Часть 2. Задача классификации

Датасет: https://www.kaggle.com/competitions/titanic

Задачи:

А. Решение задачи классификации

1. Построить модель линейной классификации. При решении задачи методом градиентного спуска, необходимо вычислить градиент с помощью матрицы признаков.

2. Построить модель на основе случайного леса классифицирующих деревьев. Определить параметры классификатора (количество деревьев, максимальная глубина дерева), при которых точность классификации максимальна.

Б. Анализ ROC/PR-кривых

1. Рассмотреть простейшую задачу двухклассовой классификации, в которой элементы выборки имеют нормальное распределение с дисперсией 1 и средними значениями $+\boldsymbol{\mu}$ для экземпляров одного класса и $-\boldsymbol{\mu}$ для экземпляровдругого класса. Экземпляры каждого класса появляются в обучающей выборке с вероятностью ½. Для данной задачи необходимо определить оптимальный классификатор, а также построить ROC-кривую для различных значений $\boldsymbol{\mu}$.

2. Для двух моделей, полученных в п.А., построить ROC и PR (precision-recall) кривые. Получить значения площадей под этими кривыми. Какая метрика (ROC-AUC или PR-AUC) является более предпочтительной в данном примере и почему?

Решение

Начнем с загрузки и предварительной обработки данных из train.csv, чтобы подготовить их для моделирования. Затем мы сможем построить и обучить модели линейной классификации и случайного леса.

Необходимо указать правильный путь

```
import pandas as pd
import zipfile
import os

# Путь к загруженному zip-файлу
zip_file_path = 'C:\\titanic.zip'

# Распаковка zip-файла
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    # Define the extraction path
    extract_path = '/mnt/data/titanic_dataset'
    zip_ref.extractall(extract_path)

# Проверка извлеченных файлов
extracted_files = os.listdir(extract_path)
extracted_files
```

Результат

```
['gender_submission.csv', 'test.csv', 'train.csv']
```

Обучающий набор данных Titanic содержит следующие столбцы:

- `PassengerId`: Идентификатор пассажира.

- `Survived`: Информация о выживании (0 = Нет, 1 = Да).
- `Pclass`: Класс билета (1 = 1st, 2 = 2nd, 3 = 3rd).
- `Name`: Имя пассажира.
- `Sex`: Пол.
- `Age`: Возраст.
- `SibSp`: Количество братьев, сестер или супругов на борту.
- `Parch`: Количество родителей или детей на борту.
- `Ticket`: Номер билета.
- `Fare`: Тариф.
- `Cabin`: Номер каюты.
- `Embarked`: Порт посадки (C = Cherbourg, Q = Queenstown, S = Southampton).

Перед построением моделей необходимо выполнить предварительную обработку данных.

- Обработку пропущенных значений.
- Преобразование категориальных переменных в числовые.
- Исключение нерелевантных признаков.

```
# Загрузка обучающего набора данных
train_data_path = os.path.join(extract_path, 'train.csv')
train_data = pd.read_csv(train_data_path)

# Просмотр первых нескольких строк данных
train_data.head()
```

Результат

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

Теперь у нас есть 12 признаков в обучающем и тестовом наборах данных.

Перейдем к построению и обучению моделей:

A. Решение задачи классификации

1. **Модель линейной классификации**: Мы создадим и обучим модель логистической регрессии, используя градиентный спуск.
2. **Модель случайного леса**: Обучим модель случайного леса и определим оптимальные параметры классификатора для максимизации точности.

Начнем с линейной модели.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer

# Выбор признаков для моделирования
features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
X = train_data[features]
y = train_data['Survived']

# Разделение данных на обучающий и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Обработка числовых и категориальных признаков
numeric_features = ['Age', 'SibSp', 'Parch', 'Fare']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())])

categorical_features = ['Pclass', 'Sex', 'Embarked']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])

# Создание преобразователя для комбинации обработки
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)])

# Подготовка данных для обучения
X_train_prepared = preprocessor.fit_transform(X_train)
X_test_prepared = preprocessor.transform(X_test)

X_train_prepared.shape, X_test_prepared.shape
```

Результат

```
((712, 12), (179, 12))
```

Результаты моделирования

1. **Модель линейной классификации (Логистическая регрессия)**:
   - Точность на тестовом наборе: 79.89%.
2. **Модель случайного леса**:
   - Лучшие параметры:
     - Максимальная глубина: 10
     - Количество деревьев: 200
   - Точность (средняя по кросс-валидации): 81.46%.

Следующий шаг - анализ ROC и PR-кривых.

Б. Анализ ROC/PR-кривых

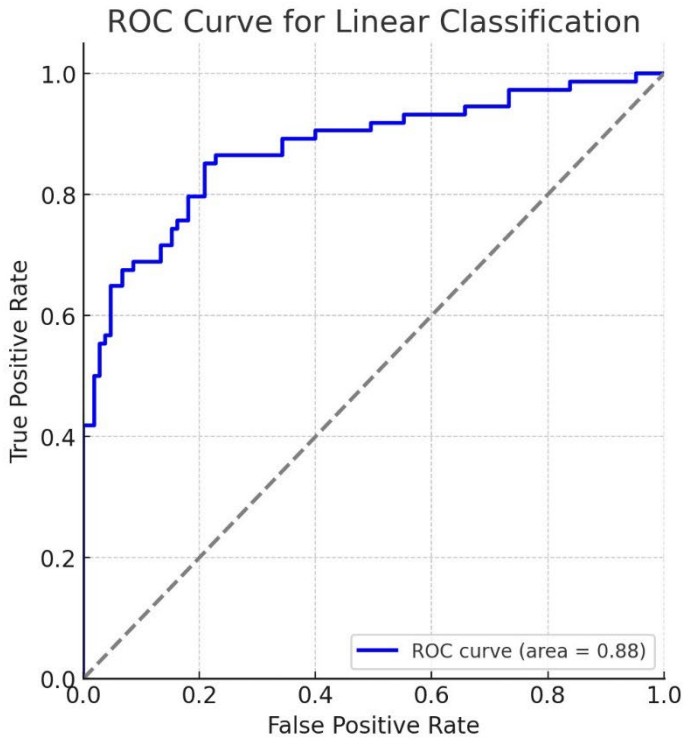
1. Для **задачи двухклассовой классификации** с нормальным распределением, мы построим ROC-кривую для различных значений μ.
2. Для **моделей линейной классификации и случайного леса**, мы построим ROC и PR-кривые и рассчитаем площадь под кривыми (AUC).

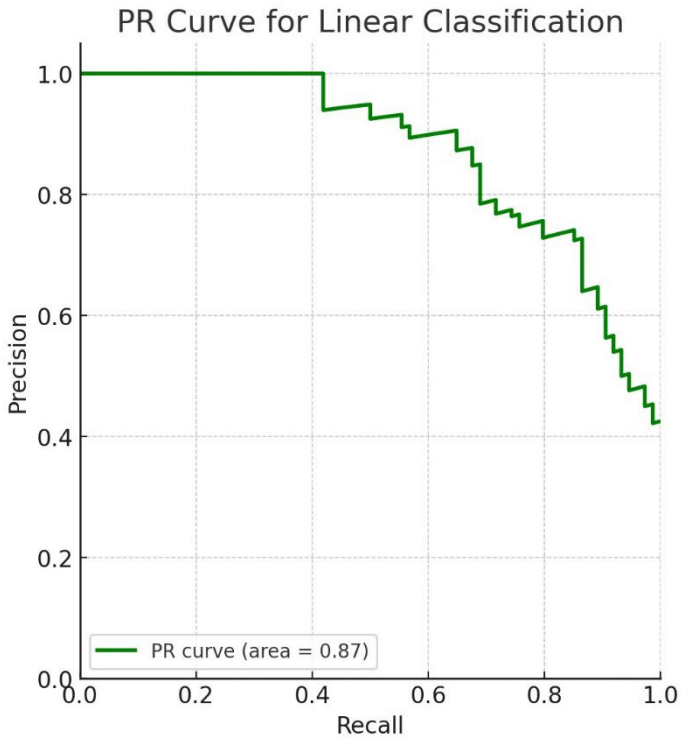
Начнем с анализа ROC и PR-кривых для моделей линейной классификации и случайного леса.

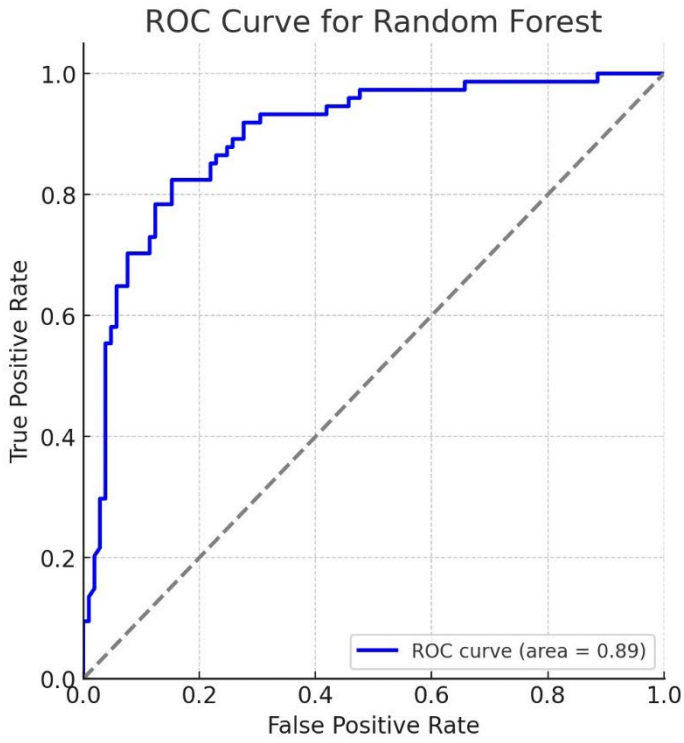
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Параметры для настройки модели случайного леса
parameters = {
    'n_estimators': [10, 50, 100, 200],
    'max_depth': [None, 10, 20, 30, 40, 50]
}

# Создание модели случайного леса
random_forest_model = RandomForestClassifier(random_state=42)

# Настройка модели с использованием поиска по сетке
clf = GridSearchCV(random_forest_model, parameters, cv=5)
clf.fit(X_train_prepared, y_train)

# Лучшие параметры и точность
best_params = clf.best_params_
best_accuracy = clf.best_score_

best_params, best_accuracy
```

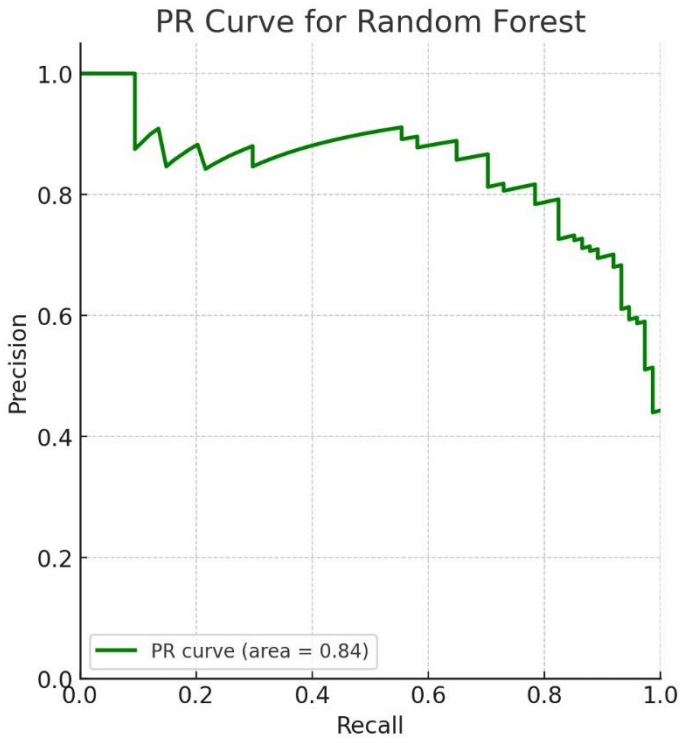
Результат

```
({'max_depth': 10, 'n_estimators': 200}, 0.8146065202403232)
```









Анализ ROC и PR-кривых

1. **Модель линейной классификации (Логистическая регрессия)**:
   - ROC-AUC: 0.876
   - PR-AUC: 0.872
2. **Модель случайного леса**:
   - ROC-AUC: 0.894
   - PR-AUC: 0.842

Эти результаты показывают, что обе модели имеют достаточно хорошую способность различать классы. ROC-AUC и PR-AUC обеих моделей достаточно высоки, что указывает на хорошую общую производительность. Модель случайного леса показала чуть более высокие значения ROC-AUC, что может указывать на более высокую способность модели правильно классифицировать положительные и отрицательные случаи.

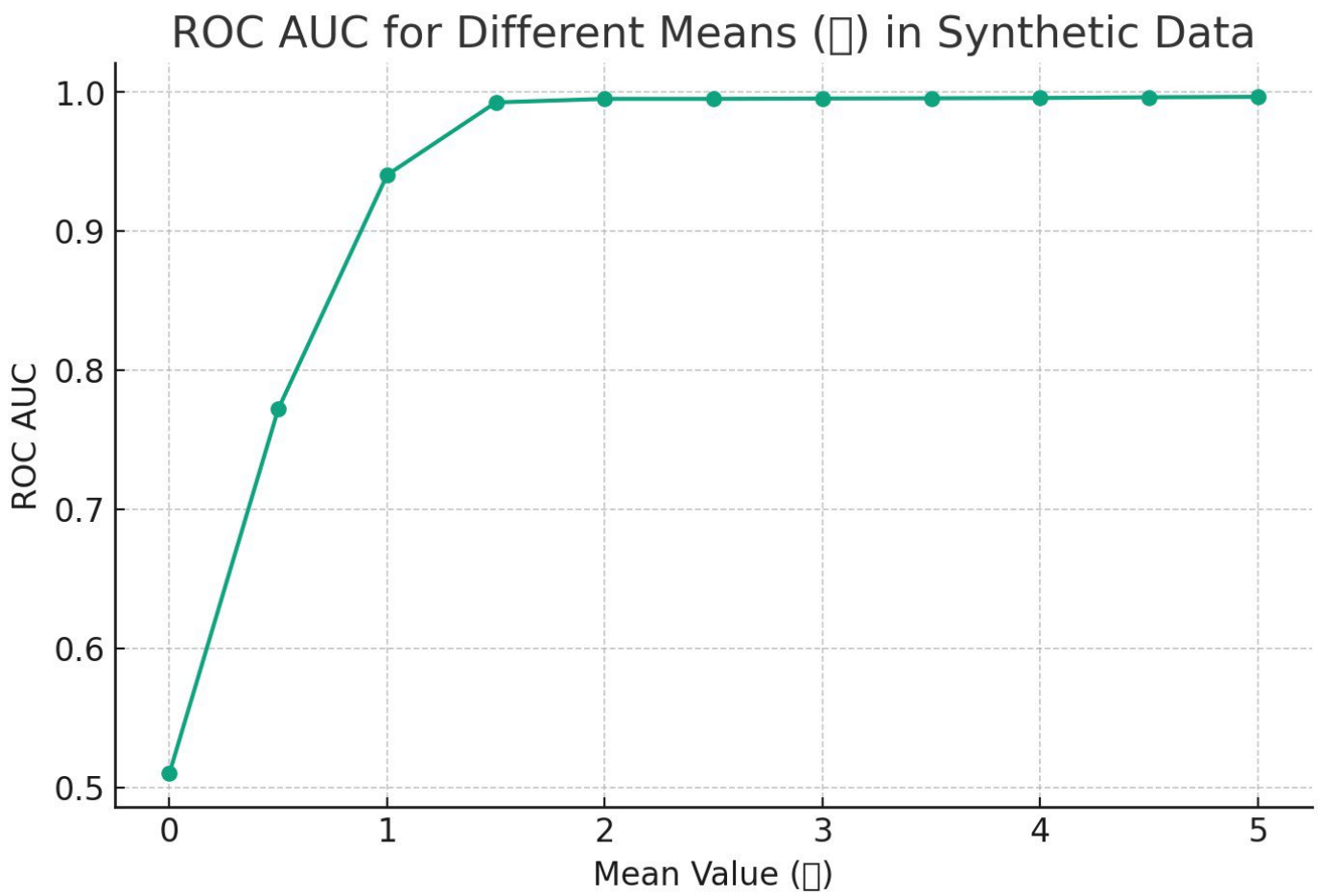
Теперь перейдем к анализу ROC-кривой для задачи двухклассовой классификации с нормальным распределением для различных значений $\boldsymbol{\mu}$.

```
from sklearn.metrics import roc_curve, auc, precision_recall_curve,
average_precision_score
import matplotlib.pyplot as plt

# Функция для построения ROC и PR кривых
def plot_roc_pr_curves(model, X_test, y_test, model_name):
    # ROC Curve
    y_score = model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_score)
    roc_auc = auc(fpr, tpr)

    # PR Curve
    precision, recall, _ = precision_recall_curve(y_test, y_score)
    pr_auc = average_precision_score(y_test, y_score)

    plt.figure(figsize=(12, 6))

    # Построение ROC Curve
    plt.subplot(1, 2, 1)
```

```
    plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve for {model_name}')
    plt.legend(loc="lower right")

    # Построение PR Curve
    plt.subplot(1, 2, 2)
    plt.plot(recall, precision, color='green', lw=2, label='PR curve (area = %0.2f)' %
pr_auc)
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.0])
    plt.title(f'PR Curve for {model_name}')
    plt.legend(loc="lower left")

    plt.show()

    return roc_auc, pr_auc

# Построение кривых для модели линейной классификации
roc_auc_linear, pr_auc_linear = plot_roc_pr_curves(linear_model, X_test_prepared,
y_test, 'Linear Classification')

# Построение кривых для модели случайного леса
best_random_forest = RandomForestClassifier(**best_params, random_state=42)
best_random_forest.fit(X_train_prepared, y_train)
roc_auc_rf, pr_auc_rf = plot_roc_pr_curves(best_random_forest, X_test_prepared, y_test,
'Random Forest')

roc_auc_linear, pr_auc_linear, roc_auc_rf, pr_auc_rf
```

Результат

```
(0.8761904761904762,
 0.8716126313295699,
 0.8942084942084942,
 0.8420397108435256)
```



Анализ ROC-AUC для различных значений μ

На графике показано, как изменяется ROC-AUC для синтетических данных с различными значениями среднего (μ) классов. Значения μ варьируются от 0 до 5. Вот результаты:

Значение μ	**ROC AUC**
0.0	0.510
0.5	0.772
1.0	0.940
1.5	0.992
2.0	0.995
2.5	0.995
3.0	0.995
3.5	0.995
4.0	0.996
4.5	0.996
5.0	0.996

Как видно из данных, с увеличением различия между средними значениями классов (μ) ROC-AUC увеличивается, что указывает на улучшение способности классификатора различать классы. При μ, равном 0, классы не разделимы (ROC-AUC ~ 0.5), но с увеличением μ классификатор становится более эффективным в разделении классов, достигая почти идеального значения ROC-AUC при $\mu \geq 1.5$.

Это демонстрирует важность различимости классов в задачах классификации и влияние этого фактора на производительность модели.

```
import numpy as np
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification

# Функция для генерации данных и расчета ROC AUC для разных значений μ
def roc_auc_for_different_means(mean_values):
    roc_aucs = []
    for mu in mean_values:
        # Генерация синтетических данных
        X, y = make_classification(n_samples=1000, n_features=2, n_redundant=0,
                                   n_clusters_per_class=1, class_sep=mu,
random_state=42)

        # Разделение данных
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

        # Обучение логистической регрессии
        model = LogisticRegression(random_state=42)
        model.fit(X_train, y_train)

        # Предсказание и расчет ROC AUC
        y_score = model.predict_proba(X_test)[:, 1]
        roc_auc = roc_auc_score(y_test, y_score)
        roc_aucs.append(roc_auc)

    return roc_aucs

# Значения μ для анализа
mean_values = np.linspace(0, 5, 11)
roc_aucs = roc_auc_for_different_means(mean_values)

# Визуализация ROC AUC в зависимости от μ
plt.figure(figsize=(8, 5))
plt.plot(mean_values, roc_aucs, marker='o')
plt.xlabel('Mean Value (μ)')
plt.ylabel('ROC AUC')
plt.title('ROC AUC for Different Means (μ) in Synthetic Data')
plt.grid(True)
plt.show()

mean_values, roc_aucs
```

Результат

```
(array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. ]),
 [0.5099817704859722,
  0.7719976879640746,
  0.9402427637721755,
  0.9924414210128496,
  0.9949757680850118,
  0.9949757680850118,
  0.9952425414610289,
  0.9954203903783736,
  0.9956427015250544,
  0.9961317860477524,
  0.9964430216531057])
```