

# Отчет по заданию №1 "Метрические алгоритмы классификации".

## Алгоритм k ближайших соседей

### Содержание

<b>1</b>	<b>Введение</b>	<b>2</b>
<b>2</b>	<b>Эксперименты</b>	<b>2</b>
2.1	Скорость поиска ближайших соседей . . . . .	2
2.1.1	Дизайн эксперимента: . . . . .	2
2.1.2	Результаты . . . . .	2
2.1.3	Выводы . . . . .	2
2.2	Зависимость точности модели от количества соседей и метрики . . . . .	3
2.2.1	Дизайн эксперимента . . . . .	3
2.2.2	Результаты . . . . .	3
2.2.3	Выводы . . . . .	3
2.3	Сравнение точности взвешенного метода и метода без весов . . . . .	4
2.3.1	Дизайн эксперимента . . . . .	4
2.3.2	Результаты . . . . .	4
2.3.3	Выводы . . . . .	4
2.4	Анализ модели с лучшим качеством . . . . .	4
2.4.1	Дизайн эксперимента . . . . .	4
2.4.2	Результаты . . . . .	4
2.4.3	Выводы . . . . .	4
2.5	Применение трансформаций к тренировочной выборке . . . . .	5
2.5.1	Дизайн эксперимента . . . . .	5
2.5.2	Результаты . . . . .	6
2.5.3	Выводы . . . . .	6
2.6	Применение трансформаций к тестовой выборке . . . . .	7
2.6.1	Дизайн эксперимента . . . . .	7
2.6.2	Результаты . . . . .	7
2.6.3	Выводы . . . . .	7
<b>3</b>	<b>Сравнение экспериментов 5 и 6</b>	<b>8</b>
<b>4</b>	<b>Дополнение</b>	<b>8</b>
4.1	Adversarial validation . . . . .	8
4.2	Новые метрики расстояния . . . . .	9
4.3	Новые метрики качества . . . . .	9
4.4	Ансамбль алгоритмов . . . . .	10

# 1 Введение

В этом документе представлен отчет о проделанных экспериментах по практическому заданию №1, анализ результатов.

Краткое описание задания: необходимо реализовать алгоритмы  $k$  ближайших соседей и кросс-валидации, провести эксперименты с датасетом изображений цифр **MNIST**.

## 2 Эксперименты

В этом блоке приведены все обязательные эксперименты, которые изложены в формулировке задания.

### 2.1 Скорость поиска ближайших соседей

#### 2.1.1 Дизайн эксперимента:

Были протестированы 4 алгоритма поиска 5 ближайших соседей с разными размерами признакового пространства:

Алгоритмы:

- «my\_own»
- «brute»
- «kd\_tree»
- «ball\_tree»

Размеры признакового пространства:

- 10
- 20
- 100

#### 2.1.2 Результаты

Подробные результаты экспериментов приведены в таблице 1:

Таблица 1: Результаты эксперимента №1

размерность	алгоритм	время работы
10	my_own	68.07
	brute	7.84
	<b>kd_tree</b>	<b>0.44</b>
	ball_tree	1.72
20	my_own	76.82
	brute	7.95
	<b>kd_tree</b>	<b>1.40</b>
	ball_tree	6.63
100	my_own	78.31
	<b>brute</b>	<b>8.28</b>
	kd_tree	82.76
	ball_tree	97.67

#### 2.1.3 Выводы

Самым стабильным алгоритмом оказался «brute», который практически не деградирует с ростом размерности. Как и ожидалось, алгоритмы, реализованные на деревьях, сильно замедлились на признаковом пространстве размерности 100.

## 2.2 Зависимость точности модели от количества соседей и метрики

### 2.2.1 Дизайн эксперимента

В этом эксперименте была рассмотрена зависимость точности и времени работы модели  $k$  ближайших соседей от следующих параметров на 3 валидационных фолдах:

- $k$  от 1 до 10 (только влияние на точность).
- Евклидова или косинусная метрика.

### 2.2.2 Результаты

Зависимость средней точности от числа соседей для различных метрик приведена на графике 1. Измерения скорости для евклидовой и косинусной метрик приведены в таблице 2.

Рис. 1:

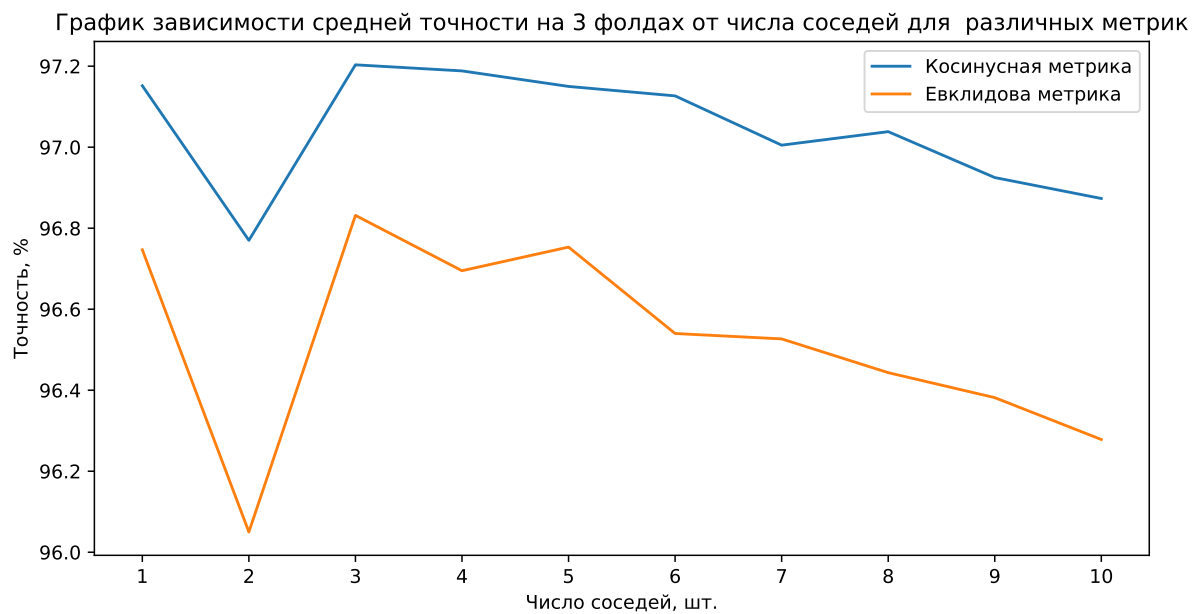


Таблица 2: Скорость работы метрик (измеренная при кросс-валидации на 3 фолдах)

метрика	время
евклидова	103.42
косинусная	101.69

### 2.2.3 Выводы

Из графика на Рис. 1 видно, что:

1. Наибольшая точность алгоритма достигается при  $k = 3$  для обеих метрик.
2. С увеличением количества соседей (при  $k \geq 5$ ) точность начинает убывать.
3. Точность алгоритма с косинусной метрикой превосходит точность алгоритма с евклидовой метрикой  $\forall k \in \{1, \dots, 10\}$

Из таблицы 2 видно, что скорость работы алгоритмов практически не зависит от выбора данных метрик.

## 2.3 Сравнение точности взвешенного метода и метода без весов

### 2.3.1 Дизайн эксперимента

Был рассмотрен метод  $k$  ближайших соседей «brute» с косинусным расстоянием  $\forall k \in \{1, \dots, 10\}$  в двух случаях:

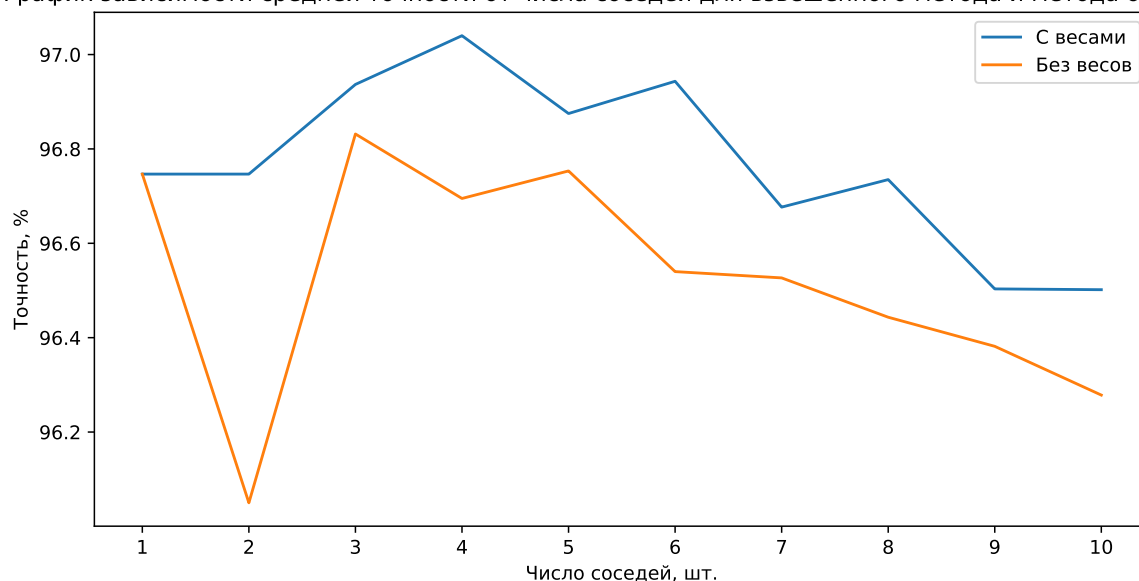
1. Взвешенный метод
2. Метод без весов

### 2.3.2 Результаты

Результаты эксперимента №3 приведены на графике 2

Рис. 2:

График зависимости средней точности от числа соседей для взвешенного метода и метода без весов



### 2.3.3 Выводы

По графику 2 можно заметить, что взвешенный метод лучше по точности во всех случаях, кроме  $k = 1$ , что является логичным, так как при предсказании по одному соседу веса бесполезны.

## 2.4 Анализ модели с лучшим качеством

### 2.4.1 Дизайн эксперимента

Применим лучший алгоритм («brute», косинусная метрика, с весами) к исходной обучающей и тестовой выборке и проанализируем матрицу ошибок (confusion matrix).

### 2.4.2 Результаты

Значения точности лучшего алгоритма на кросс-валидации, на тестовой выборке и значение точности лучшего алгоритма из интернета представлены в таблице 3. Матрица ошибок представлена на рисунке 3, Визуализированные объекты – на рисунке 4

### 2.4.3 Выводы

Можно увидеть, что самые частые ошибки происходят на парах цифр, представленных ниже:

Таблица 3:

алгоритм	условия	точность
реализованный по заданию	кросс-валидация	0.9741
	тестовая выборка	0.9752
сверточная нейронная сеть	тестовая выборка	0.9979

Рис. 3: Матрица ошибок для эксперимента №4

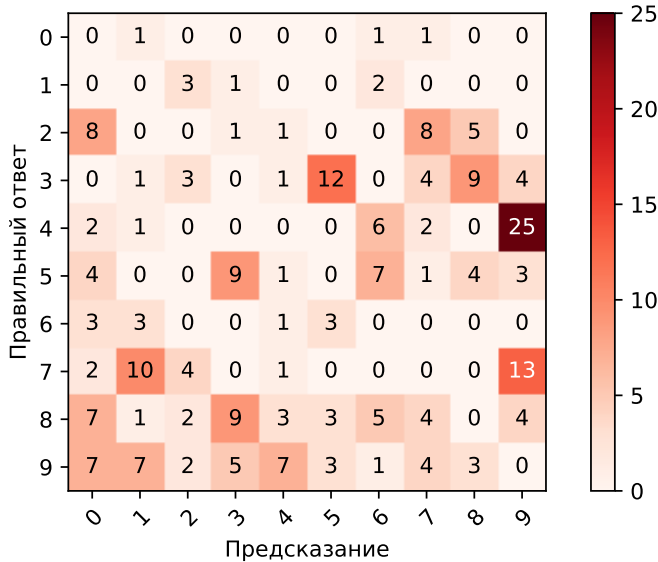
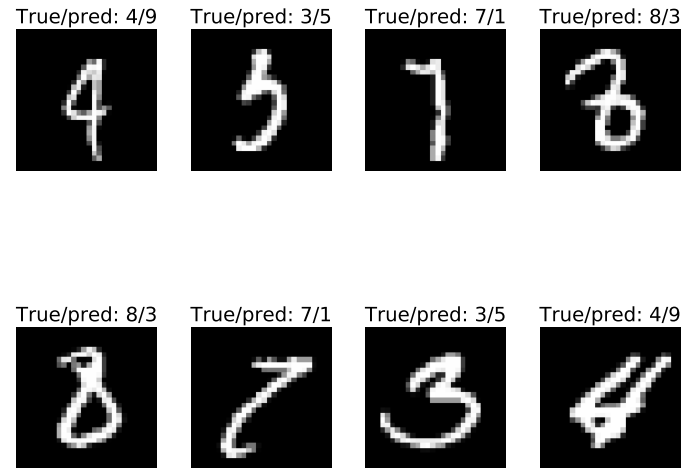


Рис. 4: Цифры с ошибками в предсказаниях



- 4 и 9
- 3 и 5

- 1 и 7
- 3 и 8

Эти пары похожи по написанию, поэтому алгоритму тяжелее отличить их друг от друга.

Рассмотрим теперь объекты, на которых произошли ошибки. У них можно заметить характерные особенности:

1. Засечки
2. Крючки
3. Отсутствие некоторых частей

В представленных случаях видно, что написанные цифры немного деформированы и похожи на другие в некоторых чертах. Эти особенности затрудняют классификацию для алгоритма.

## 2.5 Применение трансформаций к тренировочной выборке

### 2.5.1 Дизайн эксперимента

Пусть у нас есть  $m$  элементарных преобразований:  $\phi_j(x) : \mathbb{R}^d \rightarrow \mathbb{R}^d, j \in \{1, \dots, m\}$ , введем  $\phi_0(x) = x$ . Создадим новую обучающую выборку:

$$X_{new} = \left( \bigcup_{j=0}^m (\phi_j(x_i), y_i)_{i=1}^l \right)$$

Рассматриваемые преобразования<sup>1</sup>:

- Поворот на: 5, 10, 15° (в каждую из двух сторон).
- Смещение на 1, 2, 3 px (по каждой из двух размерностей).
- Дисперсия фильтра Гаусса<sup>2</sup>  $\sigma^2 = 0.5, 1, 1.5$ .

То есть применение конкретного преобразования позволяет нам расширить тренировочную выборку в 2 раза<sup>3</sup>.

## 2.5.2 Результаты

Результаты<sup>4</sup> приведены в таблице 4 (только по 3 лучшим параметра для каждого преобразования) и на рисунках 5, 6, 7, 8 представлены матрицы ошибок.

Таблица 4: Точность в зависимости от преобразования

тип преобразования	параметр	точность
отсутствует	—	0.945
поворот	5	0.954
	<b>10</b>	<b>0.956</b>
	15	0.954
сдвиг	(1, -1)	0.952
	<b>(1, 0)</b>	<b>0.953</b>
	(0, 1)	0.952
размытие	0.5	0.957
	<b>1.0</b>	<b>0.960</b>
	1.5	0.959

Рис. 5: Без преобр.

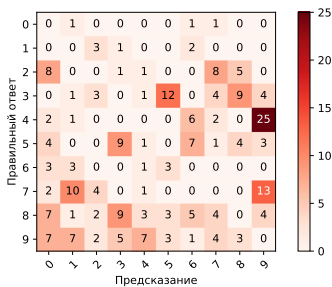


Рис. 6: Поворот на 10°

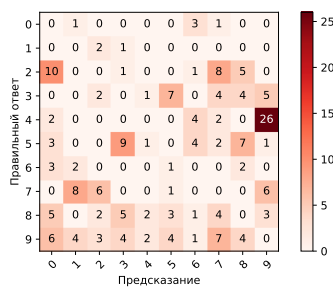


Рис. 7: Сдвиг на (1, 0) px

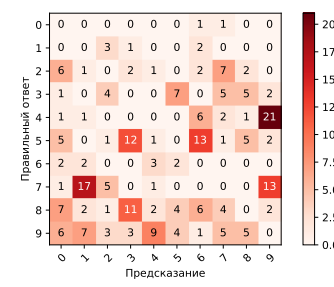
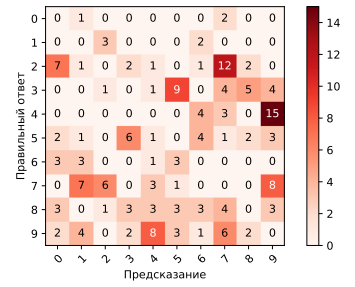


Рис. 8:  $\sigma^2 = 1$



## 2.5.3 Выводы

Можно отметить ряд фактов, которые видны из матриц ошибок:

1. Поворот позволяет алгоритму меньше путать цифры:

- 3 с 5

<sup>1</sup>Для трансформаций тренировочной выборки использовалась библиотека **scipy**.

<sup>2</sup>Важно заметить, что функция **scipy.ndimage.gaussian\_filter** принимает в качестве аргумента  $\sigma$ , а не  $\sigma^2$ .

<sup>3</sup>Замечание: если размножить тренировочную выборку до вызова функции кросс-валидации, то точность сильно возрастает (особенно при взвешенном методе). Это возникает из-за того, что преобразованные объекты остаются очень близкими к исходным. Чтобы избежать этого – нужно размножать каждый тренировочный фолд отдельно, а тестовый не трогать.

<sup>4</sup>Кросс-валидация проводилась на первых 10000 объектах из тренировочной выборки с целью экономии времени.

- 7 с 9
- 3 с 8

2. Сдвиг позволяет алгоритму меньше путать цифры:

- 4 с 9
- 3 с 8

3. Размытие Гаусса позволяет алгоритму меньше путать цифры:

- 4 с 9
- 7 с 9

## 2.6 Применение трансформаций к тестовой выборке

### 2.6.1 Дизайн эксперимента

Этот способ основан на преобразовании объектов тестовой выборки. Для каждого объекта тестовой выборки  $x'$  получим множество объектов  $\Phi(x') = \{\phi_j(x') | j \in \{1, \dots, m\}\}$ . Введем «расстояние» от множества  $\Phi(x')$  до объекта обучающей выборки  $x$ :

$$\rho(x, \Phi(x')) = \min_j \{\rho(x, \phi_j(x')) | j \in \{1, \dots, m\}\}$$

Вычисление  $k$  ближайших соседей для  $\Phi(x')$  можно организовать следующим образом:

1. Для каждого объекта из  $\Phi(x')$  найдем него  $k$  ближайших соседей из обучающей выборки  $X$
2. Среди всех найденных соседей выберем  $k$  объектов с наименьшим расстоянием

По полученному множеству значений ближайших соседей вычисляется ответ алгоритма.

### 2.6.2 Результаты

Результаты приведены в таблице 5 (только по 3 лучшим параметра для каждого преобразования) и на рисунках 9, 10, 11, 12 приведены матрицы ошибок.

Таблица 5: Точность в зависимости от преобразования

тип преобразования	параметр	точность
отсутствует	—	0.972
поворот	-15	0.976
	<b>-10</b>	<b>0.977</b>
	-5	0.974
сдвиг	(1, -1)	0.977
	<b>(1, 0)</b>	<b>0.978</b>
	(3, -1)	0.976
размытие	<b>0.5</b>	<b>0.972</b>
	1.0	0.963
	1.5	0.951

### 2.6.3 Выводы

Поворот на  $10^\circ$  против часовой стрелки позволяет избавиться от значительной части ошибок, при которых модель путает 3 с 5.

Сдвиг на (1, 0) px убирает ошибки, связанные с идентификацией 8 и 0, 9 и 7.

Размытие Гаусса с  $\sigma^2 = 0.5$  хорошо справляется с ошибками вида 3-5, но очень плохо себя показало с 4-9. Этому есть объяснение – 4 и 9 и без преобразований очень похожи, а после размытия Гаусса эти цифры становятся практически не различимы

Рис. 9: Без преобразований

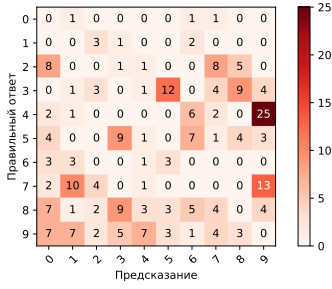


Рис. 10: Поворот на  $-10^\circ$

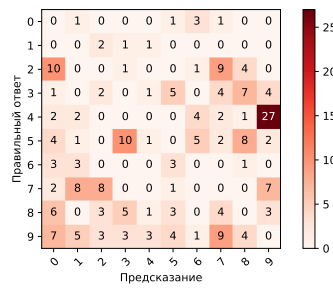


Рис. 11: Сдвиг на (1, 0) по x

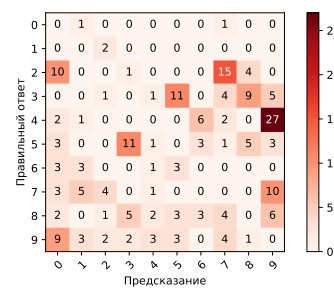
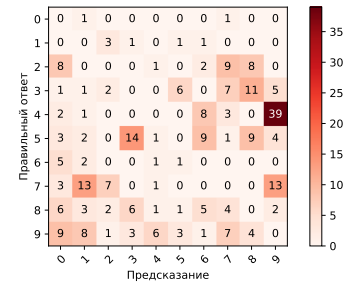


Рис. 12: Размытие Гаусса с  $\sigma^2 = 0.5$



### 3 Сравнение экспериментов 5 и 6

Данные эксперименты можно сравнить по двум характеристикам:

- Затрачиваемая память:

В условиях задачи тестовая выборка в несколько раз меньше, чем тренировочная, поэтому эффективнее увеличивать размеры именно теста.

- Точность алгоритма:

Результаты измерений точности приведены в таблице 6

Таблица 6:

номер эксперимента	точность
5	0.984
6	0.976

## 4 Дополнение

### 4.1 Adversarial validation

Рассмотрим альтернативный способ валидации. Основная идея заключается в следующем: нужно добиться того, чтобы алгоритм не смог отличать тренировочную выборку от тестовой.

Для реализации этого нужно сформировать новую выборку  $X = X_{train} \cup X_{test}$ , где  $X_{train}, X_{test}$  – объекты тренировочной и тестовой выборок соответственно (без меток классов). Поставим каждому объекту из  $X_{train}$  в соответствие метку «0», а каждому из  $X_{test}$  – «1». Таким образом получена новая тренировочная выборка, к которой можно применить метод кросс-валидации для оценки модели.

Выбор лучших параметров происходит на основе предсказаний модели для каждого фолда. За метрику качества удобнее всего брать площадь под кривой ошибок (**AUC ROC**).

Пусть даны две модели:  $M_1$  и  $M_2$ ;  $auc\_roc_1$ , и  $auc\_roc_2$  – значения площадей под кривой ошибок, усредненные по фолдам для каждой из моделей соответственно. Тогда модель  $M_1$  считается лучше модели  $M_2$ , если выполнено следующее неравенство:

$$|auc\_roc_1 - 0.5| < |auc\_roc_2 - 0.5|$$

С помощью представленного способа валидации будет более результативный отбор параметров аугментаций из экспериментов 2.5 и 2.6, так как он (способ) позволяет минимизировать отличия между тренировочной и тестовой выборкой.

Но есть и минус у **Adversarial validation** – обобщающая способность модели, настроенной по этому методу будет ниже, чем у модели, настроенной с помощью стандартного метода кросс-валидации, так как в первом случае она (первая модель) учитывает структуру тестовой выборки. Если новая тестовая выборка имеет совсем другое распределение, то первая модель с большей вероятностью выдаст качество хуже, чем вторая.



## 4.2 Новые метрики расстояния

Были проведены эксперименты на лучшей модели 4-х ближайших соседей с весами на различных метриках<sup>5</sup>:

- **Расстояние Чебышева:**  $\rho(x, y) = \max_{i \in \{1, \dots, d\}} |x_i - y_i|$ , где  $x, y \in \mathbb{R}^d$ .
- **Манхэттенское расстояние:**  $\rho(x, y) = \sum_{i=1}^d |x_i - y_i|$ , где  $x, y \in \mathbb{R}^d$ .
- **Расстояние Минковского:**  $\rho(x, y) = \left( \sum_{i=1}^d |x_i - y_i|^p \right)^{\frac{1}{p}}$ , где  $x, y \in \mathbb{R}^d, p \in \mathbb{R}$ .

Результаты экспериментов приведены в таблице 7 (результаты для евклидовой и косинусной метрик приведены для сравнения):

Таблица 7:

метрика	точность
Евклидова	<b>0.9752</b>
Косинусная	0.9714
Чебышева	0.8265
Манхэттенское расстояние	0.9659
Расстояние Минковского, p=3	0.9540

## 4.3 Новые метрики качества

Были реализованы следующие метрики качества<sup>6</sup> алгоритма:

- **Precision:**  $\frac{TP}{TP+FP}$
- **Recall:**  $\frac{TP}{TP+FN}$
- **Specifity:**  $\frac{TN}{TN+FP}$
- **F1 Score:**  $\frac{2 \times Recall \times Precision}{Recall + Precision}$

Пояснение обозначений: допустим, что решается задача многоклассовой классификации, где  $Y = \{y_1, \dots, y_m\}$  – множество уникальных классов. Каждая из представленных выше метрик считается отдельно  $\forall y_j, j \in 1, \dots, m$ . Рассмотрим их вычисление для конкретного класса  $y_j$ :

- **TP(True Positive)** – количество объектов класса  $y_j$ , которым алгоритм поставил метку  $y_j$
- **FP(False Positive)** – количество объектов, **не** принадлежащих классу  $y_j$ , которым алгоритм поставил метку  $y_j$
- **FN(False Negative)** – количество объектов класса  $y_j$ , которым алгоритм поставил метку, отличную от  $y_j$
- **TN(True Negative)** – количество объектов, **не** принадлежащих классу  $y_j$ , которым алгоритм поставил метку, отличную от  $y_j$

---

<sup>5</sup>Все метрики, описанные ниже, реализованы в модуле **distances.py**

<sup>6</sup>Все метрики качества, описанные ниже, реализованы в модуле **metrics.py**.

## 4.4 Ансамбль алгоритмов

В данной части реализован простейший ансамбль алгоритмов<sup>7</sup> – пусть имеется множество моделей  $M_1, \dots, M_t$  и соответственно их предсказания  $pred_1, \dots, pred_t$  на тестовом объекте  $x \in X_{test}$ , где  $X_{test} \in \mathbb{R}^{q \times d}$ . Тогда предсказание ансамбля моделей для объекта  $x$  определяется следующим образом:

$$pred(x) = \arg \max_c g_c(x)$$

, где  $g_c(x) = \sum_{m=1}^t w_k \times \mathbb{1}[pred_m = c]$ ,  $c = 1, 2, \dots, C$ , где  $w_k$  - некоторые веса (например, прямо пропорциональные качеству на кросс-валидации),  $C$  - количество классов.

В результате перебора различных вариаций ансамбля моделей удалось улучшить точность (ассигасу):

количество моделей	точность
1	0.9843
4	<b>0.9844</b>

### Анализ результата:

Ансамбль моделей помог улучшить точность, но эта разница совсем маленькая, а для её получения потребовалось в 4 раза больше времени. В рамках данной задачи лучше использовать одну модель.

---

<sup>7</sup>Реализован в модуле `ensemble.py`