3000/api/v1/forecast/temperature`. Вы должны получить JSON с последним прогнозом. **Важно! Используйте IP адрес Вашего продакшен сервера и порт который Вы используете.**
- **Проверьте планировщик:** Наблюдайте за логами (`journalctl -u lstm-scheduler.service -f` и `logs/app.log`) в моменты, указанные в расписании (`TRAINING_TIME`, `HISTORY_UPDATE_TIME` и т.д.), чтобы убедиться, что задачи запускаются. Проверяйте создание новых файлов прогнозов и моделей.

Шаг 10: Мониторинг и обслуживание (Рекомендации)

- **Логи:** Регулярно проверяйте логи `systemd` (`journalctl`) и приложения (`logs/app.log`) на наличие ошибок. Настройте ротацию логов, если нужно (для `app.log` у вас уже есть `RotatingFileHandler`).
- **Ресурсы:** Следите за использованием CPU, RAM, дискового пространства на сервере. Обучение LSTM может быть ресурсоемким.
- **Обновления:** Периодически обновляйте системные пакеты и зависимости Python.
- **Бэкапы:** Настройте регулярное резервное копирование кода, данных (`weather_data`, `lstm_forecasts`), моделей (`models`) и базы данных Optuna (`models/*.db`).

Остановка и перезапуск сервисов:

Останавливать процессы, запущенные через `systemd`, нужно с помощью команд `systemctl`.

Как остановить сервисы:

1. **Остановить сервис планировщика (`main.py`):**

   ```
   sudo systemctl stop lstm-scheduler.service
   ```

   Это действие пошлет сигнал завершения процессу Python, который выполняет `main.py` (и цикл `schedule`). Он должен завершиться штатно (может занять несколько секунд).

2. **Остановить сервис API (`api_server.py` через Gunicorn):**

   ```
   sudo systemctl stop lstm-api.service # Используйте ваше имя сервиса (lstm-api.service?)
   ```

   Это действие пошлет сигнал завершения главному процессу Gunicorn, который, в свою очередь, остановит свои рабочие процессы.

Как проверить, что они остановлены:

Используйте `systemctl status`:

```
sudo systemctl status lstm-scheduler.service
sudo systemctl status lstm-api.service
```

В статусе `Active`: вы должны увидеть что-то вроде `inactive (dead)`.

Как запустить их снова:

```
sudo systemctl start lstm-scheduler.service
sudo systemctl start lstm-api.service
```

Как перезапустить (остановить и запустить одной командой):

```
sudo systemctl restart lstm-scheduler.service
sudo systemctl restart lstm-api.service
```

Как отключить автозапуск при старте системы:

Если вы хотите, чтобы сервисы не запускались автоматически при перезагрузке сервера:

```
sudo systemctl disable lstm-scheduler.service
sudo systemctl disable lstm-api.service
```

При этом сами сервисы не останавливаются, если они запущены. Чтобы остановить и отключить автозапуск:

```
sudo systemctl stop lstm-scheduler.service
sudo systemctl disable lstm-scheduler.service

sudo systemctl stop lstm-api.service
sudo systemctl disable lstm-api.service
```

Важно: Управляйте сервисами через `systemctl`. Не пытайтесь «убивать» процессы (`kill`) вручную, если только сервис не завис и не реагирует на `systemctl stop`.

Данная подробная инструкция для Администратора.

Следуйте шагам внимательно, адаптируя пути и имена пользователей под вашу конкретную среду. Инструкция проверена и является рабочей.

Благодарим Вас за использование нашего программного продукта,

С Уважением ООО «Краммерти»