

Построение модели прогнозирования состояния трехфазного сепаратора (пробкоуловителя) для оптимизации выхода полезного продукта презентации



Хоменко Алексей

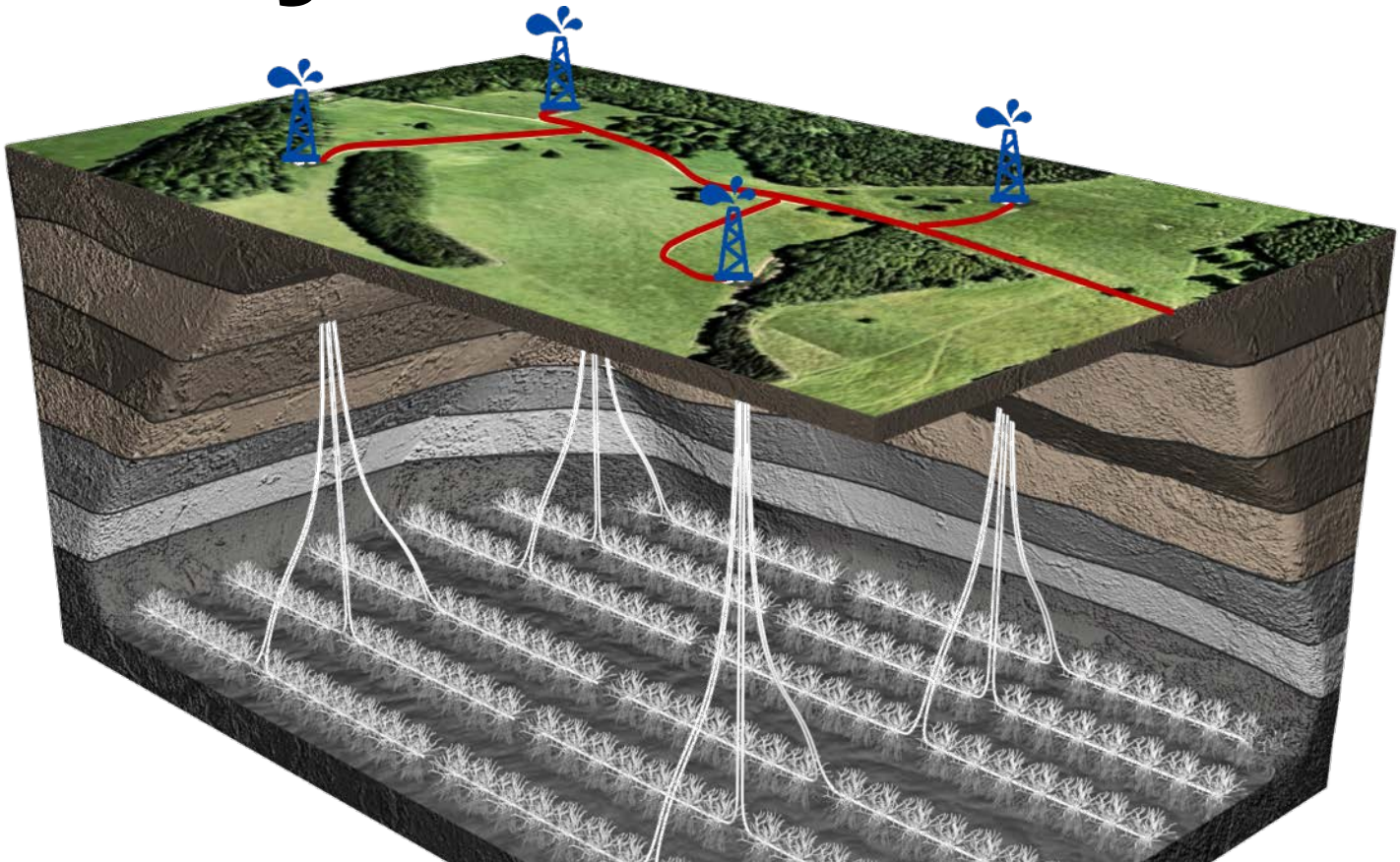


План защиты

- 1 Актуальность
- 2 Формализованная форма задачи и этапы ее решения
- 3 Математическая модель предсказания. Задача и метрики
- 4 Анализ исходных данных
- 5 Разделение на обучающую и тестовую выборку
- 6 Итоговая модель
- 7 Возможный вариант внедрения. Выводы. Пути развития



Актуальность

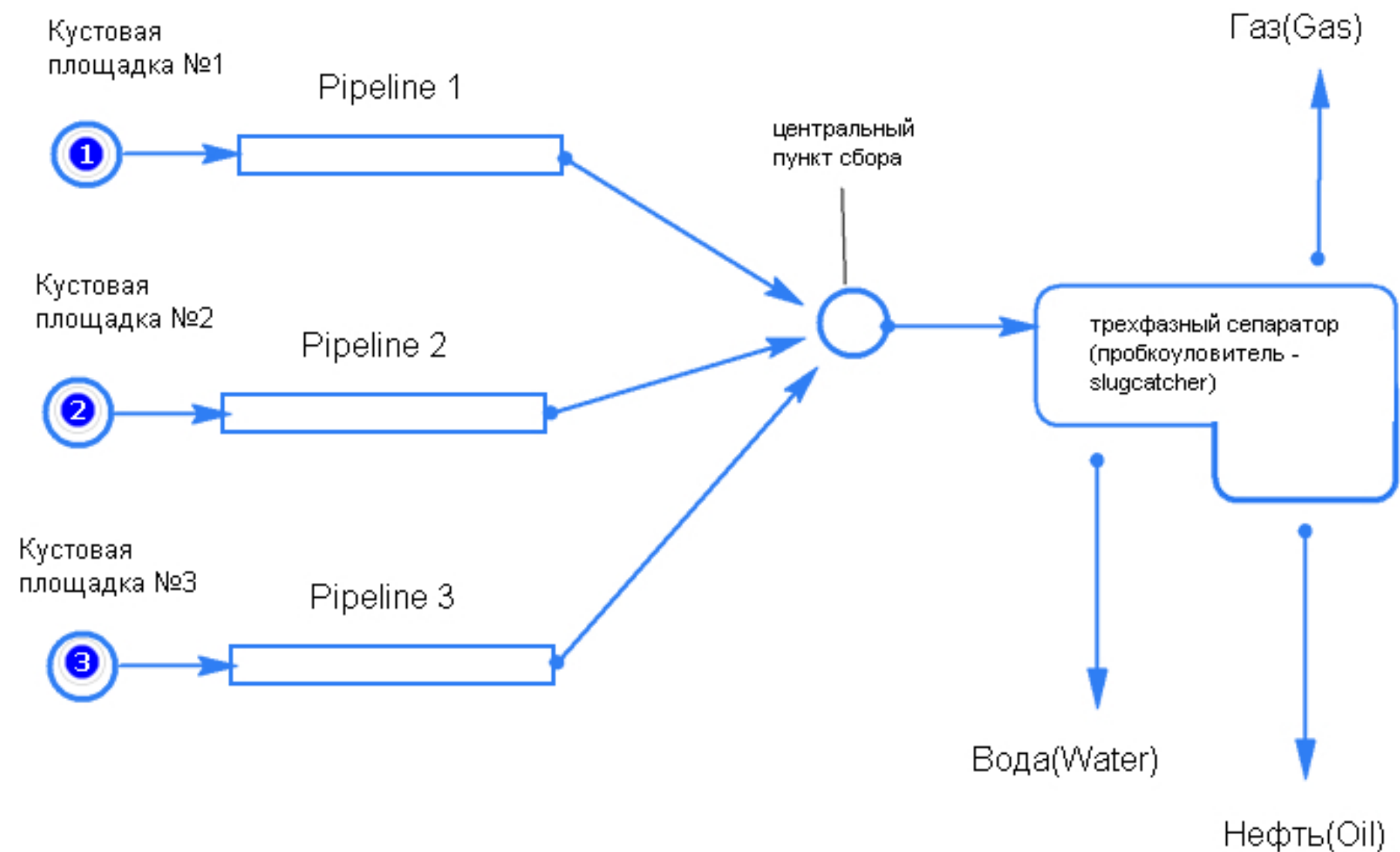


Формализованная форма задачи и этапы ее решения

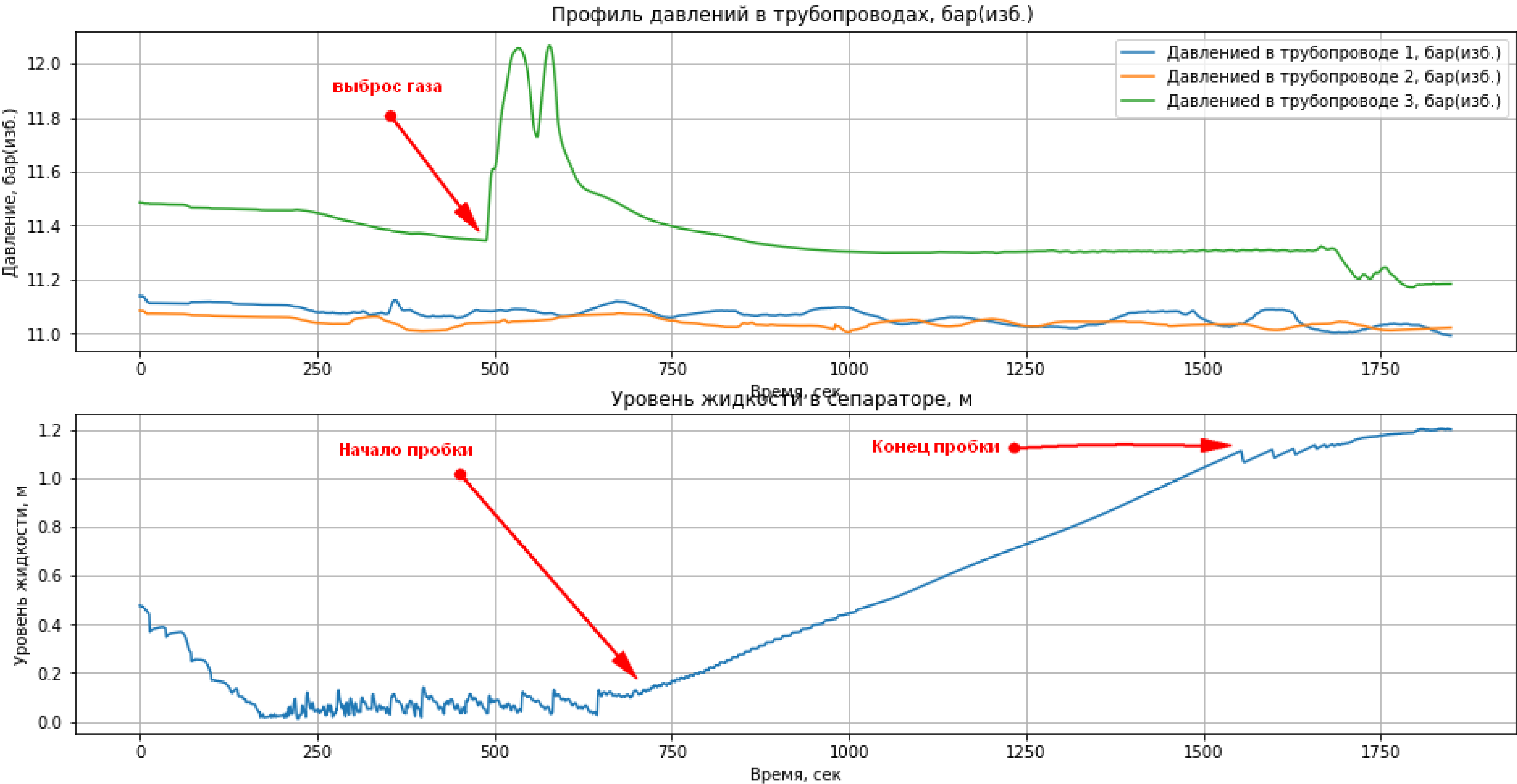
Формализованная задача: Составить математическую модель предсказания, в зависимости от изменяющихся входных данных – признаков.

За **исходные данные** принята модель сбора условного месторождения (рисунок 1). Условное месторождение состоит из 3 кустовых площадок, центральным пунктом сбора, из центрального пункта сбора скважинный продукт поступает на подготовку: очистку, переработку, транспортировку (на схеме не показано).

Рисунок 1. Схема системы сбора условного месторождения



Анализ исходных данных



Математическая модель предсказания. Задача и метрики

Формулы каждой из метрик:

MSE:

$$\Delta x = y_{real_liquid_level} - y_{predicted}$$

$$L = \Delta x^2$$

Функция потерь Хьюбера (при delta =1 является SmoothL1 Loss):

$$\Delta x = y_{real_liquid_level} - y_{predicted}$$

$$L = \begin{cases} 0.5 \cdot \Delta x^2, & \text{if } |\Delta x| < \delta \\ \delta \cdot (|\Delta x| - 0.5 \cdot \delta), & \text{otherwise} \end{cases}$$



Анализ исходных данных

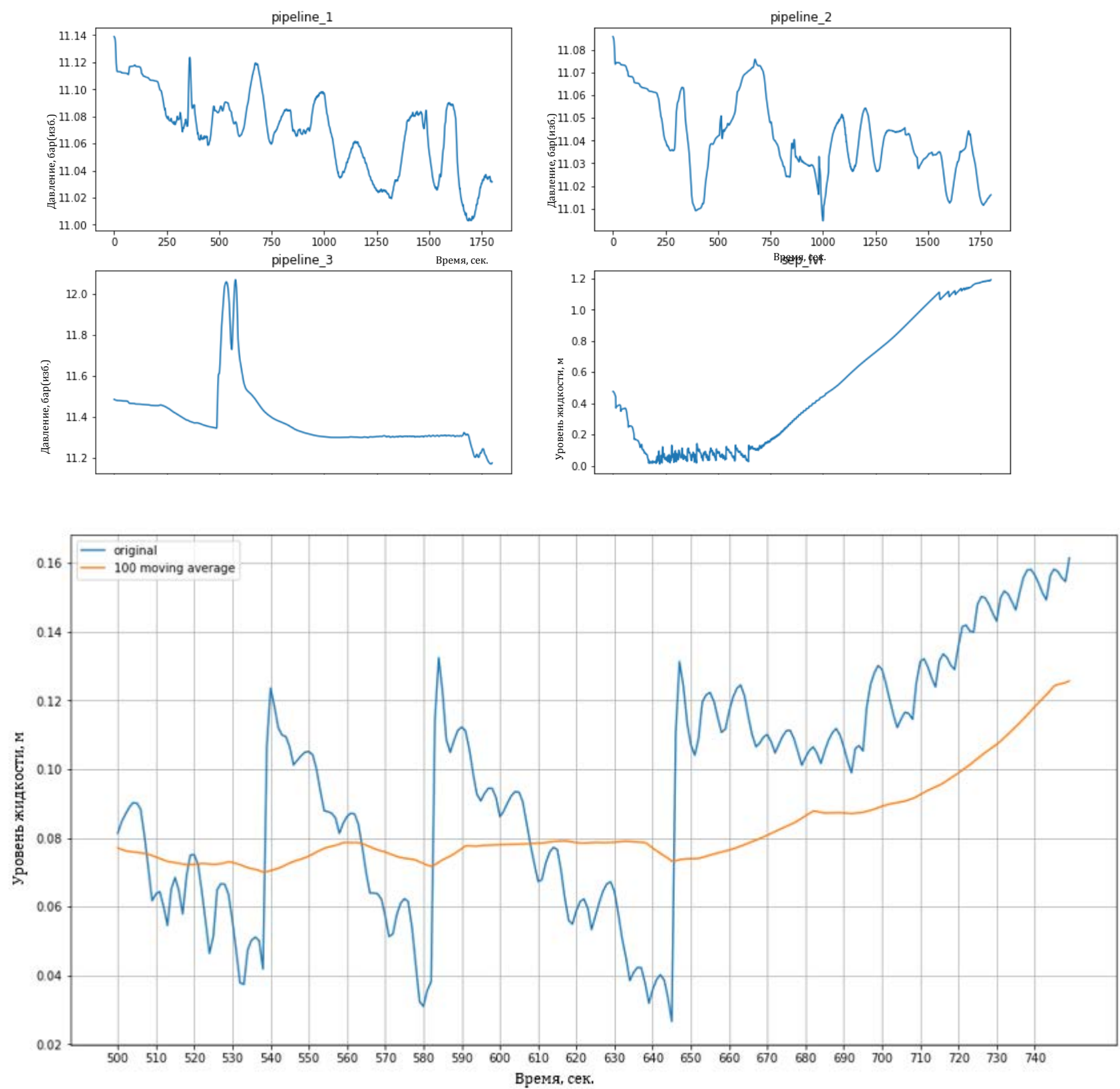
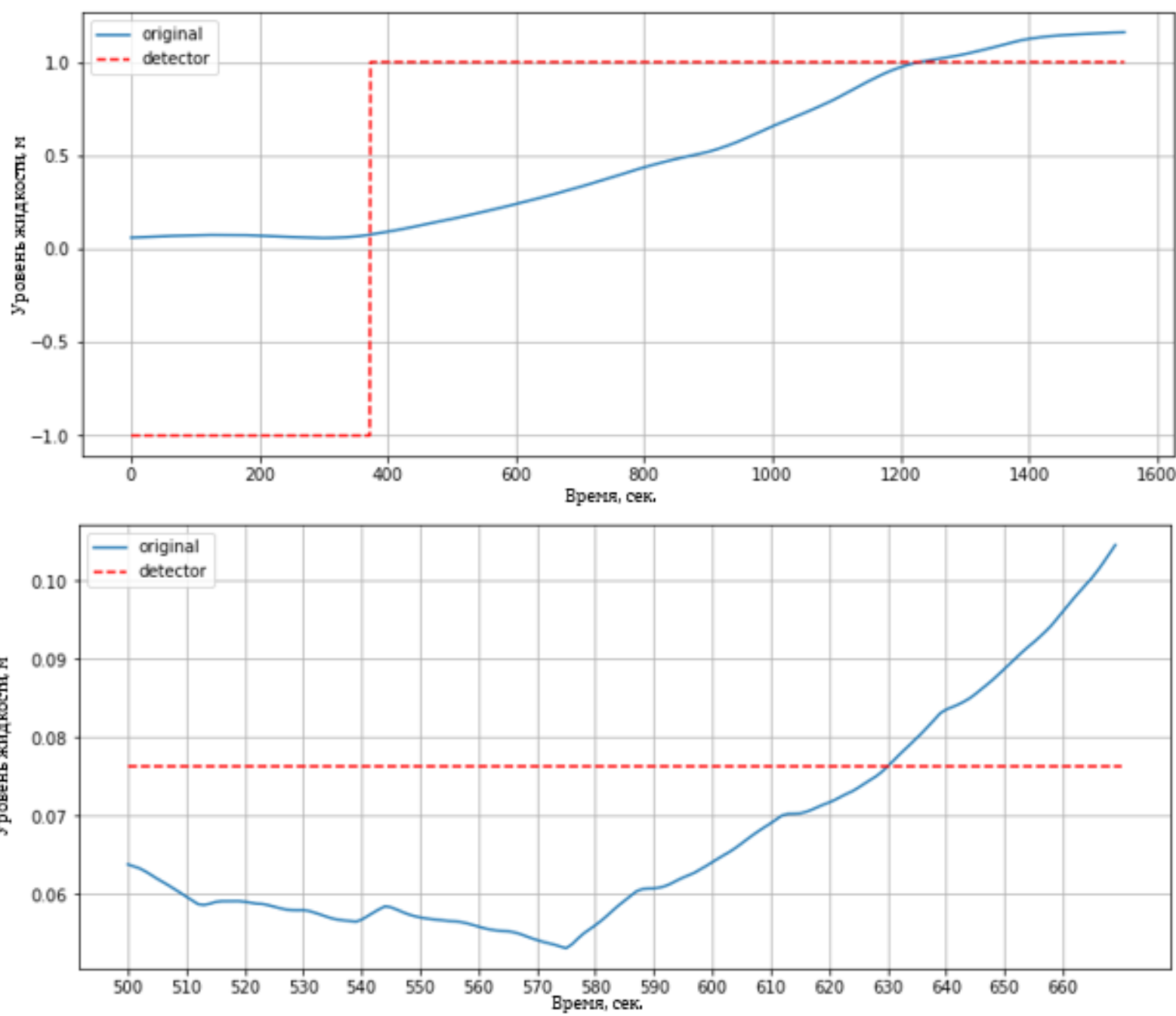


Рисунок 6.1. SVM One class detector. Тренд роста объема сепаратора.



Итоговая модель

Код приближен к sklearn:

```
'''Python
net = AttentionModel()
net.fit(X,y)
net.predict(X)
'''
```

```
SEP_LEVELS = int(2.65 * 1000)

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        self.linear = torch.nn.Linear(300, 300, bias = True, dtype=torch.float32)
        self_emb_lvl = torch.nn.Embedding(SEP_LEVELS, 300)
        #self.linear_attention = torch.nn.Linear(300, 300, dtype=torch.float32)

        self.lstm = torch.nn.LSTM(input_size=300, hidden_size=128, batch_first = True, bidirectional=True) #direction...
        self.linear_1 = torch.nn.Linear(in_features=128, out_features = 1, bias = True)

    def forward(self, x, previous_y):
        # pass pipe pressure lags through Dense 200 -> 300
        # self.linear = torch.nn.Linear(300, 300, bias = True, dtype=torch.float32)
        # get previous separator level embedding
        previous_separator_level = round(previous_y, 2) * 1000 # previous_y = y[0]
        previous_separator_level = int(previous_separator_level)
        previous_separator_level_emb = self_emb_lvl(previous_separator_level)
        previous_separator_level_emb = self_emb_lvl(previous_separator_level) # Dense may pass up emb = 300
        previous_separator_level_emb = previous_separator_level_emb.reshape(1, 300)

        # get attention
        attention = self.linear * previous_separator_level_emb
        attention = self.linear_attention(attention)
        attention_softmax = torch.nn.functional.softmax(attention, dim=0)

        # multiply pressure lags with attention softmax
        final_state = x * attention_softmax
        final_state = final_state.reshape(1, 300)

        # LSTM
        embeddings, (shortterm, longterm) = self.lstm(final_state)
        longterm = torch.add(longterm[0], longterm[1]) / 2
        predict = self.linear_1(longterm)

        return predict

    def fit(self, X_features, y_target):
        EPOCHS = 10
        criterion = torch.nn.MSELoss()
        OPT_EPOCH_SIZE = len(y_target) // 5
        min_loss = 0.0
        train_loader = torch.utils.data.DataLoader(X_features, y_target, shuffle=True)
        len = 0.001

        for epoch in range(EPOCHS):
            prediction_array = torch.tensor([]).float()

            for i in tqdm(range(1, len(y_target) // 5 + 1)):
                y_pred_array = torch.tensor([])
                for j in range(10):
                    x = np.array(X_features.iloc[(i-1)*10:(i*10)+(y_target*1000)])
                    x = torch.from_numpy(x).reshape(10, 300).float()

                    previous_y = y_target[i]

                    y_pred = self.forward(x, previous_y)
                    y_pred_array = torch.cat([y_pred_array, y_pred])
                    y_pred = torch.mean(y_pred_array, reshape=1)

                prediction_array = torch.cat([prediction_array, y_pred], dim=0)

            loss = criterion(prediction_array.reshape(-1), train_loader)

            if loss < min_loss:
                min_loss = loss
                torch.save(net, "/model_loss_{}.pt".format(min_loss))
                len = 0.001

            loss.backward()

            optimizer = torch.optim.Adam(net.parameters(), lr = 1e-4)

            optimizer.step() # minimize loss again
            optimizer.zero_grad() # minimize loss

            # at the end of each epoch print the loss
            if epoch % 1 == 0: print(f'epoch {epoch}: MSELoss = {loss}')

        torch.save(net)

    def predict(self, X_features):
        prediction_array = np.array([], dtype=np.float32)

        #with torch.no_grad():
        for i in tqdm(range(1, len(X_features) // 5 + 1)):
            y_pred_array = torch.tensor([])
            for j in range(10):
                x = np.array(X_features.iloc[(i-1)*10:(i*10)+(y_target*1000)])
                x = torch.from_numpy(x).reshape(10, 300).float()

                previous_y = prediction_array[-1]

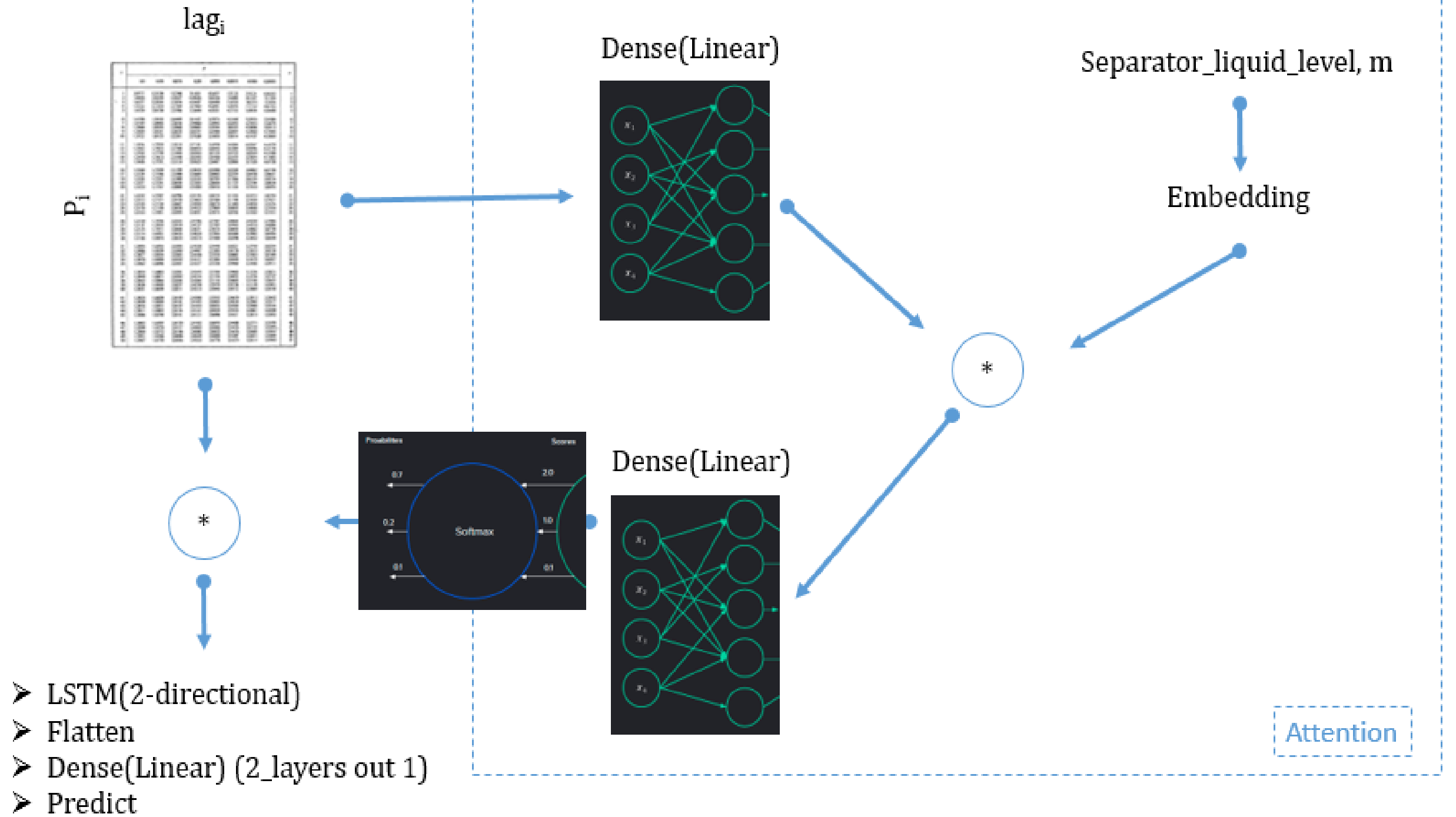
                y_pred = self.forward(x, previous_y)
                y_pred_array = torch.cat([y_pred_array, y_pred])
                y_pred = abs(torch.mean(y_pred_array))

            prediction_array = np.append(prediction_array, y_pred.numpy())

        return prediction_array[1:]

net = Net()
print(net)
<
>
Included in 99ms, Memory 23.80 MB, 2022-07-02

Net:
  (linear): Linear(in_features=300, out_features=300, bias=True)
  (emb_lvl): Embedding(1000, 300)
  (lstm): LSTM(300, 128, batch_first=True, bidirectional=True)
  (linear_1): Linear(in_features=128, out_features=1, bias=True)
```



Сравнение результатов

Figure 1. MSE and HuberLoss Results
The best Algorithm in LOWER LEFT corner

