**Импортируемые библиотеки**

python

Копировать код

import os

* **os**: Модуль для взаимодействия с операционной системой, включая работу с файловой системой (путь, проверка существования файлов и т.д.).

python

Копировать код

import numpy as np

* **numpy**: Библиотека для работы с массивами и выполнения численных операций. Очень полезна для научных вычислений и обработки данных.

python

Копировать код

import matplotlib.pyplot as plt

* **matplotlib.pyplot**: Библиотека для создания графиков и визуализаций данных. **pyplot** предоставляет интерфейс для создания фигур и осей, нанесения данных и т.д.

python

Копировать код

import cv2

* **cv2** (OpenCV): Библиотека для обработки изображений и компьютерного зрения. Используется для чтения, обработки и отображения изображений.

python

Копировать код

from pycocotools.coco import COCO

* **pycocotools.coco**: Модуль для работы с аннотациями COCO (Common Objects in Context) — популярного датасета для задач распознавания объектов.

python

Копировать код

from collections import Counter

* **collections.Counter**: Класс для подсчета хэшируемых объектов. Используется для подсчета частоты элементов в списке или массиве.

python

Копировать код

from sklearn.model\_selection import KFold, GridSearchCV

* **sklearn.model\_selection.KFold**: Класс для проведения k-блочной перекрестной проверки (k-fold cross-validation).
* **sklearn.model\_selection.GridSearchCV**: Инструмент для поиска оптимальных гиперпараметров модели через кросс-валидацию.

python

Копировать код

from sklearn.ensemble import RandomForestClassifier, BaggingClassifier

* **RandomForestClassifier**: Классификатор на основе ансамбля деревьев решений (Random Forest).
* **BaggingClassifier**: Классификатор на основе ансамбля с бутстрэпингом (bagging).

python

Копировать код

from sklearn.linear\_model import LogisticRegression

* **LogisticRegression**: Классификатор логистической регрессии.

python

Копировать код

from sklearn.svm import SVC

* **SVC**: Классификатор опорных векторов (Support Vector Classifier).

python

Копировать код

from sklearn.neighbors import KNeighborsClassifier

* **KNeighborsClassifier**: Классификатор k-ближайших соседей (k-Nearest Neighbors).

python

Копировать код

from sklearn.naive\_bayes import GaussianNB

* **GaussianNB**: Наивный байесовский классификатор с нормальным (Гауссовым) распределением.

python

Копировать код

from sklearn.tree import DecisionTreeClassifier

* **DecisionTreeClassifier**: Классификатор на основе дерева решений.

python

Копировать код

from sklearn.neural\_network import MLPClassifier

* **MLPClassifier**: Многослойный перцептрон (нейронная сеть) для классификации.

python

Копировать код

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix, roc\_curve, auc

* **accuracy\_score**: Метрика точности.
* **precision\_score**: Метрика точности (precision).
* **recall\_score**: Метрика полноты (recall).
* **f1\_score**: F1-метрика (гармоническое среднее точности и полноты).
* **confusion\_matrix**: Матрица ошибок (confusion matrix).
* **roc\_curve**: ROC-кривая (Receiver Operating Characteristic).
* **auc**: Площадь под ROC-кривой (Area Under Curve).

python

Копировать код

from sklearn.decomposition import PCA

* **PCA**: Метод главных компонент (Principal Component Analysis) для понижения размерности данных.

python

Копировать код

from sklearn.manifold import TSNE

* **TSNE**: t-SNE (t-Distributed Stochastic Neighbor Embedding) — метод нелинейного снижения размерности для визуализации данных.

python

Копировать код

from sklearn.cluster import KMeans, DBSCAN

* **KMeans**: Алгоритм кластеризации k-средних.
* **DBSCAN**: Алгоритм кластеризации плотности (Density-Based Spatial Clustering of Applications with Noise).

python

Копировать код

from sklearn.preprocessing import LabelEncoder, StandardScaler

* **LabelEncoder**: Преобразователь для кодирования меток.
* **StandardScaler**: Масштабирование данных до стандартизированного вида (нормализация).

python

Копировать код

from sklearn.pipeline import Pipeline

* **Pipeline**: Инструмент для последовательного выполнения нескольких шагов обработки данных и обучения модели.

python

Копировать код

import seaborn as sns

* **seaborn**: Библиотека для визуализации данных, основанная на matplotlib, с более удобным интерфейсом и расширенными возможностями.

python

Копировать код

import time

* **time**: Модуль для работы со временем (засекание времени выполнения и т.д.).

**Путь к данным COCO**

python

Копировать код

dataDir = r'F:\git\Kursach' dataType = 'val2017' annFile = os.path.join(dataDir, f'annotations/instances\_{dataType}.json')

* **dataDir**: Указывает на директорию с данными.
* **dataType**: Определяет тип данных (например, валидационный набор за 2017 год).
* **annFile**: Полный путь к файлу аннотаций COCO.

**Загрузка аннотаций COCO**

python

Копировать код

coco = COCO(annFile)

* Создается объект **COCO** для работы с аннотациями.

**Получение всех идентификаторов изображений**

python

Копировать код

imgIds = coco.getImgIds() images = coco.loadImgs(imgIds[:50]) # Загружаем первые 50 изображений для тестирования

* **imgIds**: Список идентификаторов всех изображений.
* **images**: Список данных о первых 50 изображениях.

**Функции для работы с изображениями**

**Отображение изображения с аннотациями**

python

Копировать код

def show\_image\_with\_annotations(img, anns): plt.imshow(img) coco.showAnns(anns) plt.show()

* **show\_image\_with\_annotations**: Функция для отображения изображения с аннотациями. Использует **plt.imshow** для отображения изображения и **coco.showAnns** для отображения аннотаций.

**Загрузка изображения**

python

Копировать код

def load\_image(img\_path): if os.path.exists(img\_path): img = cv2.imread(img\_path) if img is not None: return cv2.cvtColor(img, cv2.COLOR\_BGR2RGB) else: print(f"Ошибка загрузки изображения: {img\_path}") else: print(f"Файл не найден: {img\_path}") return None

* **load\_image**: Функция для загрузки изображения с указанного пути. Использует **cv2.imread** для чтения изображения и **cv2.cvtColor** для преобразования цветовой схемы из BGR в RGB.

**Загрузка и отображение примера изображения**

python

Копировать код

image = images[0] img\_path = os.path.join(dataDir, dataType, image['file\_name']) print(f"Путь к изображению: {img\_path}") img = load\_image(img\_path) if img is not None: anns\_ids = coco.getAnnIds(imgIds=image['id']) anns = coco.loadAnns(anns\_ids) show\_image\_with\_annotations(img, anns)

* Загружается первое изображение из списка и отображается вместе с аннотациями.

**Загрузка данных и преобразование bounding boxes**

python

Копировать код

def load\_data(coco, imgIds): images = [] labels = [] for idx, imgId in enumerate(imgIds): img\_info = coco.loadImgs(imgId)[0] img\_path = os.path.join(dataDir, dataType, img\_info['file\_name']) img = load\_image(img\_path) if img is not None: images.append(img) annIds = coco.getAnnIds(imgIds=img\_info['id']) anns = coco.loadAnns(annIds) img\_labels = set() for ann in anns: category\_id = ann['category\_id'] img\_labels.add(category\_id) labels.append(list(img\_labels)) if idx % 10 == 0: print(f"Загружено изображений: {idx}") return images, labels

* **load\_data**: Функция для загрузки изображений и меток. Обходит каждый идентификатор изображения, загружает изображение и соответствующие аннотации, сохраняет их в списки.

**Загрузка данных**

python

Копировать код

images, labels = load\_data(coco, imgIds[:50]) # Загружаем первые 50 изображений для примера

* Загружаются первые 50 изображений и их метки.

**Анализ распределения классов объектов**

python

Копировать код

all\_labels = [label for sublist in labels for label in sublist] label\_counter = Counter(all\_labels) num\_classes = max(label\_counter.keys()) + 1

* **all\_labels**: Все метки объектов.
* **label\_counter**: Подсчет частоты каждой метки.
* **num\_classes**: Количество уникальных классов объектов.

**Визуализация распределения классов**

python

Копировать код

plt.figure(figsize=(10, 5)) plt.bar(label\_counter.keys(), label\_counter.values()) plt.xlabel('Class ID') plt.ylabel('Frequency') plt.title('Distribution of Object Classes in COCO Dataset') plt.show()

* Создается бар-график для визуализации распределения классов объектов.

**Преобразование данных для обучения**

python

Копировать код

train\_images = np.array([cv2.resize(img, (224, 224)).flatten() for img in images[:40]]) val\_images = np.array([cv2.resize(img, (224, 224)).flatten() for img in images[40:]])

* Изображения масштабируются до 224x224 и преобразуются в одномерные массивы.
* **train\_images**: Обучающая выборка (первые 40 изображений).
* **val\_images**: Валидационная выборка (последние 10 изображений).

**Создание "one-hot" представления для меток**

python

Копировать код

def create\_one\_hot\_labels(labels, num\_classes): one\_hot\_labels = np.zeros((len(labels), num\_classes)) for i, label\_list in enumerate(labels): for label in label\_list: one\_hot\_labels[i, label] = 1 return one\_hot\_labels train\_labels = create\_one\_hot\_labels(labels[:40], num\_classes) val\_labels = create\_one\_hot\_labels(labels[40:], num\_classes)

* **create\_one\_hot\_labels**: Функция для создания "one-hot" представления меток.
* **train\_labels**: "One-hot" метки для обучающей выборки.
* **val\_labels**: "One-hot" метки для валидационной выборки.

**Конвертация "one-hot" представления в плоские метки для sklearn**

python

Копировать код

train\_labels\_flat = np.argmax(train\_labels, axis=1) val\_labels\_flat = np.argmax(val\_labels, axis=1)

* Преобразование "one-hot" меток в плоские метки.

**Использование LabelEncoder для кодирования меток**

python

Копировать код

le = LabelEncoder() all\_labels\_flat = np.concatenate([train\_labels\_flat, val\_labels\_flat]) le.fit(all\_labels\_flat) train\_labels\_flat = le.transform(train\_labels\_flat) val\_labels\_flat = le.transform(val\_labels\_flat)

* **LabelEncoder** используется для последовательного кодирования меток.

**Проверка на наличие пропущенных данных**

python

Копировать код

def check\_missing\_data(images, labels): missing\_images = [i for i in images if i is None] missing\_labels = [i for i in labels if i is None] print(f"Number of missing images: {len(missing\_images)}") print(f"Number of missing labels: {len(missing\_labels)}") check\_missing\_data(images, labels)

* Функция проверяет наличие пропущенных изображений и меток.

**Проверка аномалий в метках**

python

Копировать код

def check\_anomalies(labels): label\_counts = Counter([label for sublist in labels for label in sublist]) print("Label counts:", label\_counts) plt.bar(label\_counts.keys(), label\_counts.values()) plt.xlabel('Class ID') plt.ylabel('Frequency') plt.title('Label Distribution') plt.show() check\_anomalies(labels)

* Функция анализирует и визуализирует распределение меток.

**Понижение размерности с помощью PCA**

python

Копировать код

pca = PCA(n\_components=2) pca\_result = pca.fit\_transform(train\_images) plt.scatter(pca\_result[:, 0], pca\_result[:, 1], c=train\_labels\_flat, cmap='viridis') plt.title("PCA of Image Data") plt.colorbar() plt.show()

* **PCA**: Метод главных компонент для понижения размерности данных до 2D и их визуализации.

**Понижение размерности с помощью t-SNE**

python

Копировать код

tsne = TSNE(n\_components=2) tsne\_result = tsne.fit\_transform(train\_images) plt.scatter(tsne\_result[:, 0], tsne\_result[:, 1], c=train\_labels\_flat, cmap='viridis') plt.title("t-SNE of Image Data") plt.colorbar() plt.show()

* **t-SNE**: Метод нелинейного снижения размерности для визуализации данных.

**Кластеризация с помощью KMeans**

python

Копировать код

kmeans = KMeans(n\_clusters=min(num\_classes, 10), random\_state=42) clusters = kmeans.fit\_predict(train\_images) plt.scatter(pca\_result[:, 0], pca\_result[:, 1], c=clusters, cmap='viridis') plt.title("KMeans Clustering of Image Data") plt.colorbar() plt.show()

* **KMeans**: Алгоритм кластеризации для группировки данных.

**Поиск аномалий с помощью DBSCAN**

python

Копировать код

dbscan = DBSCAN(eps=3, min\_samples=5) anomalies = dbscan.fit\_predict(train\_images) plt.scatter(pca\_result[:, 0], pca\_result[:, 1], c=anomalies, cmap='viridis') plt.title("DBSCAN Anomaly Detection") plt.colorbar() plt.show()

* **DBSCAN**: Алгоритм для обнаружения аномалий в данных.

**Обучение нескольких моделей**

python

Копировать код

models = [ ("SVM", SVC(max\_iter=1000)), ("KNN", KNeighborsClassifier()), ("Naive Bayes", GaussianNB()), ("Decision Tree", DecisionTreeClassifier(random\_state=42)), ("Random Forest", RandomForestClassifier(n\_estimators=50, random\_state=42)), ("Bagging", BaggingClassifier(estimator=DecisionTreeClassifier(), n\_estimators=10, random\_state=42)), ("MLP", MLPClassifier(max\_iter=300)) ] kf = KFold(n\_splits=3, shuffle=True, random\_state=42) model\_accuracies = [] for name, model in models: accuracies = [] start\_time = time.time() for train\_index, test\_index in kf.split(train\_images): X\_train, X\_test = train\_images[train\_index], train\_images[test\_index] y\_train, y\_test = train\_labels\_flat[train\_index], train\_labels\_flat[test\_index] model.fit(X\_train, y\_train) y\_pred = model.predict(X\_test) accuracy = accuracy\_score(y\_test, y\_pred) accuracies.append(accuracy) avg\_accuracy = np.mean(accuracies) end\_time = time.time() training\_time = end\_time - start\_time model\_accuracies.append((name, avg\_accuracy, training\_time)) print(f"{name} Cross-Validation Accuracy: {avg\_accuracy:.2f}, Training Time: {training\_time:.2f} seconds")

* Обучение и оценка нескольких моделей с использованием k-блочной перекрестной проверки.
* **KFold**: k-блочная перекрестная проверка для разбиения данных на тренировочные и тестовые наборы.
* **models**: Список моделей для обучения.
* **model\_accuracies**: Список для хранения точности и времени обучения моделей.

**Усовершенствование моделей (гиперпараметры, регуляризация)**

**Grid Search для Random Forest**

python

Копировать код

param\_grid\_rf = { 'n\_estimators': [50, 100], 'max\_depth': [None, 10, 20], 'min\_samples\_split': [2, 5], 'min\_samples\_leaf': [1, 2] } rf\_grid\_search = GridSearchCV(RandomForestClassifier(random\_state=42), param\_grid\_rf, cv=3, n\_jobs=-1, verbose=2) rf\_grid\_search.fit(train\_images, train\_labels\_flat) print(f"Best parameters for Random Forest: {rf\_grid\_search.best\_params\_}") print(f"Best cross-validation accuracy for Random Forest: {rf\_grid\_search.best\_score\_:.2f}")

* **GridSearchCV**: Поиск оптимальных гиперпараметров для **RandomForestClassifier** с использованием сеточного поиска и кросс-валидации.

**Замеры времени обучения**

python

Копировать код

start\_time = time.time() rf\_model = RandomForestClassifier(\*\*rf\_grid\_search.best\_params\_, random\_state=42) rf\_model.fit(train\_images, train\_labels\_flat) end\_time = time.time() training\_time = end\_time - start\_time print(f"Training time for Random Forest: {training\_time:.2f} seconds")

* Замер времени обучения модели **RandomForestClassifier**.

**Оценка модели Random Forest на тестовых данных**

python

Копировать код

y\_pred\_rf = rf\_model.predict(val\_images) rf\_accuracy = accuracy\_score(val\_labels\_flat, y\_pred\_rf) rf\_precision = precision\_score(val\_labels\_flat, y\_pred\_rf, average='weighted', zero\_division=0) rf\_recall = recall\_score(val\_labels\_flat, y\_pred\_rf, average='weighted', zero\_division=0) rf\_f1 = f1\_score(val\_labels\_flat, y\_pred\_rf, average='weighted', zero\_division=0) print(f"Random Forest Test Accuracy: {rf\_accuracy:.2f}") print(f"Random Forest Test Precision: {rf\_precision:.2f}") print(f"Random Forest Test Recall: {rf\_recall:.2f}") print(f"Random Forest Test F1 Score: {rf\_f1:.2f}")

* Оценка модели **RandomForestClassifier** на валидационном наборе данных с использованием различных метрик.

**Визуализация результатов (Матрица ошибок и ROC-кривые)**

**Матрица ошибок**

python

Копировать код

conf\_matrix = confusion\_matrix(val\_labels\_flat, y\_pred\_rf) sns.heatmap(conf\_matrix, annot=True, fmt="d") plt.title("Confusion Matrix for Random Forest") plt.xlabel("Predicted Label") plt.ylabel("True Label") plt.show()

* Визуализация матрицы ошибок (confusion matrix) для модели **RandomForestClassifier**.

**ROC-кривые**

python

Копировать код

y\_pred\_proba\_rf = rf\_model.predict\_proba(val\_images) fpr, tpr, \_ = roc\_curve(val\_labels\_flat, y\_pred\_proba\_rf[:, 1], pos\_label=1) roc\_auc = auc(fpr, tpr) plt.plot(fpr, tpr, label=f'Random Forest (area = {roc\_auc:.2f})') plt.plot([0, 1], [0, 1], 'k--') plt.xlim([0.0, 1.0]) plt.ylim([0.0, 1.05]) plt.xlabel('False Positive Rate') plt.ylabel('True Positive Rate') plt.title('ROC Curve') plt.legend(loc="lower right") plt.show()

* Визуализация ROC-кривой для модели **RandomForestClassifier**.

**Оценка модели мультиклассовой логистической регрессии на тестовых данных**

python

Копировать код

logreg\_model = Pipeline([ ('scaler', StandardScaler()), ('logreg', LogisticRegression(max\_iter=1000, multi\_class='multinomial', solver='lbfgs')) ]) logreg\_model.fit(train\_images, train\_labels\_flat) y\_pred\_logreg = logreg\_model.predict(val\_images) logreg\_accuracy = accuracy\_score(val\_labels\_flat, y\_pred\_logreg) logreg\_precision = precision\_score(val\_labels\_flat, y\_pred\_logreg, average='weighted', zero\_division=0) logreg\_recall = recall\_score(val\_labels\_flat, y\_pred\_logreg, average='weighted', zero\_division=0) logreg\_f1 = f1\_score(val\_labels\_flat, y\_pred\_logreg, average='weighted', zero\_division=0) print(f"Logistic Regression Test Accuracy: {logreg\_accuracy:.2f}") print(f"Logistic Regression Test Precision: {logreg\_precision:.2f}") print(f"Logistic Regression Test Recall: {logreg\_recall:.2f}") print(f"Logistic Regression Test F1 Score: {logreg\_f1:.2f}")

* Создание и оценка модели логистической регрессии на валидационном наборе данных с использованием различных метрик.

Этот код представляет собой комплексный подход к обработке и анализу изображений из датасета COCO, включающий этапы загрузки данных, их предобработки, анализа распределения классов, понижения размерности, кластеризации, обнаружения аномалий, обучения и оценки различных моделей машинного обучения.

**Дополнительные и измененные импорты**

python

Копировать код

from sklearn.model\_selection import train\_test\_split import pandas as pd import skimage.io as io

* **train\_test\_split**: Разделяет массивы или матрицы на случайные обучающие и тестовые подмножества.
* **pandas**: Библиотека для анализа данных и работы с таблицами (DataFrame).
* **skimage.io**: Модуль из библиотеки scikit-image для работы с изображениями (чтение, запись и т.д.).

**Проверка существования директории для скриншотов**

python

Копировать код

screenshot\_dir = 'screen' if not os.path.exists(screenshot\_dir): os.makedirs(screenshot\_dir)

* Создается директория **screen** для сохранения скриншотов, если она не существует.

**Загрузка и отображение нескольких изображений**

python

Копировать код

# Визуализация нескольких изображений из набора данных fig, axes = plt.subplots(3, 3, figsize=(15, 15)) fig.suptitle('Исходный набор данных', fontsize=20) for i, img\_info in enumerate(img\_infos[:9]): img\_path = os.path.join(data\_dir, data\_type, img\_info['file\_name']) if os.path.isfile(img\_path): image = io.imread(img\_path) row, col = divmod(i, 3) axes[row, col].imshow(image) axes[row, col].axis('off') else: print(f"Файл не найден: {img\_path}") plt.savefig(os.path.join(screenshot\_dir, 'original\_dataset.png')) plt.show()

* Загружается и отображается несколько изображений из набора данных. Используется **skimage.io.imread** для чтения изображений.

**Преобразование данных в DataFrame**

python

Копировать код

data = [] for img\_info in img\_infos: img\_path = os.path.join(data\_dir, data\_type, img\_info['file\_name']) if not os.path.isfile(img\_path): continue ann\_ids = coco.getAnnIds(imgIds=img\_info['id']) anns = coco.loadAnns(ann\_ids) for ann in anns: data.append({ 'image\_id': img\_info['id'], 'file\_name': img\_info['file\_name'], 'category\_id': ann['category\_id'], 'bbox': ann['bbox'] }) df = pd.DataFrame(data)

* Преобразование загруженных данных и аннотаций в DataFrame для удобного анализа и обработки с помощью **pandas**.

**Первичный анализ данных**

python

Копировать код

df\_description = df.describe(include='all') print(df\_description)

* Описание DataFrame, включающее статистическую сводку по данным.

**Визуализация и сохранение метрик данных**

python

Копировать код

fig, ax = plt.subplots(figsize=(10, 6)) ax.axis('off') ax.text(0.5, 0.5, df\_description.to\_string(), horizontalalignment='center', verticalalignment='center', fontsize=12) fig.savefig(os.path.join(screenshot\_dir, 'data\_metrics.png'))

* Визуализация и сохранение метрик данных в виде скриншота.

**Проверка на наличие пропущенных значений и их визуализация**

python

Копировать код

missing\_data = df.isna().sum() print(missing\_data) fig, ax = plt.subplots(figsize=(10, 6)) ax.axis('off') ax.text(0.5, 0.5, missing\_data.to\_string(), horizontalalignment='center', verticalalignment='center', fontsize=12) fig.savefig(os.path.join(screenshot\_dir, 'missing\_data.png'))

* Проверка на наличие пропущенных значений в данных и визуализация результата.

**Очистка данных**

python

Копировать код

df\_cleaned = df.dropna() df\_cleaned\_description = df\_cleaned.describe(include='all') fig, ax = plt.subplots(figsize=(10, 6)) ax.axis('off') ax.text(0.5, 0.5, df\_cleaned\_description.to\_string(), horizontalalignment='center', verticalalignment='center', fontsize=12) fig.savefig(os.path.join(screenshot\_dir, 'cleaned\_data\_metrics.png')) plt.show()

* Очистка данных путем удаления строк с пропущенными значениями и визуализация очищенных данных.

**Разделение данных на обучающую и тестовую выборки**

python

Копировать код

X\_train, X\_test, y\_train, y\_test = train\_test\_split(train\_images, train\_labels\_flat, test\_size=0.2, random\_state=42)

* Разделение данных на обучающую и тестовую выборки с использованием **train\_test\_split**.

**Визуализация структуры моделей**

**Визуализация структуры Random Forest**

python

Копировать код

def plot\_random\_forest\_structure(filename): fig, ax = plt.subplots(figsize=(10, 6)) ax.set\_title('Структура Random Forest') ax.text(0.5, 0.5, 'Random Forest\nс множеством деревьев решений', horizontalalignment='center', verticalalignment='center', fontsize=12, bbox=dict(facecolor='white', alpha=0.5)) ax.axis('off') plt.savefig(filename) plt.close(fig)

* Функция для визуализации структуры Random Forest.

**Визуализация структуры Gradient Boosting**

python

Копировать код

def plot\_gradient\_boosting\_structure(filename): fig, ax = plt.subplots(figsize=(10, 6)) ax.set\_title('Структура Gradient Boosting') ax.text(0.5, 0.5, 'Gradient Boosting\nс последовательными деревьями решений', horizontalalignment='center', verticalalignment='center', fontsize=12, bbox=dict(facecolor='white', alpha=0.5)) ax.axis('off') plt.savefig(filename) plt.close(fig)

* Функция для визуализации структуры Gradient Boosting.

**Визуализация структуры Stacking**

python

Копировать код

def plot\_stacking\_structure(filename): fig, ax = plt.subplots(figsize=(10, 6)) ax.set\_title('Структура Stacking') layers = [ "Random Forest (базовый алгоритм)", "Gradient Boosting (базовый алгоритм)", "Logistic Regression (финальный алгоритм)" ] for i, layer in enumerate(layers): ax.text(0.5, 1 - (i \* 0.3), layer, horizontalalignment='center', verticalalignment='center', fontsize=12, bbox=dict(facecolor='white', alpha=0.5)) ax.axis('off') plt.savefig(filename) plt.close(fig)

* Функция для визуализации структуры Stacking.

**Визуализация точности моделей**

python

Копировать код

plt.figure(figsize=(10, 6)) for (name, accuracy, \_), color in zip(model\_accuracies, colors): if accuracy in model\_colors: model\_colors[accuracy].append(name) else: model\_colors[accuracy] = [name] new\_colors = {} for idx, (accuracy, names) in enumerate(model\_colors.items()): if len(names) > 1: new\_color = plt.cm.hsv(idx / len(model\_colors)) new\_colors[new\_color] = names plt.plot(range(len(X\_test)), [accuracy] \* len(X\_test), color=new\_color, label=', '.join(names)) else: plt.plot(range(len(X\_test)), [accuracy] \* len(X\_test), color=colors[idx], label=names[0]) plt.xlabel('Итерации') plt.ylabel('Точность') plt.title('Точность предсказаний моделей на тестовом наборе данных') plt.legend(loc='best') plt.show()

* Создание и отображение графика точности предсказаний моделей на тестовом наборе данных.

Этот код отличается от предыдущего дополнительными этапами визуализации, анализа и обработки данных, а также использованием **pandas** для более удобной работы с данными и их анализа.