НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет ПИиКТ

Системы искусственного интеллекта

Лабораторная работа № 2

Выполнил студент

Набокова Алиса Владиславовна

Группа № P3320

Преподаватель: Королёва Юлия Александровна

г. Санкт-Петербург

2025

Задание:

Добавить в обучение моделей из ЛР1 регуляризацию (L1, L2, ElasticNet), вывести метрики качества обучения

Реализовать класс модели KNN для линейной и логистической регрессии. Провести обучение моделей, вывести метрики качества обучения

Формулы регуляризации

$$L_1 = \lambda \sum_{i=1}^{n} \left| w_i \right|$$

$$L_2 = \lambda \sum_{i=1}^{n} w_i^2$$

$$ElasticNet = \lambda_1 \sum_{i=1}^{n} \left| w_i \right| + \lambda_2 \sum_{i=1}^{n} w_i^2$$

Линейная регрессия

На рисунках 1-2 приведен алгоритм линейной регрессии, обучение модели с внедрением регуляризации L1, L2, ElasticNet с выводом метрик качества обучения

```
class LinearRegressionMatrix:
    def __init__(self, lr=0.01, n_iters=1000, l1=0.01, l2=0.01):

        self.lr = lr
        self.n_iters = n_iters
        self.l1 = l1
        self.l2 = l2
        self.weights = None
        self.bias = None


        self.history_mse = []
        self.history_rmse = []
        self.history_mae = []

    def fit(self, X, y):
        X = np.array(X, dtype=float)
        y = np.array(y, dtype=float).reshape(-1)

        n_samples, n_features = X.shape

        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.n_iters):
          y_pred = np.dot(X, self.weights) + self.bias

          error = y_pred - y

          dw = (2 / n_samples) * np.dot(X.T, error)
          db = (2 / n_samples) * np.sum(error)

          if self.l1 > 0:
            dw += self.l1 * np.sign(self.weights)

          if self.l2 > 0:
            dw += 2 * self.l2 * self.weights


          self.weights -= self.lr * dw
          self.bias -= self.lr * db

          mse = np.mean(error**2)
          rmse = np.sqrt(mse)
          mae = np.mean(np.abs(error))

          self.history_mse.append(mse)
          self.history_rmse.append(rmse)
          self.history_mae.append(mae)

    def predict(self, X):
        X = np.array(X, dtype=float)
        return np.dot(X, self.weights) + self.bias
```

Рисунок 1. Класс модели линейной регрессии

```python
models = {
    "Без регуляризации": LinearRegressionMatrix(lr=0.01, n_iters=1000, l1=0.0, l2=0.0),
    "L1": LinearRegressionMatrix(lr=0.01, n_iters=1000, l1=0.01, l2=0.0),
    "L2": LinearRegressionMatrix(lr=0.01, n_iters=1000, l1=0.0, l2=0.01),
    "ElasticNet": LinearRegressionMatrix(lr=0.01, n_iters=1000, l1=0.01, l2=0.01)
}

results = []

for name, model in models.items():
    model.fit(X_train_scaled, y_train_scaled)
    results.append({
        'Модель': name,
        'MSE': model.history_mse[-1],
        'RMSE': model.history_rmse[-1],
        'MAE': model.history_mae[-1]
    })

df_results = pd.DataFrame(results)
df_results = df_results[['Модель', 'MSE', 'RMSE', 'MAE']]
df_results = round(df_results, 4)
print(df_results)
```

Рисунок 2. Обучение модели с внедрением регуляризации L1, L2, ElasticNet с выводом метрик качества обучения

На рисунках 3-6 представлен метод построения графиков оценки качества обучения модели при разных регуляризациях и его применение

```
def plot_histories_lin(histories, metric_name, zoom=False):
  min_val = min(min(hist) for hist in histories.values())
  max_val = max(max(hist) for hist in histories.values())

  plt.figure(figsize=(14,8))
  plt.subplot(1, 2, 1)
  for name, losses in histories.items():
    plt.plot(losses, label=name)
  plt.title(f"{metric_name} по итерациям")
  plt.xlabel("Итерация")
  plt.ylabel(metric_name)
  plt.legend()
  plt.grid(True)


  plt.subplot(1, 2, 2)
  for name, losses in histories.items():
      plt.plot(losses, label=name)
  plt.title(f"{metric_name} по итерациям")
  plt.xlabel("Итерация")
  plt.ylabel(metric_name)
  if zoom:
    plt.ylim(min_val * 0.99, min_val * 1.01)
  plt.legend()
  plt.grid(True)

  plt.tight_layout()
  plt.show()
```

Рисунок 3. Метод построения графиков оценки качества обучения модели при разных регуляризациях

```
histories_mse = {name: model.history_mse for name, model in models.items()}
plot_histories_lin(histories_mse, metric_name="MSE", zoom=True)
```

Рисунок 4. Применение метода построения графиков метрики MSE модели при разных регуляризациях

```
histories_rmse = {name: model.history_rmse for name, model in models.items()}
plot_histories_lin(histories_rmse, metric_name="RMSE", zoom=True)
```

Рисунок 5. Применение метода построения графиков метрики RMSE модели при разных регуляризациях

```
histories_mae = {name: model.history_mae for name, model in models.items()}
plot_histories_lin(histories_mae, metric_name="MAE", zoom=True)
```

Рисунок 6. Применение метода построения графиков метрики MAE модели при разных регуляризациях

Метрики оценки модели

Метрики ошибок модели линейной регрессии на последней эпохе и графики ошибок в динамике по итерациям представлены на рисунках 7-9

	Модель	MSE	RMSE	MAE
0	Без регуляризации	0.4067	0.6377	0.4740
1	L1	0.4068	0.6378	0.4736
2	L2	0.4068	0.6378	0.4741
3	ElasticNet	0.4070	0.6380	0.4738

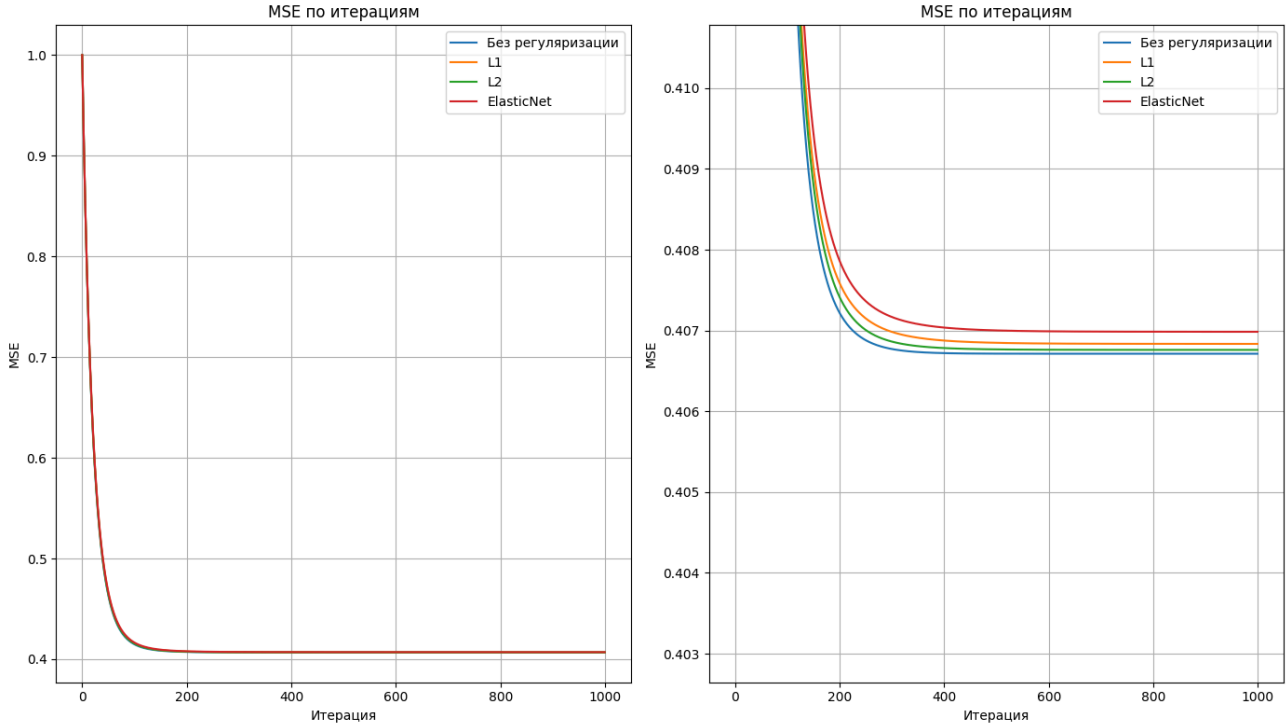

Рисунок 7. Графики MSE по итерациям в двух масштабах

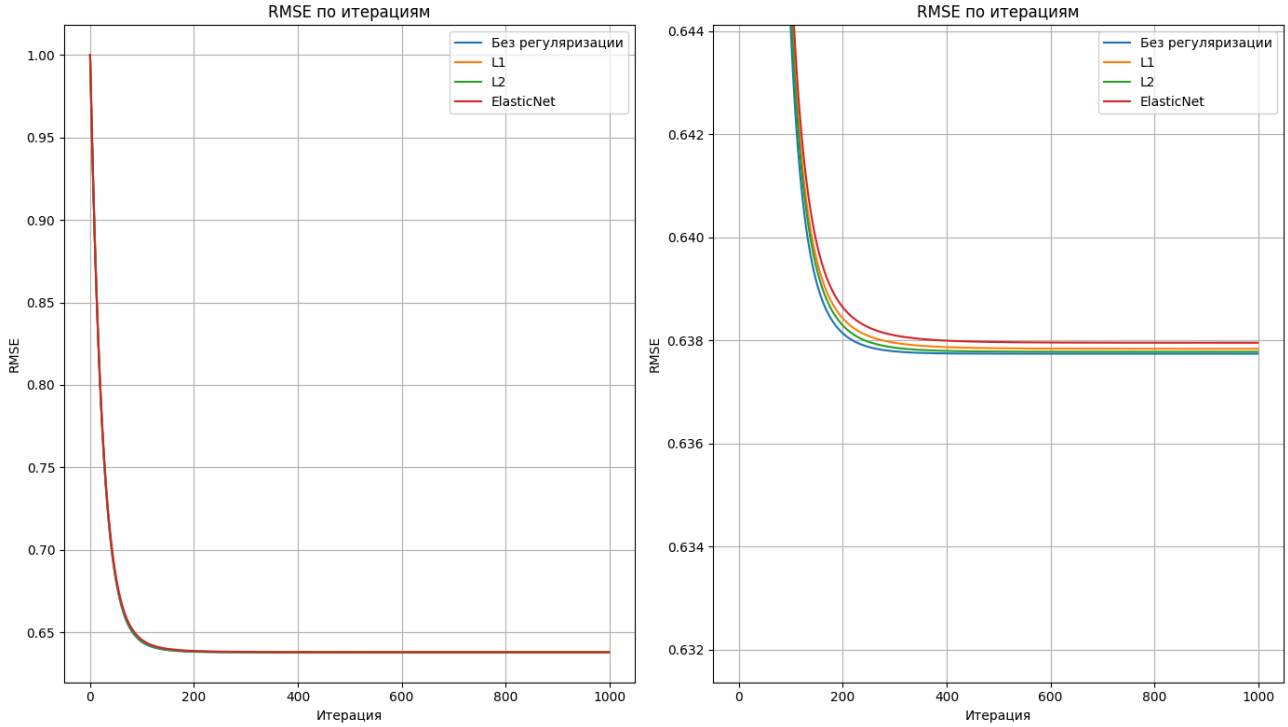

Рисунок 8. Графики RMSE по итерациям в двух масштабах

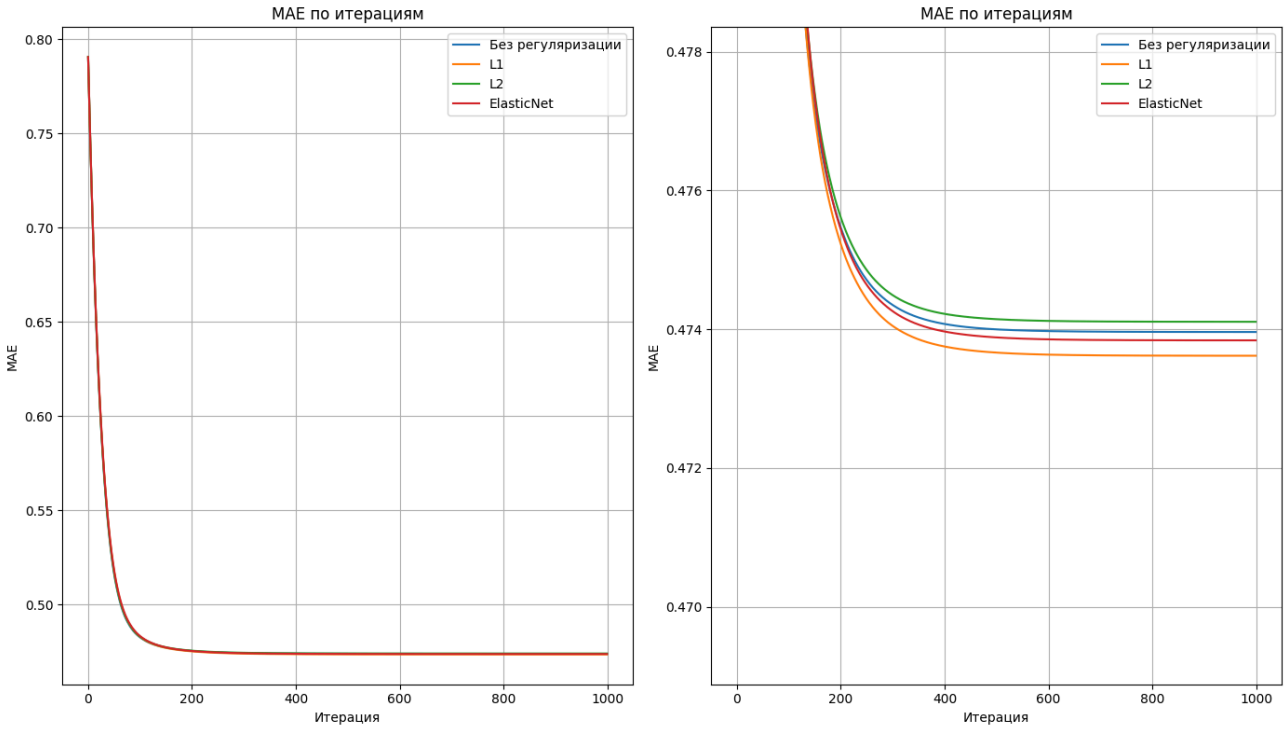

Рисунок 9. Графики MAE по итерациям в двух масштабах

Анализ результатов модели с разными видами регуляризации

Различия между моделями очень небольшие, то есть регуляризация почти не изменила качество модели на тесте. Это говорит о том, что модель не склонна к переобучению, даже без регуляризации. Но если анализировать даже незначительные изменения метрик, то можно сказать, что самые низкие значения MSE и RMSE показала версия без регуляризации, что показывает хорошее соответствие модели данным. При этом L1-регуляризация дала наилучшее значение MAE. Это связано с тем, что L1 уменьшает абсолютное значение весов и делает модель более устойчивой к отдельным выбросам. Поэтому MAE, как метрика, чувствительная к среднему отклонению, стала немного лучше.

Для линейной регрессии регуляризация почти не нужна, но L1 полезна для устойчивости к выбросам

Логистическая регрессия

На рисунках 10-11 приведен алгоритм логистической регрессии, обучение модели с внедрением регуляризации L1, L2, ElasticNet с выводом метрик качества обучения

```python
class LogisticRegressionMatrix:
    def __init__(self, lr=0.01, n_iters=1000, l1=0.01, l2=0.01):
        self.lr = lr
        self.n_iters = n_iters
        self.l1 = l1
        self.l2 = l2
        self.weights = None
        self.bias = None

        self.history_loss = []
        self.history_acc = []
        self.history_f1 = []

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

    def fit(self, X, y):
        X = np.array(X, dtype=float)
        y = np.array(y, dtype=float).reshape(-1)

        n_samples, n_features = X.shape

        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.n_iters):
            y_pred = self.sigmoid(np.dot(X, self.weights) + self.bias)

            err = y_pred - y

            dw = np.dot(X.T, err) / n_samples
            db = np.sum(err) / n_samples

            if self.l1 > 0:
              dw += self.l1 * np.sign(self.weights)

            if self.l2 > 0:
              dw += 2 * self.l2 * self.weights

            self.weights -= self.lr * dw
            self.bias -= self.lr * db

            eps = 1e-15
            y_pred_clipped = np.clip(y_pred, eps, 1 - eps)
            loss = -np.mean(y * np.log(y_pred_clipped) + (1 - y) * np.log(1 - y_pred_clipped))
            self.history_loss.append(loss)

            preds = (y_pred >= 0.5).astype(int)
            acc = accuracy_score(y, preds)
            f1 = f1_score(y, preds)
            self.history_acc.append(acc)
            self.history_f1.append(f1)


    def predict_proba(self, X):
        return self.sigmoid(np.dot(X, self.weights) + self.bias)

    def predict(self, X, threshold=0.5):
        return (self.predict_proba(X) >= threshold).astype(int)
```

Рисунок 10. Класс модели логистической регрессии

```python
models = {
    "Без регуляризации": LogisticRegressionMatrix(lr=0.01, n_iters=1000, l1=0.0, l2=0.0),
    "L1": LogisticRegressionMatrix(lr=0.01, n_iters=1000, l1=0.01, l2=0.0),
    "L2": LogisticRegressionMatrix(lr=0.01, n_iters=1000, l1=0.0, l2=0.01),
    "ElasticNet": LogisticRegressionMatrix(lr=0.01, n_iters=1000, l1=0.01, l2=0.01)
}

results = []

for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    results.append({
        'Модель': name,
        'Log-loss': model.history_loss[-1],
        'Accuracy': model.history_acc[-1],
        'F1-score': model.history_f1[-1]
    })

df_results = pd.DataFrame(results)
df_results = df_results[['Модель', 'Log-loss', 'Accuracy', 'F1-score']]
df_results = round(df_results, 4)
print(df_results)
```

Рисунок 11. Обучение модели с внедрением регуляризации L1, L2, ElasticNet с выводом метрик качества обучения

На рисунках 12-15 представлен метод построения графиков оценки качества обучения модели при разных регуляризациях и его применение

```python
def plot_histories_log(histories, metric_name, zoom=False, minimum=True):
  min_val = min(min(hist) for hist in histories.values())
  max_val = max(max(hist) for hist in histories.values())

  plt.figure(figsize=(18,6))
  plt.subplot(1, 2, 1)
  for name, losses in histories.items():
    plt.plot(losses, label=name)
  plt.title(f"{metric_name} по итерациям")
  plt.xlabel("Итерация")
  plt.ylabel(metric_name)
  plt.legend()
  plt.grid(True)


  plt.subplot(1, 2, 2)
  for name, losses in histories.items():
      plt.plot(losses, label=name)
  plt.title(f"{metric_name} по итерациям")
  plt.xlabel("Итерация")
  plt.ylabel(metric_name)
  if zoom and minimum:
    plt.ylim(min_val * 0.97, min_val * 1.12)
  if zoom and (not minimum):
    plt.ylim(max_val * 0.93, max_val * 1.02)
  plt.legend()
  plt.grid(True)

  plt.tight_layout()
  plt.show()
```

Рисунок 12. Метод построения графиков оценки качества обучения модели при разных регуляризациях

```python
histories_loss = {name: model.history_loss for name, model in models.items()}
plot_histories_log(histories_loss, metric_name="Loss", zoom=True, minimum=True)
```

Рисунок 13. Применение метода построения графиков метрики потерь модели при разных регуляризациях

```python
histories_acc = {name: model.history_acc for name, model in models.items()}
plot_histories_log(histories_acc, metric_name="Accuracy", zoom=True, minimum=False)
```

Рисунок 14. Применение метода построения графиков метрики точности модели при разных регуляризациях

```python
histories_f1 = {name: model.history_f1 for name, model in models.items()}
plot_histories_log(histories_f1, metric_name="F1-score", zoom=True, minimum=False)
```

Рисунок 15. Применение метода построения графиков метрики F1-score модели при разных регуляризациях

Метрики оценки модели

Метрики ошибок модели линейной регрессии на последней эпохе и графики ошибок в динамике по итерациям представлены на рисунках 16-18

	Модель	Log-loss	Accuracy	F1-score
0	Без регуляризации	0.4617	0.7921	0.7121
1	L1	0.4671	0.7921	0.7154
2	L2	0.4660	0.7935	0.7135
3	ElasticNet	0.4719	0.7879	0.7079

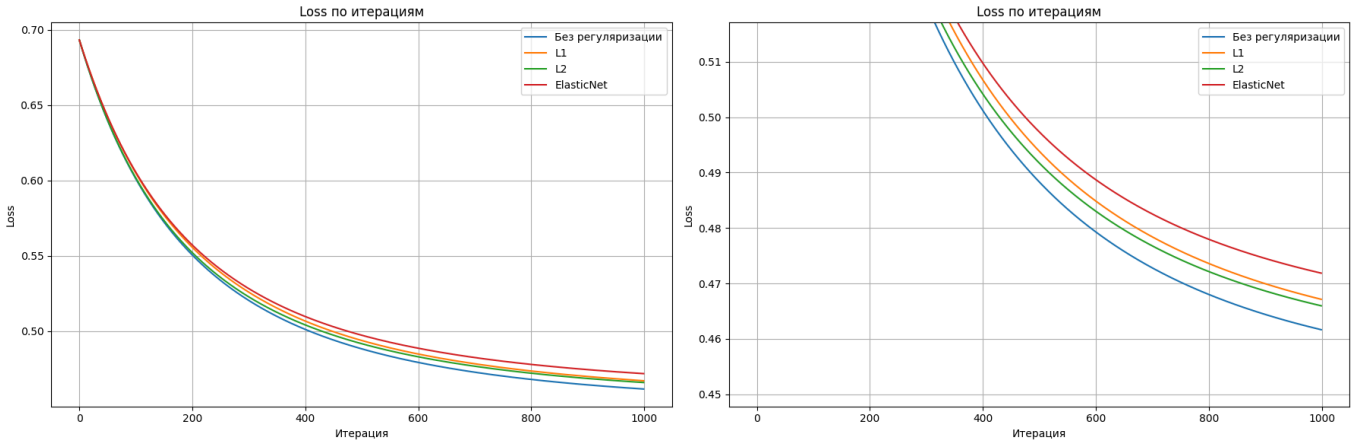

Рисунок 16. Графики потерь по итерациям в двух масштабах

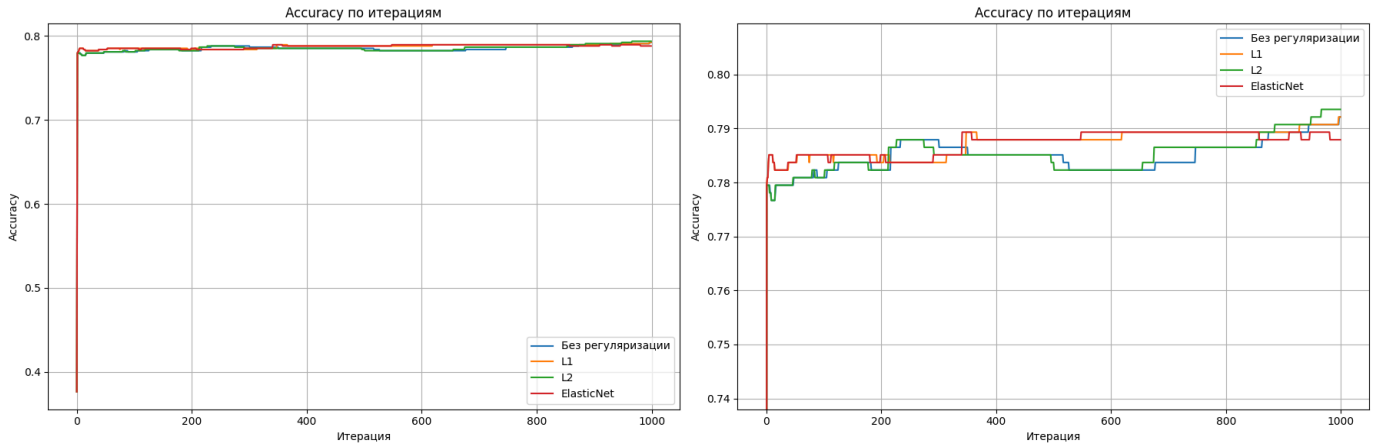

Рисунок 17. Графики точности по итерациям в двух масштабах

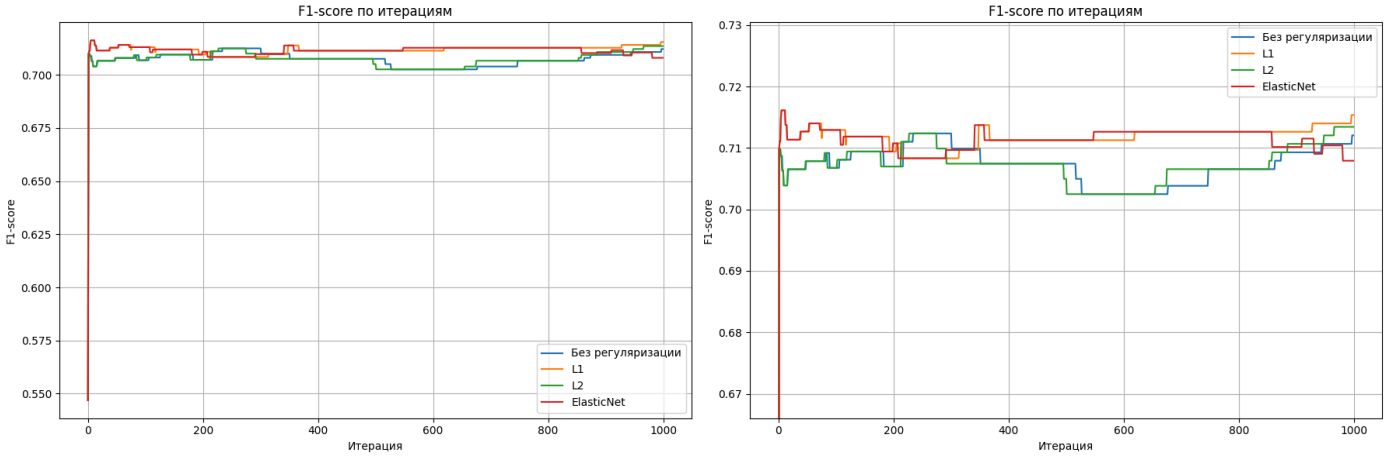

Рисунок 18. Графики F1-score по итерациям в двух масштабах

Анализ результатов модели с разными видами регуляризации

Различия между моделями относительно небольшие, однако влияние регуляризации заметно по метрикам потерь и качества классификации. В отличие от линейной регрессии, где поведение было почти одинаковым, здесь влияние регуляризации проявляется более явным образом.

Минимальный Log-loss показала модель без регуляризации, а наибольшее ухудшение ElasticNet, что означает слишком сильное сглаживание весов. Это указывает на то, что регуляризация может быть

избыточной в данной модели, при этом, уменьшив коэффициент регуляризации она показывает еще меньшие изменения в сравнении с её отсутствием

При это модель с L2-регуляризацией показала самую высокую точность, тк она стабилизирует веса, что улучшает способность модели классифицировать корректно. Модель без регуляризацией и L1 имеют одинаковую точность, а ElasticNet снова показал худший результат

В отношении полноты и точности наилучший результат показала модель с L1-регуляризацией, тк она склонна отбрасывать менее значимые признаки. L2 немного уступает, но лучше модели без регуляризации. ElasticNet снова на последнем месте.

Стоит отметить, что метрики ElasticNet и L1-регуляризация показывали наилучшие результаты в процессе обучения на метриках Accuracy и F1-score. То есть с данными видами регуляризации модель учится устойчивее

Для логистической регрессии L1 и L2 имеют смысл:

- L1 - для улучшения отношения полноты и точности, стабильного обучения
- L2 - для максимальной точности

KNN

На рисунке 19 представлен класс модели KNN, реализующий алгоритм регрессии и классификации. В качестве меры близости используется евклидово расстояние, а веса соседей уменьшаются экспоненциально

```
class KNearestNeighbors:
  def __init__(self, n_neighbors=5, regression=True):
    self.n_neighbors = n_neighbors
    self.regression = regression

  def fit(self, X_train, y_train):
    self.X_train = X_train
    self.y_train = y_train

  def euclidean_distances(self, x):
    return np.sqrt(np.sum((self.X_train - x)**2, axis=1))

  def make_prediction(self, x):
    distances = self.euclidean_distances(x)
    k_nearest_indexes = np.argsort(distances)[:self.n_neighbors]
    d = distances[k_nearest_indexes]
    y = self.y_train[k_nearest_indexes]

    weights = np.exp(-d)

    if self.regression:
      return np.sum(weights * y) / np.sum(weights)
    else:
      class_votes = {}
      for label, w in zip(y, weights):
          class_votes[label] = class_votes.get(label, 0) + w
      return max(class_votes, key=class_votes.get)

  def predict(self, X_test):
    return np.array([self.make_prediction(x) for x in X_test])
```

Рисунок 19. Класс модели KNN для регрессии и классификации

На рисунках 20-21 представлена предобработка данных для регрессии и классификации

```
data = pd.read_csv('/content/insurance.csv')
data = pd.get_dummies(data, columns=['sex', 'smoker', 'region'], drop_first=True)

X = data.drop('charges', axis=1)
y = data['charges'].values.astype('float32')

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler_X = StandardScaler()
X_train = scaler_X.fit_transform(X_train)
X_test = scaler_X.transform(X_test)

scaler_y = StandardScaler()
y_train = scaler_y.fit_transform(y_train.reshape(-1, 1)).ravel()
y_test = scaler_y.transform(y_test.reshape(-1, 1)).ravel()
```

Рисунок 20. Предобработка данных для регрессии

```
train = pd.read_csv('/content/fashion-mnist_train.csv')
test = pd.read_csv('/content/fashion-mnist_test.csv')

X_train = train.drop('label', axis=1).values.astype('float') / 255.0
y_train = train['label'].values.astype('int')

X_test = test.drop('label', axis=1).values.astype('float') / 255.0
y_test = test['label'].values.astype('int')
```

Рисунок 21. Предобработка данных для классификации

На рисунках 22-23 показан процесс обучения моделей KNN-регрессии и классификации с перебором числа соседей в диапазоне от 1 до 20. Для каждой модели рассчитывались метрики качества обучения, на основе которых было выявлено оптимальное число соседей при обучении

```
k_values = list(range(1, 21))
mse_list, rmse_list, mae_list = [], [], []

for k in k_values:
    knn_reg = KNearestNeighbors(n_neighbors=k, regression=True)
    knn_reg.fit(X_train, y_train)
    y_pred = knn_reg.predict(X_test)

    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_test, y_pred)

    mse_list.append(mse)
    rmse_list.append(rmse)
    mae_list.append(mae)

best_k_index = np.argmin(mse_list)
best_k = k_values[best_k_index]
print(f"Лучшее k по минимальной MSE: {best_k}, MSE = {mse_list[best_k_index]:.4f}")

plt.figure(figsize=(10,6))
plt.plot(k_values, mse_list, label='MSE', marker='o')
plt.plot(k_values, rmse_list, label='RMSE', marker='o')
plt.plot(k_values, mae_list, label='MAE', marker='o')
plt.xlabel('k (число соседей)')
plt.ylabel('Ошибка')
plt.title('Метрики регрессии по числу соседей')
plt.legend()
plt.grid(True)
plt.xticks(k_values)

plt.scatter(best_k, mse_list[best_k_index], color='red', s=100, label=f'Лучшее k={best_k}')
plt.legend()
plt.show()
```

Рисунок 22. Обучение модели KNN-регрессии с выявление оптимального числа соседей по минимальному MSE

```python
k_values = list(range(1, 21))
accuracy_list, f1_list = [], []

for k in k_values:
    knn_class = KNearestNeighbors(n_neighbors=k, regression=False)
    knn_class.fit(X_train, y_train)
    y_pred = knn_class.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='weighted')

    accuracy_list.append(acc)
    f1_list.append(f1)

best_k_index = np.argmax(f1_list)
best_k = k_values[best_k_index]
print(f"Лучшее k по максимальному F1: {best_k}, F1 = {f1_list[best_k_index]:.4f}")

plt.figure(figsize=(10,6))
plt.plot(k_values, accuracy_list, label='Accuracy', marker='o')
plt.plot(k_values, f1_list, label='F1-score', marker='o')
plt.xlabel('k (число соседей)')
plt.ylabel('Метрика')
plt.title('Метрики классификации по числу соседей')
plt.legend()
plt.grid(True)
plt.xticks(k_values)

plt.scatter(best_k, f1_list[best_k_index], color='red', s=100, label=f'Лучшее k={best_k}')
plt.legend()
plt.show()
```

Рисунок 23. Обучение модели KNN-классификации с выявление оптимального числа соседей по максимальному F1-score

Метрики оценки модели

На рисунках 24-25 представлены метрики качества обучения модели KNN-регрессии и классификации с выявлением оптимального числа соседей

```
Лучшее k по минимальной MSE: 7, MSE = 0.1940
```

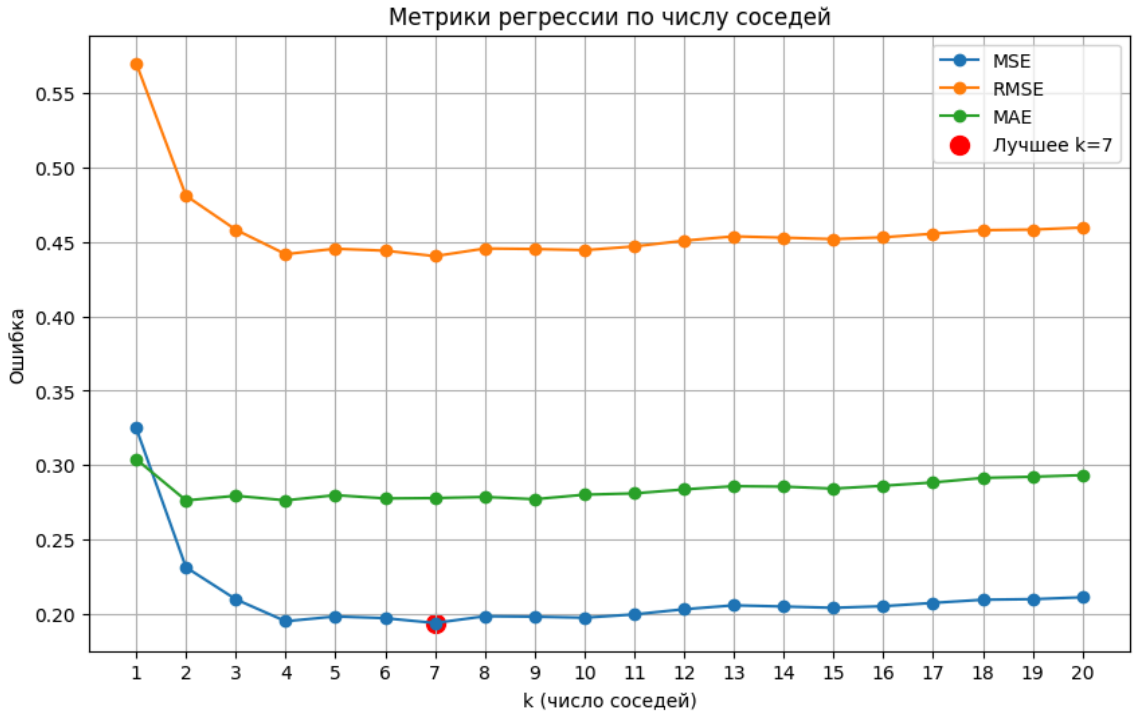

Рисунок 24. Метрики MSE, RMSE, MAE для KNN-регрессии

```
Лучшее k по максимальному F1: 6, F1 = 0.8651
```

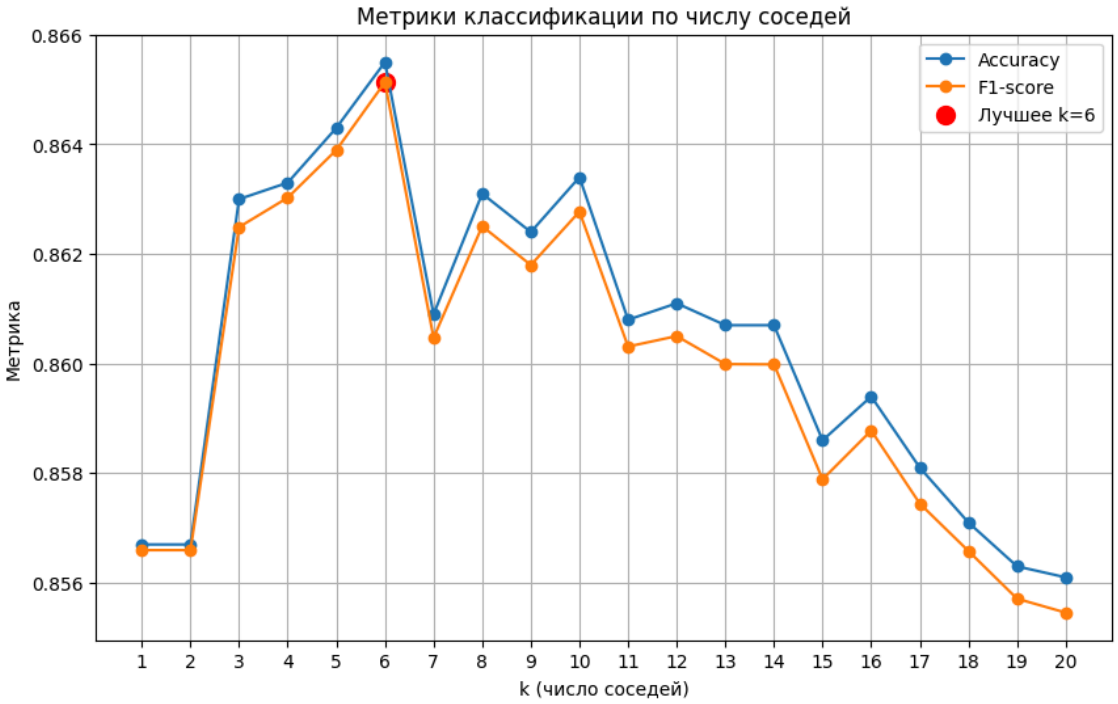

Рисунок 25. Метрики Accuracy и F1-score для KNN-классификации

Вывод

В ходе данной лабораторной работы были изучены 3 типа регуляризации L1, L2, ElasticNet и применены к моделям линейной и логистической регрессии, реализованных в прошлой работе. Был проведен анализ влияния регуляризации на обучение моделей, который показал, что к данным моделям, обученым на предоставленных датасетах лучше всего подходят:

“Для линейной регрессии регуляризация почти не нужна, но L1 полезна для устойчивости к выбросам”

“Для логистической регрессии L1 и L2 имеют смысл:

- L1 - для улучшения отношения полноты и точности, стабильного обучения
- L2 - для максимальной точности”

Также я ознакомилась с KNN и реализовала классы моделей регрессии и классификации по алгоритму ближайших соседей.