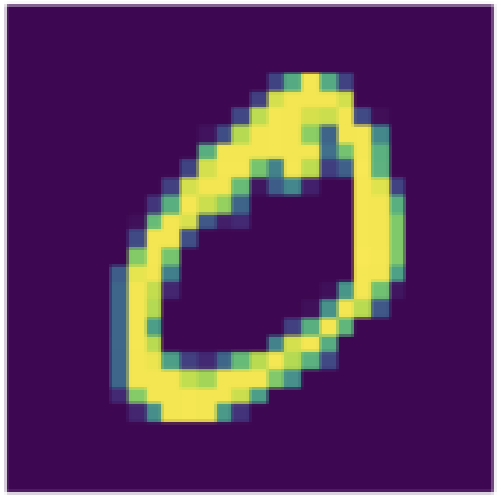
Robust classifier

Подготовил: Ореховский Илья

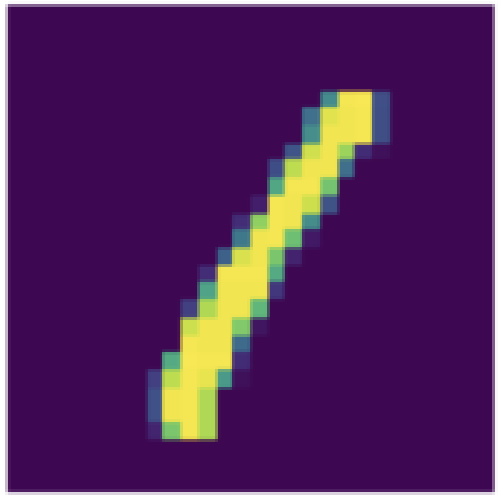
С чем работаем?

Датасет MNIST

Используемые классы



-1



1

Что нужно сделать?

1 шаг: тренировка обычной модели

- Разбить данные на train и test
- Нормализовать датасет
- Натренировать модель логистической регрессии
- Оценить точность

Реализация 1 шага

```
# Импортируем необходимые библиотеки
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler

# Разделим датасет на тренировочный и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X_bin, y_bin,
test_size=0.2, random_state=42)

# Нормализуем данные
X_train /= 255
X_test /= 255

# Тренируем модель логистической регрессии
model = LogisticRegression(max_iter=1000, solver='lbfgs')
model.fit(X_train, y_train)

# Оценим точность модели на тестовом наборе
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Точность модели на чистом тестовом наборе: {accuracy:.4f}")
```

Значения пикселей [0:255]
→[0:1]

```
Точность модели на чистом тестовом наборе: 0.9997
```

2 шаг: проведение атаки на модель

- Вычислить значения оптимальной атаки $\delta_\star$ для каждого класса (-1 и 1)
- Применить атаку к каждому изображению в test set
- Оценить точность

Реализация 2 шага

```
# Определим максимальное изменение каждого пикселя
epsilon = 0.2

# Получим весовые коэффициенты модели
weights = model.coef_
sign_weights = np.sign(weights)

# Вычислим оптимальную атаку
def optimal_attack(X, y, weights, epsilon):
    delta = -y[:, np.newaxis] * epsilon * np.sign(weights)
    return X + delta

# Атака на тестовый набор
X_test_attacked = optimal_attack(X_test, y_test, weights, epsilon)

# Точность на атакованном тестовом наборе
y_pred_attacked = model.predict(X_test_attacked)
accuracy_attacked = accuracy_score(y_test, y_pred_attacked)
print(f"Точность на атакованном тестовом наборе:
{accuracy_attacked:.4f}")
```

$x' = x + \delta, ||\delta||_\infty \leq 0.2 = \epsilon$

$\delta^\star = -y \cdot \epsilon \cdot \text{sign}(w)$

Точность на атакованном тестовом наборе: 0.0873

3 шаг: Тренировка Robust Classifier

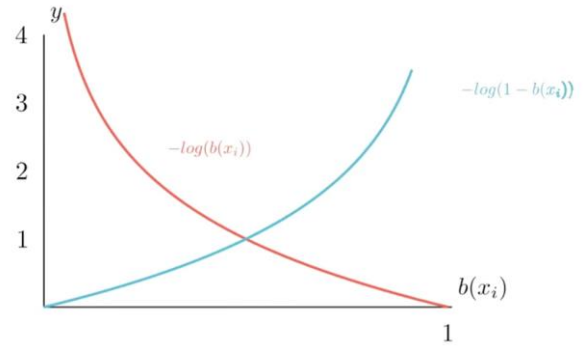
- Реализовать свой ML-метод
- Натренировать модель
- Вычислить оптимальную атаку
- Оценить точность на атакованном наборе данных

Реализация 3 шага

```python
class RobustLogisticRegression(BaseEstimator):
    def __init__(self, learning_rate=0.01, max_iter=1000, epsilon=0.2):
        self.learning_rate = learning_rate
        self.max_iter = max_iter
        self.epsilon = epsilon
        self.weights = None
        self.bias = None

    def _sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def _loss(self, X, y):
        z = X @ self.weights + self.bias
        margin = y * z - self.epsilon * np.sum(np.abs(self.weights))
        return np.mean(np.log(1 + np.exp(-margin)))

    def fit(self, X, y):
        # Добавим смещение к признакам
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.max_iter):
            z = X @ self.weights + self.bias
            predictions = self._sigmoid(z)

            # Градиент обновлений
            gradient_w = -(y * (1 - predictions)) @ X / n_samples
            gradient_b = -np.mean(y * (1 - predictions))

            # Учитываем робастность
            gradient_w += self.epsilon * np.sign(self.weights)

            # Обновление весов
            self.weights -= self.learning_rate * gradient_w
            self.bias -= self.learning_rate * gradient_b

    def predict(self, X):
        return np.sign(X @ self.weights + self.bias)
```

$$\text{sigmoid}(z) = \frac{1}{1 + e^{-z}}$$

$$z = X \cdot w + b$$



$$\text{loss} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)]$$

$$\nabla_w \text{loss} = -\frac{1}{N}\sum_{i=1}^{N}(y_i - \sigma(z_i)) \cdot X_i + \epsilon \cdot \text{sign}(w)$$

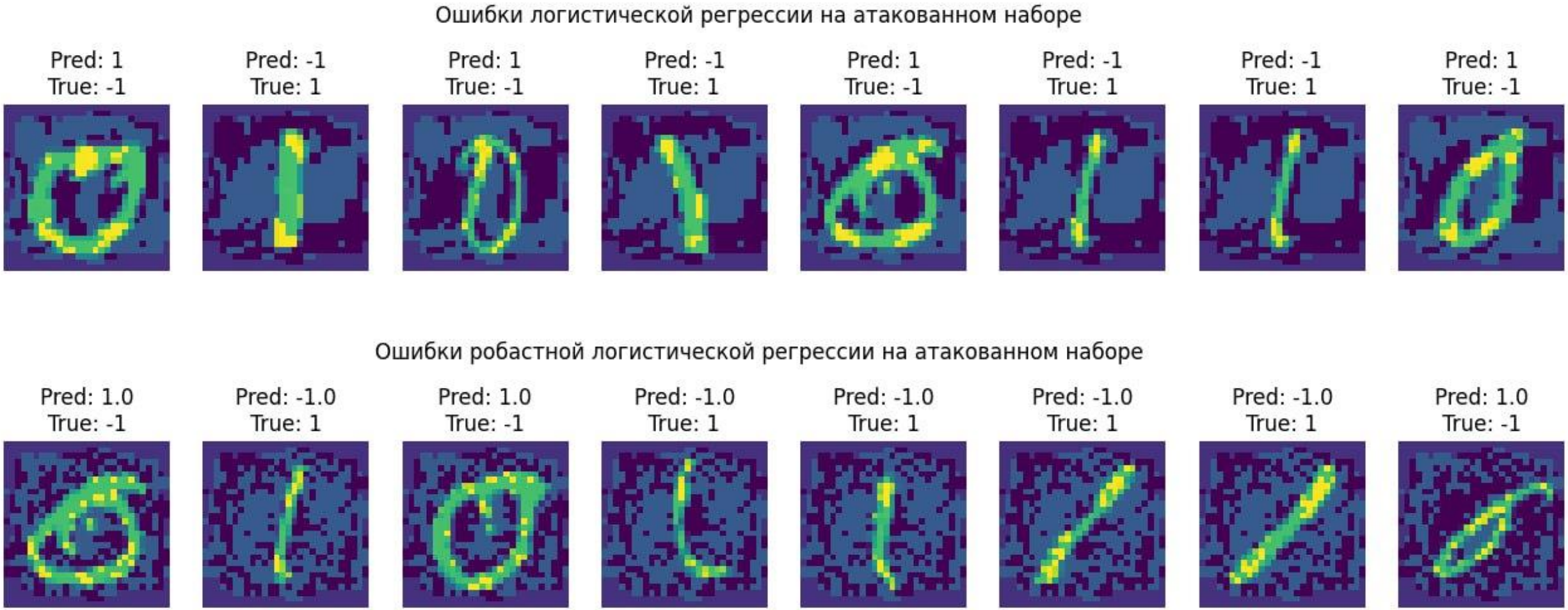
$$\nabla_b \text{loss} = -\frac{1}{N}\sum_{i=1}^{N}(y_i - \sigma(z_i))$$

Реализация 3 шага

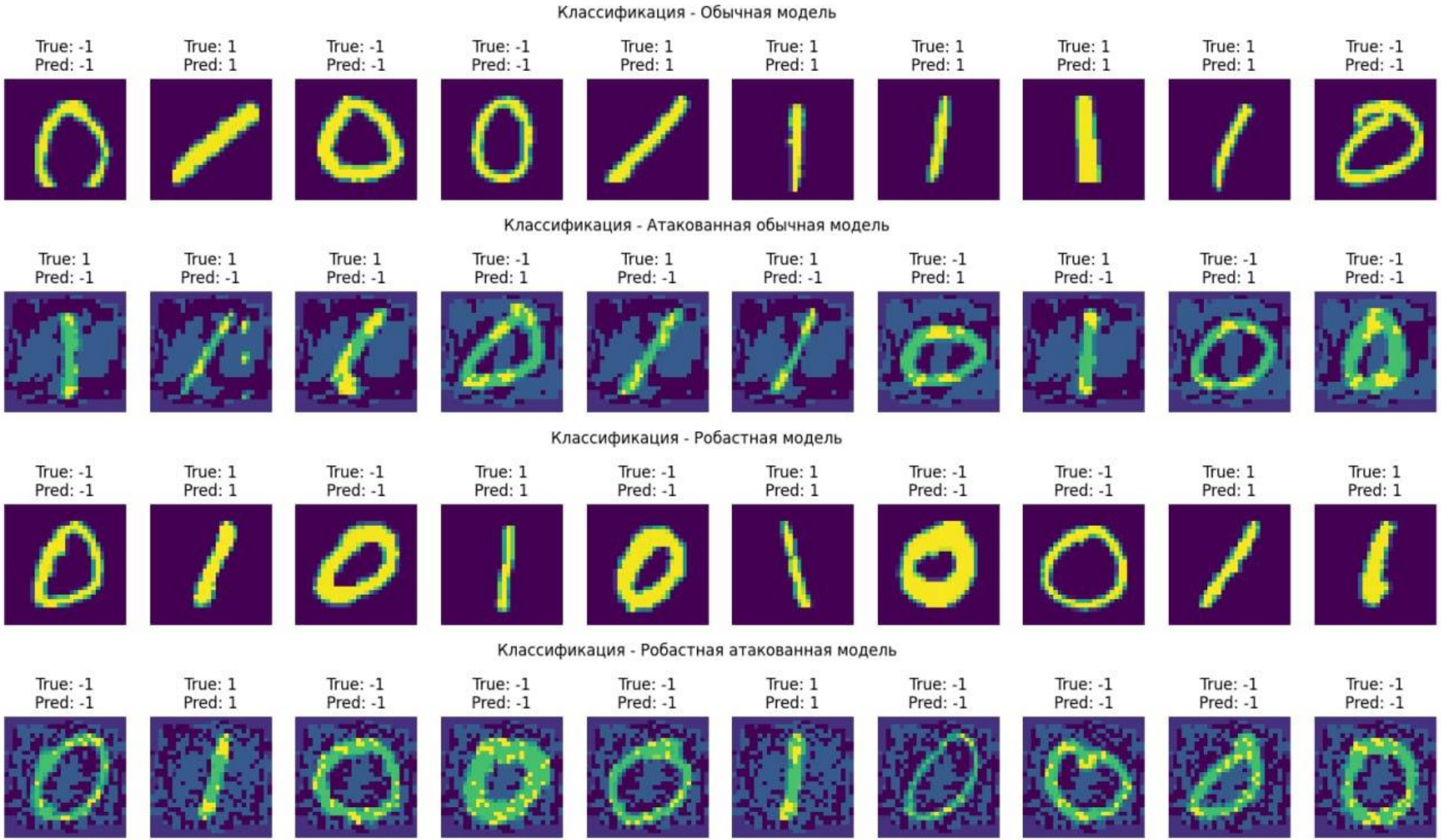
```
# Тренируем модель робастной логистической регрессии
robust_model = RobustLogisticRegression(learning_rate=0.01, max_iter=1000,
epsilon=0.2)
robust_model.fit(X_train, y_train)

# Оценим точность робастной модели на чистом тестовом наборе
y_pred_robust = robust_model.predict(X_test)
accuracy_robust = accuracy_score(y_test, y_pred_robust)
print(f"Точность робастной модели на чистом тестовом наборе:
{accuracy_robust:.4f}")

# Применим оптимальную атаку на робастную модель
weights_robust = robust_model.weights
X_test_attacked_robust = optimal_attack(X_test, y_test, weights_robust,
epsilon)

# Оценим точность робастной модели на атакованном тестовом наборе
y_pred_attacked_robust = robust_model.predict(X_test_attacked_robust)
accuracy_attacked_robust = accuracy_score(y_test, y_pred_attacked_robust)
print(f"Точность робастной модели на атакованном тестовом наборе:
{accuracy_attacked_robust:.4f}")
```

- Точность робастной модели на чистом тестовом наборе: 0.9915
- Точность робастной модели на атакованном тестовом наборе: 0.9100

Визуализация результатов



Визуализация результатов



Визуализация результатов