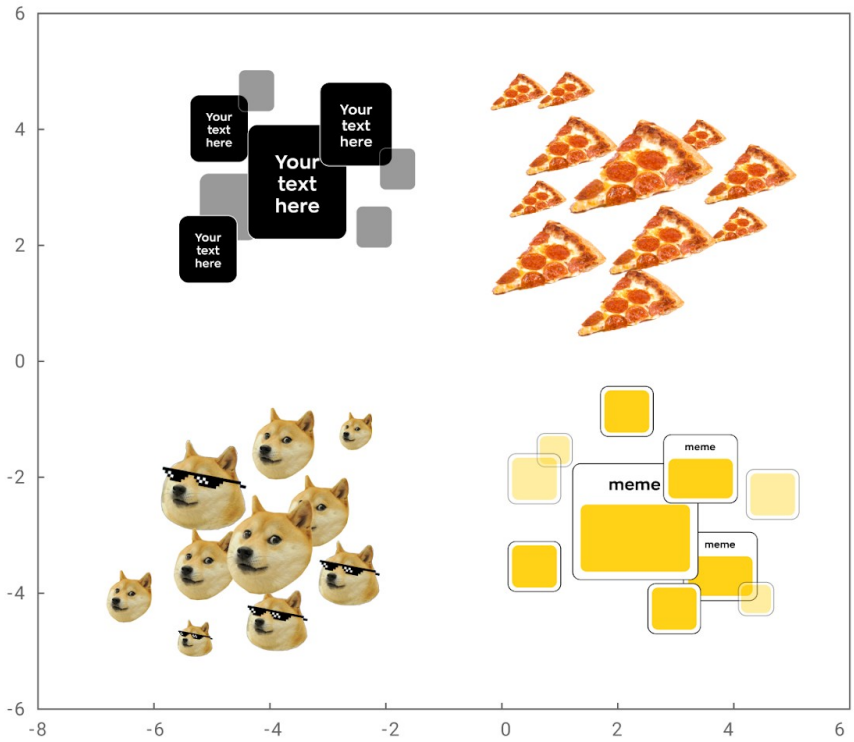
Предсказываем оценку с помощью классификации



В контексте machine learning (ml) классификация относится к обучению с учителем. Она призвана решать задачи по распределению объектов в классы. К примеру, есть конечное множество объектов разделенное некоторым образом на классы, данное множество называется **обучающей выборкой**. Требуется создать такой алгоритм который на основе имеющейся информации был способен определить класс произвольного объекта (количества объектов).

Но бывают разные модели классификации, в частности сегодня познакомимся с методом Random Forest, так называемый «Случайный Лес». Данная модель является продолжением модели «Древо решений».

Модель «Древо решений» постепенно разделяет данные на уменьшающиеся подмножества. Для каждого подмножества свои критерии сортировки. При каждом последующем разделении данных количество критериев сокращается. Классификация будет закончена когда останется только 1 критерий. Объединив несколько таких деревьев, получается «Случайный лес».

В первую очередь необходимо импортировать библиотеки:

- pandas – считывание и предобработка файла;
- numpy – обработка данных;
- sklearn – поможет разработать модели и проанализировать их;
- matplotlib – визуализация данных;
- datetime – анализ скорости выполнения участков кода;
- seaborn – настройка визуализации.

Данные я предпочитаю брать из этого источника. Загрузим данные с оценками кофе и приступим к предобработке данных. С помощью pandas прочитаем csv файл и проанализируем его. В ходе анализа выяснилось что датасет содержит строки с некорректными данными.

	name	roaster	roast	loc_country	origin_1	100g_USD	rating
311	Barichu Double	Barrington Coffee Roasting	Medium-Light	United States	Kenya	11.76	94
874	El Salvador Pacamara	RamsHead Coffee Roasters	Medium-Light	United States	El Salvador	5.28	93
1230	Ethiopia Bekele Heto Natural	Bassline Coffee	Light	United States	Ethiopia	7.41	93
770	Kona SL34 Sweet Spot	Kona Farm Direct	Medium-Light	Hawai'i	Hawai'I	25.17	94
469	Peaberry Medium Roast	Hala Tree Kona Coffee	Medium	Hawai'i	Hawai'I	13.23	92
685	Thailand Robusta Panid Choosit Espresso	Paradise Roasters	Medium	United States	Thailand	4.98	91
963	Natural Ka'u Maragogipe	Big Island Coffee Roasters	Medium-Light	Hawai'i	Hawai'I	16.75	94
943	United States	United States	United States	United States	United States	United States	United States
171	Ethiopia Misti Valley	Durango Coffee Company	Medium	United States	Ethiopia	4.98	93
286	Colombia Pink Bourbon	Paradise Roasters	Medium-Light	United States	Colombia	7.33	94

Очистим датасет и конвертируем типы столбцов в числовой.

```
df = df.query("(rating != 'United States')" or "(100g_USD != 'United States')").astype({'rating':int, '100g_USD':float})
```

Теперь столбцы рейтинг и цена за 100г имеют корректный вид и тип.

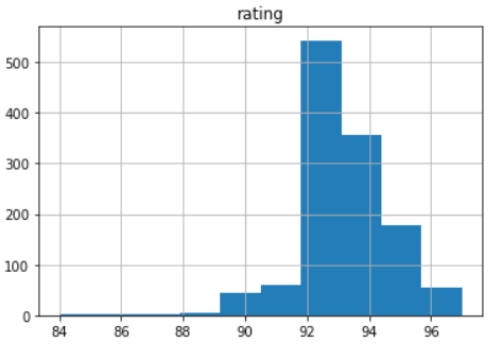
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1246 entries, 0 to 1266
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   name         1246 non-null   object 
 1   roaster      1246 non-null   object 
 2   roast        1234 non-null   object 
 3   loc_country  1246 non-null   object 
 4   origin_1     1246 non-null   object 
 5   100g_USD     1246 non-null   float64
 6   rating       1246 non-null   int32  
dtypes: float64(1), int32(1), object(5)
memory usage: 73.0+ KB
```

Перейдем к разметке данных. Проанализируем распределение оценок рейтинга.

```
# Распределение величин рейтинга
df[['rating']].hist()
```

```
array([[<AxesSubplot:title={'center':'rating'}>]], dtype=object)
```



Оптимальным решением было взять величину, находящуюся посредине (93). Тем сам разделить датасет приблизительно пополам (размер всего датасета — **1246** и часть с рейтингом больше 93 — **588**). Данная операция необходима чтобы при обучении модели не случилась нулевое пересечение классовых меток, когда в тренировочную выборку попадут только элементы одного класса.

Далее закодируем информацию которая представлена текстом для того чтобы модель смогла вести зависимости. Так будут выглядеть данные после кодирования.

	name	roaster	roast	loc_country	origin_1	100g_USD	rating
286	271	12	1	4	3	7.33	94.0
1058	788	32	0	0	0	5.88	94.0
646	465	40	0	0	3	6.17	95.0

Вспомним что классификация относится к обучению с учителем, это значит что теперь нужно разметить данные. Решено было взять оценку рейтинга равную 93. Проставим метки классов следующим образом, те строки в которых рейтинг выше 93 будут относиться к классу с меткой **1** все те кто имеют рейтинг 93 и ниже будут относиться к классу с меткой **0**.

После разметки, разделим датасет на тренировочный и тестовый. В этом поможет встроенный метод из библиотеки *sklearn* **train_test_split()**. Данный метод очень полезен, так как автоматически перемешивает строки данных, но это значение можно переопределить(параметр *shuffle* = *False*). Но тогда вы рискуете получить нулевое пересечение классовых меток, что неприятно скажется на ваших моделях.

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

После того как данные разделены перейдем к созданию модели. Используя библиотеку *sklearn* создадим несколько моделей с различными параметрами и посмотрим как они влияют на качество моделей.

Параметр *n_estimators* – это количество деревьев в лесу, по умолчанию равно 100. Создадим еще пару моделей с 500 и 1000 деревьями.

```
Время создания 100 деревьев -  0:00:00
Время создания 500 деревьев -  0:00:00
Время создания 1000 деревьев -  0:00:00
```

Как видим время создания модели не изменилось. Перейдем к обучению моделей. Обучение производится с помощью метода **fit()**, в него передаются тренировочные данные, на которых обучается модель.

```
Время обуения 100 деревьев -  0:00:00.514000
Время обуения 500 деревьев -  0:00:02.715000
Время обуения 1000 деревьев -  0:00:05.174000
```

Эти данные нам пригодятся немного позже, а пока переходим к следующему этапу.

Модель уже обучена и теперь можно посмотреть на какие признаки наиболее влияют на принятие решений моделью.

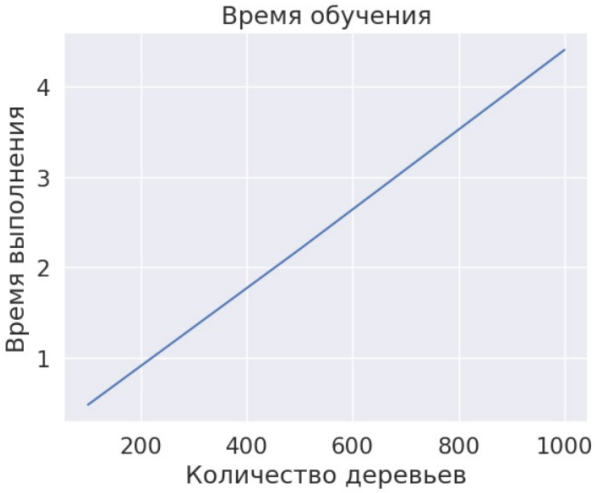
Посмотрим на значимость признаков. Различия были не существенны. Изменения произошли только между последними двумя строчками.

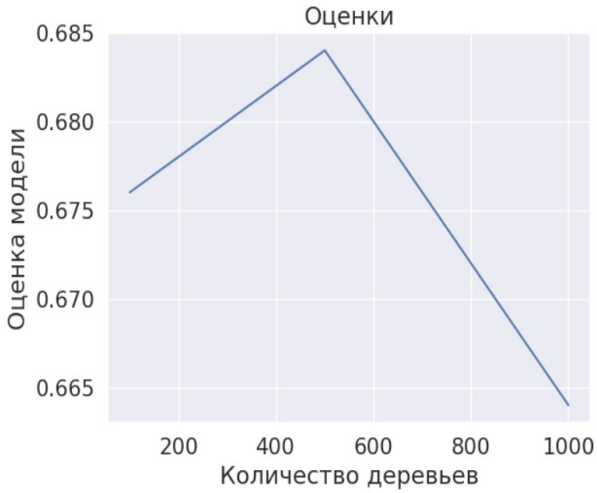
```
100 деревьев                500 деревьев                1000 деревьев
Значимость признаков:       Значимость признаков:       Значимость признаков:
[[0.2552, 'roaster'],       [[0.2532, 'roaster'],       [[0.2512, 'roaster'],
 [0.2343, 'name'],           [0.236, 'name'],            [0.2369, 'name'],
 [0.1895, '100g_USD'],       [0.1898, '100g_USD'],       [0.1896, '100g_USD'],
 [0.1358, 'origin_1'],       [0.1336, 'origin_1'],       [0.1347, 'origin_1'],
 [0.0877, 'rating'],         [0.088, 'rating'],          [0.0882, 'rating'],
 [0.0493, 'roast'],          [0.0503, 'loc_country'],    [0.0504, 'loc_country'],
 [0.0481, 'loc_country']]    [0.049, 'roast']]           [0.049, 'roast']]
```

Теперь перейдем к оценке моделей. Для этого будем использовать все те же встроенные модули библиотеки *sklearn*. Для начала сравним фактические значения с решением модели с помощью **accuracy_score**.

```
Accuracy 100:    0.672
Accuracy 500:    0.684
Accuracy 1000:   0.664
```

Более наглядно это будет выглядеть таким образом.





Как можно заметить не всегда наибольшее количество деревьев ведет к положительному исходу. Получилось так что затраты по времени были гораздо больше, а положительный исход был получен еще меньше чем у модели с наименьшим количеством деревьев.

Но оценка модели мало что дает по сравнению с отчетом классификации. И так вызовем еще один метод из библиотеки.

```
100 деревьев:
              precision    recall  f1-score   support

           0       0.68      0.75      0.71       135
           1       0.67      0.59      0.63       115

    accuracy                           0.68       250
   macro avg       0.67      0.67      0.67       250
weighted avg       0.68      0.68      0.67       250

500 деревьев:
              precision    recall  f1-score   support

           0       0.70      0.73      0.71       135
           1       0.67      0.63      0.65       115

    accuracy                           0.68       250
   macro avg       0.68      0.68      0.68       250
weighted avg       0.68      0.68      0.68       250

1000 деревьев:
              precision    recall  f1-score   support

           0       0.68      0.73      0.70       135
           1       0.65      0.59      0.62       115

    accuracy                           0.66       250
   macro avg       0.66      0.66      0.66       250
weighted avg       0.66      0.66      0.66       250
```

1. **Точность —** это отношение правильных положительных прогнозов к общему количеству положительных прогнозов (выражается в процентах).
2. **Отзыв —** это отношение правильных положительных прогнозов к общему количеству фактических положительных результатов (выражается в процентах).
3. **Оценка f1** – это средневзвешенное гармоническое значение точности и полноты.

2 * (Точность * Отзыв) / (Точность + Отзыв)

Используя данные показатели можно точно определить насколько хороша созданная модель.

Эти показатели куда более информативны чем простая оценка модели.

Подводя итог хотелось бы сказать, что данный пост служит очень кратким учебным примером в рамках которого были разобраны базовые методы. Стоит напомнить что любой алгоритм, пусть даже он написан самыми умными людьми не всегда будет идеально работать. Каждую модель необходимо «дорабатывать» чтобы получить тот самый заветный результат.

GitHub -