

Лабораторная работа 1.

Прогнозирование задержки рейсов путем создания модели машинного обучения в Python

Введение

Python является одним из наиболее популярных языков программирования в мире. Он широко используется в сообществе специалистов по обработке и анализу данных для машинного обучения и статистического анализа. Одна из причин его популярности — наличие тысяч библиотек с открытым кодом, таких как NumPy, Pandas, Matplotlib и scikit-learn, которые позволяют программистам и аналитикам изучать, преобразовывать, анализировать и визуализировать данные.

В этой лабораторной работе мы создадим записную книжку Jupiter Notebook, импортируем набор данных, содержащий сведения о своевременном прибытии рейсов крупной авиакомпании США, и загрузим набор данных в записную книжку. Затем мы очистим набор данных с помощью Pandas, создадим модель машинного обучения с помощью scikit-learn и визуализируем ее выходные данные с помощью Matplotlib.

Цели лабораторной работы. Получить практические навыки подготовки данных для машинного обучения, использования scikit-learn для создания модели машинного обучения и для визуализации эффективности модели с помощью Matplotlib

Создание записной книжки Jupyter Notebook и импорт данных

Создадим новый проект в Jupyter Notebook.

1. В первой ячейке записной книжки введем команда Bash – *curl*. В записной книжке Jupyter можно выполнять команды Bash, ставя перед ними восклицательный знак. Эта команда загружает CSV-File из хранилища BLOB-объектов Jupyter Notebook и сохраняет его под именем *flightdata.csv*. Выполним команду *curl* (рис. 1.1).

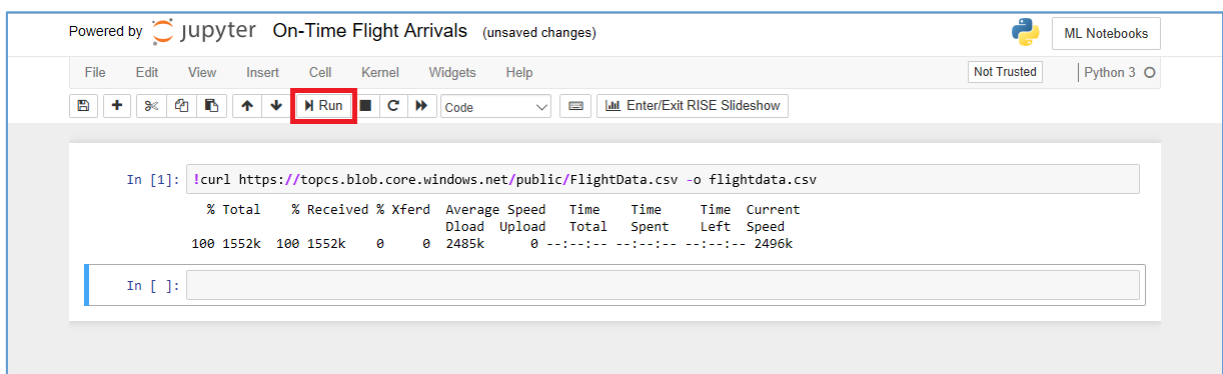
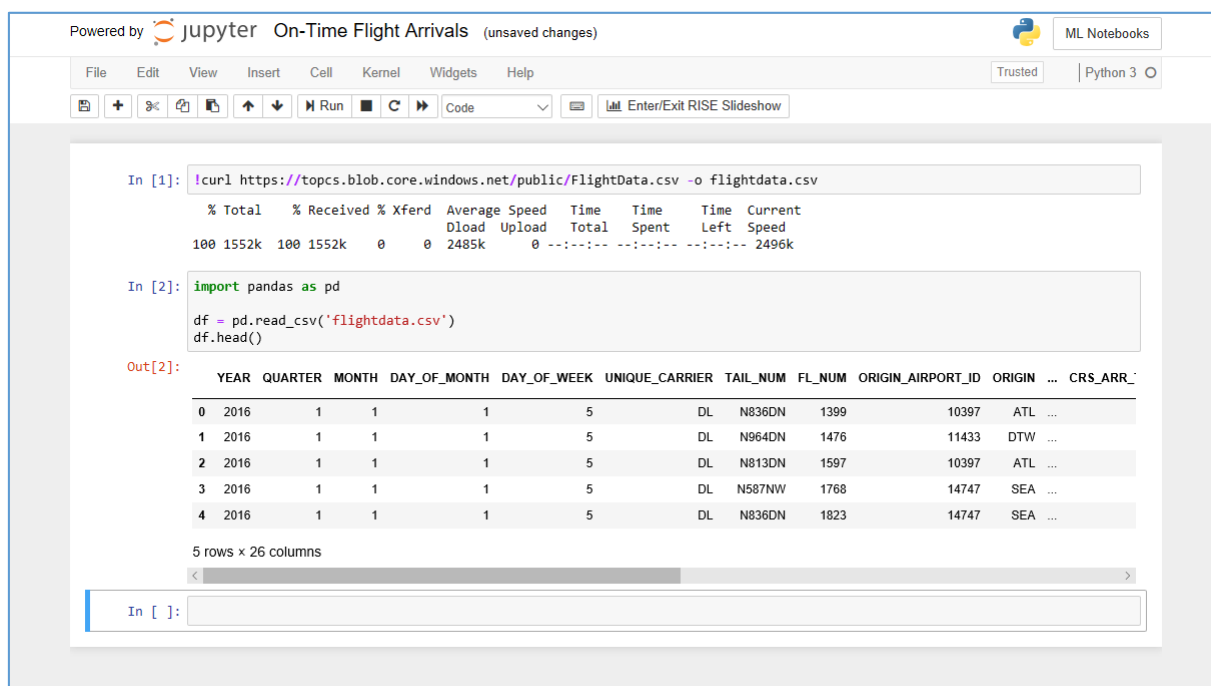


Рисунок 1.1 – Импорт набора данных

2. Загружаем данные из файла `flightdata.csv`, создаем `DataFrame` (кадр данных) `Pandas` и выводим первые пять строк (рис. 1.2).



The screenshot shows a Jupyter Notebook titled "On-Time Flight Arrivals" with the following code and output:

```
In [1]: !curl https://topcs.blob.core.windows.net/public/FlightData.csv -o flightdata.csv
```

Below the code, there is a progress bar showing the download status of the file.

```
In [2]: import pandas as pd
df = pd.read_csv('flightdata.csv')
df.head()
```

The output shows the first five rows of the DataFrame:

	YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	UNIQUE_CARRIER	TAIL_NUM	FL_NUM	ORIGIN_AIRPORT_ID	ORIGIN	...	CRS_ARR
0	2016	1	1	1	5	DL	N836DN	1399	10397	ATL	...	
1	2016	1	1	1	5	DL	N964DN	1476	11433	DTW	...	
2	2016	1	1	1	5	DL	N813DN	1597	10397	ATL	...	
3	2016	1	1	1	5	DL	N587NW	1768	14747	SEA	...	
4	2016	1	1	1	5	DL	N836DN	1823	14747	SEA	...	

Below the table, it says "5 rows x 26 columns".

Рисунок 1.2 – Загрузка набора данных

Созданный кадр данных (`df`) содержит сведения о своевременном прибытии рейсов крупной авиакомпании США. В нем более 11 000 строк и 26 столбцов. (В выходных данных сказано "5 строк", так как функция кадра данных `head` возвращает только первые пять строк.) Каждая строка представляет один рейс и содержит такие сведения, как аэропорт отправления и прибытия, запланированное время отправления и указание на своевременность прибытия.

3. Используйте команду `File -> Save and Checkpoint`, чтобы сохранить записную книжку.

Просмотрите все столбцы в наборе данных, используя полосу горизонтальной прокрутки для прокрутки влево и вправо.

Очистка и подготовка данных

Прежде чем подготовить набор данных, необходимо понять их содержимое и структуру. В предыдущем шаге были импортированы набор данных, содержащий сведения о своевременном прибытии рейсов авиакомпании. Эти данные включали 26 столбцов и тысячи строк. Каждая строка представляла один рейс и содержит такие сведения, как аэропорт отправления и прибытия и запланированное время отправления рейса.

4. Чтобы узнать размеры кадра данных будем использовать свойство `DataFrame` – `shape` (рис. 1.3). В кадре данных `df` содержится 11 231 строку и 26 столбцов:

```
In [3]: # Определение сколько строк содержит набор данных
df.shape

Out[3]: (11231, 26)
```

Рисунок 1.3 – Получение количества строк и столбцов

- Рассмотрим столбцы в наборе данных. Они содержат важную информацию, например, дату рейса (YEAR, MONTH и DAY_OF_MONTH), место вылета и назначения (ORIGIN и DEST), запланированное время вылета и прибытия (CRS_DEP_TIME и CRS_ARR_TIME), разницу между запланированным и фактическим временем прибытия в минутах (ARR_DELAY) и указание, если рейс опоздал не менее чем на 15 минут (ARR_DEL15).

В табл. 1.1 приведен полный список столбцов в наборе данных. Значения времени выражаются в 24-часовом военном формате. Например, 1130 означает 11:30, а 1500 — 15:00.

Таблица 1.1 – Информация о столбцах набора данных

Столбец	Описание
YEAR	Год, когда был совершен рейс
QUARTER	Квартал, когда был совершен рейс (1–4)
MONTH	Месяц, когда был совершен рейс (1–12)
DAY_OF_MONTH	День месяца, когда был совершен рейс (1–31)
DAY_OF_WEEK	День недели, когда был совершен рейс (1 — понедельник, 2 — вторник, и т. д.)
UNIQUE_CARRIER	Код авиакомпании (например, DL)
TAIL_NUM	Хвостовой номер самолета
FL_NUM	Номер рейса
ORIGIN_AIRPORT_ID	Идентификатор аэропорта вылета
ORIGIN	Код аэропорта вылета (ATL, DFW, SEA и т. д.)
DEST_AIRPORT_ID	Идентификатор аэропорта назначения
DEST	Код аэропорта назначения (ATL, DFW, SEA и т. д.)
CRS_DEP_TIME	Запланированное время отправления
DEP_TIME	Фактическое время отправления
DEP_DELAY	Число минут задержки отправления
DEP_DEL15	0 = отправление задержано меньше, чем на 15 минут, 1 = отправление задержано на 15 минут или более
CRS_ARR_TIME	Запланированное время прибытия
ARR_TIME	Фактическое время прибытия
ARR_DELAY	Число минут задержки прибытия
ARR_DEL15	0 = рейс прибыл не более, чем на 15 минут позже, 1 = рейс прибыл позже на 15 минут и более
CANCELLED	0 = рейс не отменен, 1 = рейс отменен
DIVERTED	0 = рейс не перенаправлен, 1 = рейс перенаправлен
CRS_ELAPSED_TIME	Запланированное время рейса в минутах
ACTUAL_ELAPSED_TIME	Фактическое время рейса в минутах
DISTANCE	Расстояние рейса в милях

Набор данных включает примерно равномерное распределение дат на протяжении года, что важно, поскольку рейс из Миннеаполиса с большей вероятностью задержат из-за снегопада в январе, чем в июле. Но этот набор данных не очищен и не готов к использованию.

Одним из самых важных аспектов подготовки набора данных для использования в машинном обучении является выбор столбцов признаков, имеющих отношение к результату, который необходимо спрогнозировать, и отсеивание столбцов, которые не влияют на результат, могут исказить его. Также необходимо исключить отсутствующие значения, удалив строки или столбцы, содержащие их, или заменив их действующими значениями. На следующем этапе подготовки данных мы удалим лишние столбцы и заменим недостающие значения в остальных столбцах.

6. Сначала специалисты по обработке и анализу данных обычно ищут в наборе данных отсутствующие значения. Есть простой способ проверить наличие недостающих значений в Pandas (рис. 1.4). Это метод `isnull().values.any()`.

```
In [4]: # Проверка наличия отсутствующих данных
df.isnull().values.any()

Out[4]: True
```

Рисунок 1.4 – Проверка отсутствующих значений

Результат «True» означает, что в наборе данных есть хотя бы одно недостающее значение.

7. Далее необходимо выяснить, где находятся отсутствующие значения. Для этого используем метод `isnull().sum()`, результат выполнения которого приведен на рис. 1.5. Выходные данные отображают количество отсутствующих значений в каждом столбце.

```
YEAR          0
QUARTER        0
MONTH          0
DAY_OF_MONTH  0
DAY_OF_WEEK    0
UNIQUE_CARRIER 0
TAIL_NUM       0
FL_NUM         0
ORIGIN_AIRPORT_ID 0
ORIGIN         0
DEST_AIRPORT_ID 0
DEST           0
CRS_DEP_TIME   0
DEP_TIME       107
DEP_DELAY      107
DEP_DEL15      107
CRS_ARR_TIME   0
ARR_TIME       115
ARR_DELAY      188
ARR_DEL15      188
CANCELLED      0
DIVERTED       0
CRS_ELAPSED_TIME 0
ACTUAL_ELAPSED_TIME 188
DISTANCE       0
Unnamed: 25    11231
dtype: int64
```

Рисунок 1.5 – Количество отсутствующих значений в каждом столбце

8. 26-й столбец ("Unnamed: 25") содержит 11 231 отсутствующих значений, что равно количеству строк в наборе данных. Данный столбец создан по ошибке, так как импортированный CSV-File содержит запятую в конце каждой строки. Чтобы исключить этот столбец, добавьте следующий код в записную книжку и выполните его:

```
# Исключение столбца Unnamed: 25
df = df.drop('Unnamed: 25', axis=1)
# Кадр данных с удаленным 26-м столбцом - Unnamed: 25
df.isnull().sum()
```

Проверьте выходные данные и убедитесь, что столбец 26 исчез из кадра данных (рис. 1.6).

YEAR	0
QUARTER	0
MONTH	0
DAY_OF_MONTH	0
DAY_OF_WEEK	0
UNIQUE_CARRIER	0
TAIL_NUM	0
FL_NUM	0
ORIGIN_AIRPORT_ID	0
ORIGIN	0
DEST_AIRPORT_ID	0
DEST	0
CRS_DEP_TIME	0
DEP_TIME	107
DEP_DELAY	107
DEP_DEL15	107
CRS_ARR_TIME	0
ARR_TIME	115
ARR_DELAY	188
ARR_DEL15	188
CANCELLED	0
DIVERTED	0
CRS_ELAPSED_TIME	0
ACTUAL_ELAPSED_TIME	188
DISTANCE	0
dtype:	int64

Рисунок 1.6 – Кадр данных с удаленным 26-м столбцом

9. Кадр данных по-прежнему содержит большое количество отсутствующих значений, но некоторые из них не нужны, так как содержащие их столбцы не важны для создаваемой модели данных. Цель этой модели заключается в прогнозировании того, прибудет ли рейс, на который необходимо купить билеты, вовремя. Если прогнозируется опоздание рейса, то можно выбрать другой рейс.

Для исключения из модели столбцов, которые не относятся к модели прогнозирования, необходимо отфильтровать набор данных. Например, хвостовой номер самолета вряд ли влияет на своевременность прибытия рейса. А вот запланированное время отправления значительно влияет на своевременность прибытия. Поскольку

большинство авиакомпаний используют звездообразную сеть, утренние рейсы отбывают вовремя чаще, чем дневные и вечерние. В некоторых крупных аэропортах трафик накапливается в течение дня, и вероятность задержки более поздних рейсов возрастает.

Pandas предоставляет простой способ фильтровать ненужные столбцы. Выполните следующий код в новой ячейке в записной книжке:

```
# Фильтрация данных, имеющих отношение к модели.
# Оставляем только те столбцы, которые слиют на вызов модели - определяет аналитик
df = df[["MONTH", "DAY_OF_MONTH", "DAY_OF_WEEK", "ORIGIN", "DEST", "CRS_DEP_TIME", "ARR_DEL15"]]
# Отфильтрованный кадр данных
df.isnull().sum()
```

Выходные данные показывают, что кадр данных теперь включает только столбцы, имеющие отношение к модели, и недостающих значений теперь гораздо меньше (рис. 1.7).

```
MONTH          0
DAY_OF_MONTH   0
DAY_OF_WEEK    0
ORIGIN         0
DEST           0
CRS_DEP_TIME   0
ARR_DEL15      188
dtype: int64
```

Рисунок 1.7 – Отфильтрованный кадр данных

10. Единственным столбцом, который теперь содержит отсутствующие значения, является столбец ARR_DEL15, где 0 означает рейсы, прибывшие вовремя, а 1 — рейсы с задержкой. Чтобы отобразить первые пять строк с недостающими значениями, используйте следующий код:

```
# Строки с отсутствующими значениями - первые 5
df[df.isnull().values.any(axis=1)].head()
```

Pandas представляет отсутствующие значения с помощью NaN, что означает *Not a Number* (не является числом). Выходные данные показывают, что эти строки действительно не имеют значений в столбце ARR_DEL15 (рис. 1.8).

	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_DEP_TIME	ARR_DEL15
177	1	9	6	MSP	SEA	701	NaN
179	1	10	7	MSP	DTW	1348	NaN
184	1	10	7	MSP	DTW	625	NaN
210	1	10	7	DTW	MSP	1200	NaN
478	1	22	5	SEA	JFK	2305	NaN

Рисунок 1.8 – Строки с отсутствующими значениями

11. Эти строки не содержат значений ARR_DEL15, потому что соответствуют отмененным или перенаправленным рейсам. Можно вызвать dropna в кадре

данных, чтобы удалить эти строки. Но поскольку рейс, который был отменен или перенаправлен в другой аэропорт, может считаться опоздавшим, давайте используем метод *fillna*, чтобы заменить отсутствующие значения значением 1.

Используйте следующий код, чтобы заменить недостающие значения в столбце ARR_DEL15 на 1 и отобразить строки с 177 по 184:

```
# заменить недостающие значения в столбце ARR_DEL15 на 1
df = df.fillna({'ARR_DEL15': 1})
df.iloc[177:185]
```

Убедитесь, что NaN в строках 177, 179 и 184 были заменены значением 1, указывающим на задержку прибытия (рис. 1.9)

	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_DEP_TIME	ARR_DEL15
177	1	9	6	MSP	SEA	701	1.0
178	1	9	6	DTW	JFK	1527	0.0
179	1	10	7	MSP	DTW	1348	1.0
180	1	10	7	DTW	MSP	1540	0.0
181	1	10	7	JFK	ATL	1325	0.0
182	1	10	7	JFK	ATL	610	0.0
183	1	10	7	JFK	SEA	1615	0.0
184	1	10	7	MSP	DTW	625	1.0

Рисунок 1.9 – NaN заменены на 1

Теперь набор данных очищен: все отсутствующие значения заменены, лишние столбцы удалены. Но работа еще не закончена. Нужно выполнить еще несколько задач, чтобы подготовить набор данных для использования в машинном обучении.

Столбец CRS_DEP_TIME в наборе данных представляет время запланированного отправления. Подробные числа в этом столбце — он содержит более 500 уникальных значений — могут отрицательно повлиять на точность модели машинного обучения. Чтобы исправить эту проблему, можно использовать прием под названием группирование или квантование. Что если разделить каждое число в этом столбце на 100 и округлить результат до ближайшего целого числа? 1030 превратилось бы в 10, 1925 — в 19 и т. д., и осталось бы максимум 24 отдельных значения. Интуитивно это кажется логичным, потому что неважно, отправляется рейс в 10:30 или в 10:40. Но важна разница между 10:30 и 5:30.

Кроме того, столбцы ORIGIN и DEST содержат коды аэропортов, которые представляют категориальные значения в машинном обучении. Эти столбцы должны быть преобразованы в дискретные столбцы, содержащие переменные индикатора, которые иногда называют формальными. Другими словами, столбец ORIGIN, который содержит пять кодов аэропортов, должен быть преобразован в пять столбцов, по одному на каждый аэропорт, и каждый столбец должен содержать значение 1 и 0, указывающее, отбыл ли

рейс из аэропорта, который представляет этот столбец. Столбец DEST должен обрабатываться точно так же.

Чтобы создать столбцы индикатора из столбцов ORIGIN и DEST сгруппируем время отправления в столбце CRS_DEP_TIME и будем использовать метод Pandas get_dummies.

12. Выведите первые пять строк из кадра данных, используя команду df.head().

В столбце CRS_DEP_TIME содержатся значения от 0 до 2359 в военном формате времени (рис. 1.10).

	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_DEP_TIME	ARR_DEL15
0	1	1	5	ATL	SEA	1905	0.0
1	1	1	5	DTW	MSP	1345	0.0
2	1	1	5	ATL	SEA	940	0.0
3	1	1	5	SEA	MSP	819	0.0
4	1	1	5	SEA	DTW	2300	0.0

Рисунок 1.10 – Кадр данных без группирования времени отправления

Используйте следующие инструкции для группирования (квантования) времени отправления:

```
import math
# Группировка времени отправления: каждое число в этом столбце поделить на 100
# и округлить результат до ближайшего целого числа
for index, row in df.iterrows():
    df.loc[index, 'CRS_DEP_TIME'] = math.floor(row['CRS_DEP_TIME'] / 100)
# Кадр данных с группированием времени отправления
df.head()
```

Результат группирования времени отправления приведен на рис. 1.11. Из рисунка видно, что числа в столбце CRS_DEP_TIME теперь попадают в диапазон от 0 до 23:

	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_DEP_TIME	ARR_DEL15
0	1	1	5	ATL	SEA	19	0.0
1	1	1	5	DTW	MSP	13	0.0
2	1	1	5	ATL	SEA	9	0.0
3	1	1	5	SEA	MSP	8	0.0
4	1	1	5	SEA	DTW	23	0.0

Рисунок 1.11 – Кадр данных с группированием времени отправления

13. Теперь используйте следующие инструкции, чтобы создать столбцы индикатора из столбцов ORIGIN и DEST и удалить сами столбцы ORIGIN и DEST:


```
# создать столбцы индикатора из столбцов ORIGIN и DEST и удалить сами столбцы ORIGIN и DEST
# столбец ORIGIN, который содержит пять кодов аэропортов, должен быть преобразован в пять столбцов,
# по одному на каждый аэропорт, и каждый столбец должен содержать значение 1 и 0, указывающее,
# отбыл ли рейс из аэропорта, который представляет этот столбец.
# Столбец DEST должен обрабатываться точно так же
df = pd.get_dummies(df, columns=['ORIGIN', 'DEST'])
df.head()
```

Результат формирования столбцов индикаторов приведен на рис. 1.12. Из рисунка видно, что столбцы ORIGIN и DEST заменены на столбцы, соответствующие кодам аэропортов из исходных столбцов. Новые столбцы имеют значения 1 и 0, обозначающие аэропорт прибытия или отправления рейса.

	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	CRS_DEP_TIME	ARR_DEL15	ORIGIN_ATL	ORIGIN_DTW	ORIGIN_JFK	ORIGIN
0	1	1	5	19	0.0	1	0	0	
1	1	1	5	13	0.0	0	1	0	
2	1	1	5	9	0.0	1	0	0	
3	1	1	5	8	0.0	0	0	0	
4	1	1	5	23	0.0	0	0	0	

Рисунок 1.12 – Кадр данных со столбцами индикатора

14. Используйте команду File -> Save and Checkpoint, чтобы сохранить записную книжку.

Набор данных теперь существенно отличается от исходного, он оптимизирован для использования в машинном обучении.

Создание модели машинного обучения

Чтобы создать модель машинного обучения, необходимо иметь два набора данных: один для обучения и один — для тестирования. Мы скачали из файла flightdata.csv один набор данных, поэтому его необходимо разделить на две части. Разделим подготовленный ранее кадр данных в соотношении 80 к 20, чтобы его можно было использовать для обучения модели машинного обучения. Также необходимо разделить кадр данных на столбцы признаков и столбцы меток. Столбцы признаков используются в качестве входных данных для модели (например, аэропорт отправления и прибытия и запланированное время отправления), а столбцы меток содержат значения, которые модель попытается предсказать, — в данном случае это столбец ARR_DEL15, который указывает, прибудет ли рейс вовремя.

16. В новой ячейке в конце записной книжки введите и запустите следующие инструкции:

```
# Формирование выборок для обучения (80%) и тестирования(20%) и
# задание столбцов признаков и столбца меток.
from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y =
train_test_split(df.drop('ARR_DEL15', axis=1), df['ARR_DEL15'],
                 test_size=0.2, random_state=42)
```

Первая инструкция импортирует вспомогательную функцию `scikit-learn train_test_split`. Во второй строке указана функция для разделения кадра данных на обучающий набор, содержащий 80 % исходных данных, и набор тестирования, содержащий оставшиеся 20 % (параметр `test_size=0.2`). Параметр `random_state` заполняет генератор случайных чисел, используемый для разделения, а первый и второй параметры — это кадры данных, содержащие столбцы признаков и столбцы меток.

17. `train_test_split` возвращает четыре кадра данных. Используйте следующую команду для отображения числа строк и столбцов в кадре данных, содержащем столбцы признаков, используемые для обучения: `train_x.shape`.

18. Используйте следующую команду для отображения числа строк и столбцов в кадре данных, содержащем столбцы признаков, используемые для тестирования: `test_x.shape`

Чем отличаются два результата и почему?

Можно ли предсказать, что вы увидите, если вызовете `shape` в других двух кадрах данных, `train_y` и `test_y`? Если вы не знаете точно, попробуйте и узнаете.

Для прогнозирования опоздания рейсов целесообразно построить модель двоичной классификации, которая будет прогнозировать, прибудет ли рейс вовремя или с опозданием.

Одним из преимуществ использования `scikit-learn` является то, что вам не нужно создавать эти модели или реализовать используемые в них алгоритмы вручную. Библиотека `scikit-learn` включает различные классы для реализации распространенных моделей машинного обучения. Один из них — `RandomForestClassifier`, который сопоставляет множество деревьев принятия решений с данными и вычисляет средние значения, чтобы повысить общую точность и ограничить лжезависимость.

19. Выполните следующий код в новой ячейке, чтобы создать объект `RandomForestClassifier` и обучить его с помощью метода `fit`.

```
# Создание модели обучения RandomForestClassifier и обучение его с помощью метода fit.
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(random_state=13)
model.fit(train_x, train_y)
```

Выходные данные показывают параметры, используемые в классификаторе, включая `n_estimators`, который указывает число деревьев в каждом лесу деревьев принятия решений, и `max_depth`, который указывает максимальную глубину дерева принятия решений. Указанные значения используются по умолчанию, но их можно переопределить при создании объекта `RandomForestClassifier`.

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                        max_depth=None, max_features='auto', max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
                        oob_score=False, random_state=13, verbose=0, warm_start=False)
```

Обучение модели

20. Для тестирования модели необходимо выполнить метод `predict`, используя значения в `test_x`, а затем метод `score`, чтобы определить среднюю точность модели. Результат тестирования приведен на рис. 1.13

```
# Тестирование модели - метод predict. Метод score определяет точность модели
predicted = model.predict(test_x)
model.score(test_x, test_y)

0.8602581219403649
```

Рисунок 1.13 – Результат тестирования модели

Средняя точность составляет 86 %, что, на первый взгляд, кажется хорошим результатом. Но средняя точность не всегда является надежным показателем точности модели классификации. Давайте детальнее проанализируем модель и посмотрим, насколько она точна на самом деле, то есть как хорошо она определяет своевременность прибытия рейсов.

Существует несколько способов для измерения точности модели классификации. Одним из лучших показателей модели двоичной классификации является площадь под ROC-кривой (иногда называется ROC AUC), которая, по сути, предназначена для количественной оценки того, как часто модель делает правильный прогноз независимо от результата.

Вычислим показатель ROC AUC для модели, созданной ранее, и узнаем, почему этот показатель меньше, чем средняя точность результатов по методу `score`.

21. Прежде чем вычислить ROC AUC, необходимо создать *вероятности прогноза* для тестового набора. Эти вероятности являются приблизительными для каждого из классов, или ответов, которые может предсказать модель. Например, `[0.88199435, 0.11800565]` означает, что существует 89 % вероятности того, что рейс прибывает вовремя (`ARR_DEL15 = 0`) и 12 % вероятности того, что он опоздает (`ARR_DEL15 = 1`). Сумма двух вероятностей составляет 100 %.

Выполним следующий код, чтобы создать набор вероятностей прогнозирования на основе тестовых данных:

```
# создать набор вероятностей прогнозирования из тестовых данных
from sklearn.metrics import roc_auc_score
probabilities = model.predict_proba(test_x)
```

22. Выполним следующую инструкцию для создания оценки ROC AUC по вероятностям с помощью метода `scikit-learn roc_auc_score`.

```
# создания оценки ROC AUC из вероятностей с помощью метода Scikit-Learn roc_auc_score
roc_auc_score(test_y, probabilities[:, 1])

0.6743824904998539
```

Результат оценка точности модели составляет 67 %:

Создание оценки AUC

Почему показатель AUC ниже, чем средняя точность, вычисленная в предыдущем примере?

В выходных данных метода `score` отражается, сколько элементов тестового набора модель предсказала правильно. Эта оценка искажена потому, что набор данных, на котором модель обучалась и тестировалась, содержит гораздо больше строк с прибытием вовремя, чем с опозданием. Из-за этого дисбаланса в данных имеется больше шансов, когда предсказывается своевременное прибытие, а не опоздание.

ROC AUC учитывает это и предоставляет более точное обозначение вероятности того, насколько прогноз прибытия вовремя или с опозданием будет правильным.

23. Чтобы узнать больше о поведении модели, создаем *матрицу неточностей*, также известную как матрица ошибок. Матрица неточностей считает количество раз, когда каждый ответ был классифицирован как правильный или неправильный. В частности, она предназначена для оценки количества ложных положительных, ложных отрицательных, истинных отрицательных и истинных положительных результатов. Это важно, ведь если модель двоичной классификации, обученная для различения кошек и собак, тестируется с набором данных, на 95 % состоящим из собак, она может получить оценку 95 %, просто если будет каждый раз отвечать "собака". Но если она не распознает ни одной кошки, она будет бесполезна.

Чтобы создать матрицу неточностей для модели, используйте следующий код:

```
# создать матрицу неточностей для модели
from sklearn.metrics import confusion_matrix
confusion_matrix(test_y, predicted)
```

```
array([[1882,  54],
       [ 260,  51]])
```

Первая строка в выходных данных представляет рейсы, которые прибыли вовремя. Первый столбец в этой строке показывает, сколько рейсов правильно предсказаны как прибывшие вовремя, а второй — сколько рейсов предсказаны как опоздавшие, но на самом деле не опоздали. По этим данным кажется, что модель справляется с прогнозированием своевременного прибытия.

Но посмотрите на вторую строку, где показаны опоздавшие рейсы. В первом столбце показано, сколько опоздавших рейсов были неправильно предсказаны как своевременные. Во втором столбце показано, сколько опоздавших рейсов было предсказано правильно. Очевидно, модель предсказывает задержки гораздо хуже, чем своевременное прибытие. В *идеале* в матрице неточностей должны быть большие числа в левом верхнем и правом нижнем углах и маленькие (желательно 0) в правом верхнем и левом нижнем углах.

Измерение точности

24. Другие меры правильности для модели классификации — *точность* и *отзыв*. Предположим, модели предоставлено три своевременных и три задержанных рейса, и она правильно спрогнозировала два своевременных, но не угадала с двумя задержанными. В этом случае точность модели будет равна 50 % (два из четырех рейсов были классифицированы как своевременные, и действительно прибыли вовремя), тогда как отзыв будет составлять 67 % (правильно определила два из трех своевременных рейса). Дополнительные сведения о точности и полноте можно получить тут: https://en.wikipedia.org/wiki/Precision_and_recall.

Библиотека `scikit-learn` содержит удобный метод `precision_score` для вычисления точности. Чтобы измерить точность модели, выполните следующие инструкции:

```
# метод precision_score для вычисления точности
from sklearn.metrics import precision_score

train_predictions = model.predict(train_x)
precision_score(train_y, train_predictions)

0.9972375690607734
```

25. Библиотека `scikit-learn` также содержит метод `recall_score` для вычисления полноты. Чтобы измерить полноту модели, запустите следующие инструкции: Python

```
# метод recall_score для вычисления отзыва
from sklearn.metrics import recall_score

recall_score(train_y, train_predictions)

0.8650159744408946
```

26. Используйте команду `File -> Save and Checkpoint`, чтобы сохранить записную книжку.

В реальном мире специалист по обработке обучающих данных будет искать способы повысить точность модели. Например, можно попробовать разные алгоритмы и *настроить* выбранный алгоритм, чтобы найти оптимальное сочетание параметров. Кроме того, можно расширить набор данных до миллионов, а не нескольких тысяч строк, и попытаться сократить дисбаланс между своевременными и опоздавшими рейсами. Но для наших целей модель подходит без изменений.

Визуализация выходных данных модели

На этом этапе мы импортируем библиотеку `Matplotlib` в записную книжку и настроим её для поддержки встроенных выходных данных `Matplotlib`.

27. В новой ячейке в конце записной книжки выполните следующие инструкции. Игнорируйте все предупреждающие сообщения, относящиеся к кэшированию шрифта:

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

sns.set()
```

Первая инструкция является одной из нескольких команд, которая позволяет Jupyter подготовить выходные данные Matplotlib к просмотру в записной книжке без повторных вызовов команды `show`. И она должна находиться перед всеми ссылками на саму библиотеку Matplotlib. Последняя инструкция настраивает Seaborn для улучшения выходных данных Matplotlib.

28. Чтобы посмотреть на Matplotlib в действии, выполните следующий код в новой ячейке, чтобы построить ROC-кривую для модели машинного обучения, созданной в предыдущем задании:

```
# выполните следующий код в новой ячейке, чтобы построить ROC-кривую
# для модели машинного обучения, созданной в предыдущем задании
from sklearn.metrics import roc_curve

fpr, tpr, _ = roc_curve(test_y, probabilities[:, 1])
plt.plot(fpr, tpr)
plt.plot([0, 1], [0, 1], color='grey', lw=1, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

Text(0, 0.5, 'True Positive Rate')
```

Результат визуализации данных представлен на рис. 1.14.

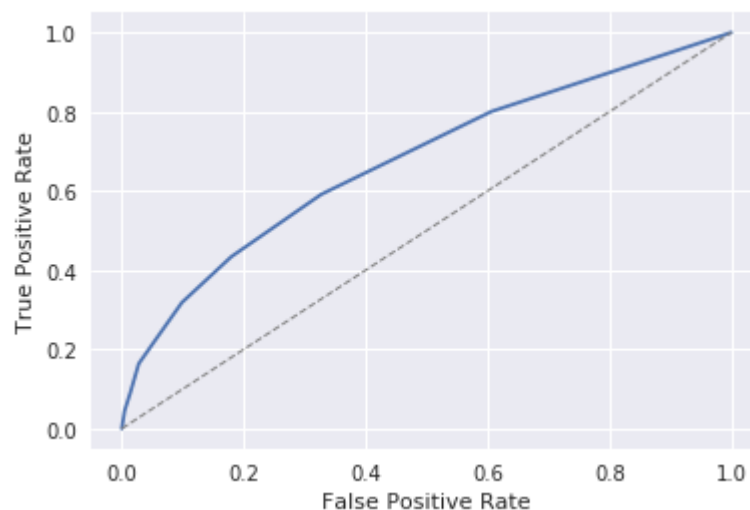


Рисунок 1.14 – ROC-кривая, созданная с помощью Matplotlib

Пунктирная линия в середине диаграммы представляет вероятность получения правильного ответа 50 на 50. Синяя кривая представляет точность модели.

Мы создали эту модель машинного обучения, чтобы предсказывать, прибедет рейс вовремя или опоздает. В этом примере мы напишем функцию Python, которая вызывает модель машинного обучения, созданную в предыдущем примере, для вычисления

вероятности того, что рейс прибудет вовремя. Затем мы используем эту функцию для анализа нескольких рейсов.

29. Введите следующее определение функции в новую ячейку, а затем выполните ячейку.

```
def predict_delay(departure_date_time, origin, destination):
    from datetime import datetime

    try:
        departure_date_time_parsed = datetime.strptime(departure_date_time,
            '%d/%m/%Y %H:%M:%S')
    except ValueError as e:
        return 'Error parsing date/time - {}'.format(e)

    month = departure_date_time_parsed.month
    day = departure_date_time_parsed.day
    day_of_week = departure_date_time_parsed.isoweekday()
    hour = departure_date_time_parsed.hour

    origin = origin.upper()
    destination = destination.upper()

    input = [{'MONTH': month,
              'DAY': day,
              'DAY_OF_WEEK': day_of_week,
              'CRS_DEP_TIME': hour,
              'ORIGIN_ATL': 1 if origin == 'ATL' else 0,
              'ORIGIN_DTW': 1 if origin == 'DTW' else 0,
              'ORIGIN_JFK': 1 if origin == 'JFK' else 0,
              'ORIGIN_MSP': 1 if origin == 'MSP' else 0,
              'ORIGIN_SEA': 1 if origin == 'SEA' else 0,
              'DEST_ATL': 1 if destination == 'ATL' else 0,
              'DEST_DTW': 1 if destination == 'DTW' else 0,
              'DEST_JFK': 1 if destination == 'JFK' else 0,
              'DEST_MSP': 1 if destination == 'MSP' else 0,
              'DEST_SEA': 1 if destination == 'SEA' else 0 }]

    return model.predict_proba(pd.DataFrame(input))[0][0]
```

Эта функция принимает в качестве входных данных дату и время, коды аэропортов отправления и прибытия и возвращает значение от 0,0 до 1,0, определяющее вероятность того, что рейс прибудет вовремя. Она использует модель машинного обучения, которая была создана в предыдущем примере, для вычисления вероятности. И для вызова модели она передает кадр данных, содержащий входные значения, в predict_proba. Структура этого кадра данных точно соответствует структуре кадра данных, который мы использовали ранее.

Примечание

Входные данные даты для функции predict_delay представлены в международном формате dd/mm/year.

30. Используйте приведенный ниже код для вычисления вероятности того, что рейс из Нью-Йорка в Атланту вечером 1 октября прибудет вовремя. Год не имеет значения, так как он не используется в модели.


```
# приведенный ниже код для вычисления вероятности того,  
# что рейс из Нью-Йорка в Атланту вечером 1 октября прибудет вовремя  
predict_delay('1/10/2018 21:45:00', 'JFK', 'ATL')
```

0.6

Вероятность своевременного прибытия составляет 60 %.

31. Изменим код, чтобы вычислить вероятность того, что тот же рейс, но на следующий день, прибудет вовремя:

Какова вероятность того, что этот рейс прибудет вовремя? Если бы вы у вас был гибкий график, вы бы отложили перелет на один день?

32. Теперь измените код, чтобы вычислить вероятность того, что утренний рейс из Атланты в Сиэтл в тот же день прибудет вовремя:

```
# вычислить вероятность того, что утренний рейс из Атланты  
# в Сиэтл в тот же день прибудет вовремя  
predict_delay('2/10/2018 10:00:00', 'ATL', 'SEA')
```

1.0

Вероятность своевременного прибытия составляет 100 %

Теперь у вас есть простой способ прогнозирования своевременности прибытия с помощью одной строки кода. Вы можете поэкспериментировать с другими датами, временем, городами отправления и прибытия. Но имейте в виду, что результаты имеют смысл только для аэропортов с кодами ATL, DTW, JFK, MSP и SEA, так как модель обучалась только на этих кодах.

33. Выполните следующий код, чтобы вычислить вероятность своевременного прибытия вечернего рейса из JFK в ATL в диапазоне дней:

```
# вычислить вероятность своевременного прибытия вечернего рейса из JFK в ATL в диапазоне дней  
import numpy as np  
  
labels = ('Oct 1', 'Oct 2', 'Oct 3', 'Oct 4', 'Oct 5', 'Oct 6', 'Oct 7')  
values = (predict_delay('1/10/2018 21:45:00', 'JFK', 'ATL'),  
          predict_delay('2/10/2018 21:45:00', 'JFK', 'ATL'),  
          predict_delay('3/10/2018 21:45:00', 'JFK', 'ATL'),  
          predict_delay('4/10/2018 21:45:00', 'JFK', 'ATL'),  
          predict_delay('5/10/2018 21:45:00', 'JFK', 'ATL'),  
          predict_delay('6/10/2018 21:45:00', 'JFK', 'ATL'),  
          predict_delay('7/10/2018 21:45:00', 'JFK', 'ATL'))  
alabels = np.arange(len(labels))  
  
plt.bar(alabels, values, align='center', alpha=0.5)  
plt.xticks(alabels, labels)  
plt.ylabel('Probability of On-Time Arrival')  
plt.ylim((0.0, 1.0))
```

(0.0, 1.0)

Результаты анализа приведены на рис. 1.15.

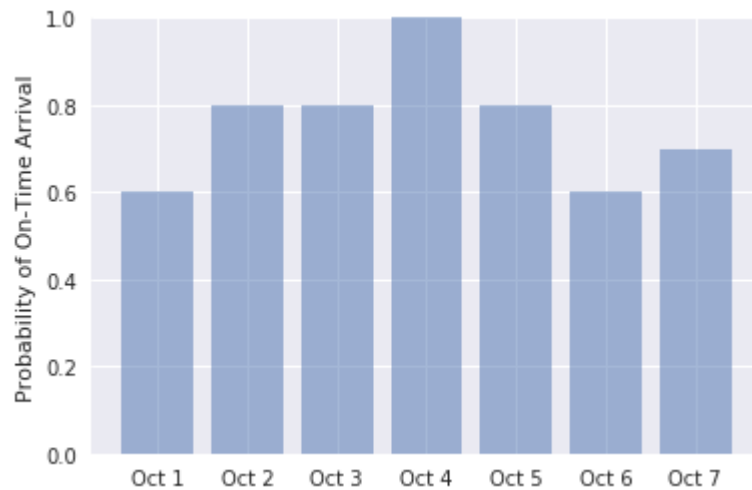


Рисунок 1.15 – Вероятность своевременного прибытия для диапазона дат

34. Измените код, чтобы создать аналогичную диаграмму для рейсов из JFK в MSP в 13:00 с 10 по 16 апреля. Как эти выходные данные соотносятся с выходными данными из предыдущего шага?
35. Самостоятельно напишите код, чтобы рассчитать вероятность того, что рейсы из SEA в ATL с отправлением в 9:00, в 12:00, в 15:00, в 18:00 и в 21:00 30 января придут вовремя. Убедитесь, что выходные данные выглядят следующим образом (рис. 1.16):

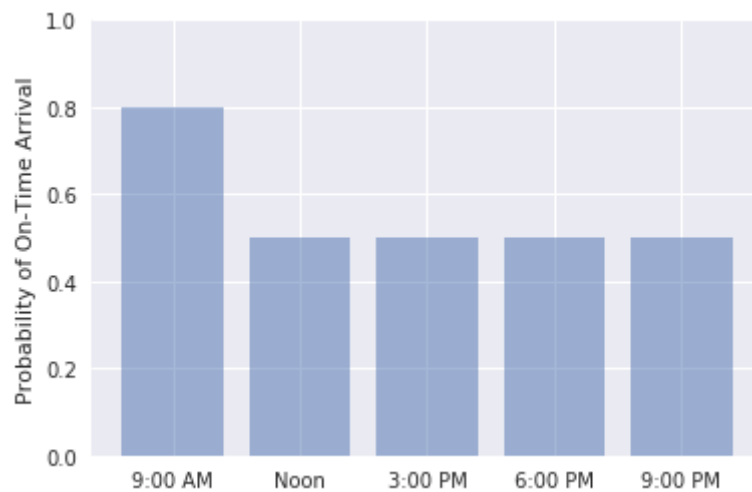


Рисунок 1.16 – Вероятность своевременного прибытия для диапазона времени

