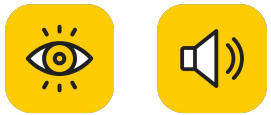
OTUS

Machine Learning. Basic

otus.ru

Меня хорошо видно && слышно?



Защита проекта

Тема: Предсказание затрат времени на проведение анализов выходного контроля при производстве БАД методами ML

Золотарева Наталья

Заместитель начальника контрольно-аналитической лаборатории
ООО ВТФ

План защиты

- Цели проекта
- Что планировалось
- Используемые технологии
- Что получилось
- Схемы/архитектура
- Выводы

Цели проекта

1. Создание модели способной прогнозировать время на анализ всех компонентов продукта на основе информации о его составе
2. Анализ статистической информации о временных затратах на каждое наименование на текущий момент
3. Получение информации о наиболее долгих анализах

Что планировалось

1. Извлечь данные из протоколов в формате .docx (3 дня)
2. Очистить данные (планировалось 3 дня, в реальности больше)
3. Построить различные модели и подобрать эффективную (5 дней)
4. Исследовать данные для получения дополнительной информации

Используемые технологии

1. PyCharm, Jupyter Notebook
2. Pandas, Scikit-learn
3. HTMLParser
4. LibreOffice (из консоли)
5. BeautifulSoup ?

Что получилось Данные

```
In [50]: data = pd.read_csv('finished_data.csv')
         data.head()
```

Out[50]:

	Protocol_number	Product	Component_1	Component_2	Component_3	Component_4	Component_5	Component_6	Component_7	Component_8	...	Com
0	8107	427	1	0	0	0	0	0	0	0	...	
1	4553	427	1	0	0	0	0	0	0	0	...	
2	9831	427	1	0	0	0	0	0	0	0	...	
3	10580	427	1	0	0	0	0	0	0	0	...	
4	1238	427	1	0	0	0	0	0	0	0	...	

5 rows × 137 columns

```
In [3]: data.head().transpose()
```

Out[3]:

	0	1	2	3	4
Protocol_number	8107.0	4553.0	9831.0	10580.0	1238.0
Product	427.0	427.0	427.0	427.0	427.0
Component_1	1.0	1.0	1.0	1.0	1.0
Component_2	0.0	0.0	0.0	0.0	0.0
Component_3	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...
Component_131	0.0	0.0	0.0	0.0	0.0
Component_132	0.0	0.0	0.0	0.0	0.0
Days_analysis	23.0	23.0	18.0	16.0	27.0
Year	2020.0	2020.0	2020.0	2020.0	2020.0
Samples_number	218.0	218.0	163.0	360.0	234.0

137 rows × 5 columns

```
In [49]: data.describe()
```

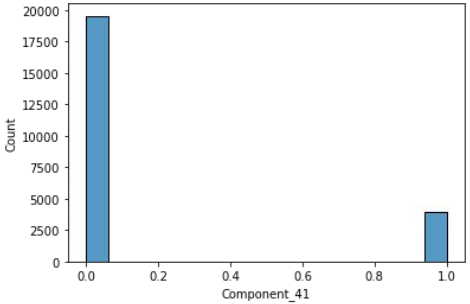
Out[49]:

	Protocol_number	Product	Component_1	Component_2	Component_3	Component_4	Component_5	Component_6	Component_7	Component_8
count	23478.000000	23478.000000	23478.000000	23478.000000	23478.000000	23478.000000	23478.000000	23478.000000	23478.000000	23478.000000
mean	11607.193969	317.309524	0.150311	0.212625	0.238606	0.206023	0.184982	0.286737	0.261564	0.204958
std	6638.486449	126.565430	0.357384	0.409173	0.426241	0.404456	0.388291	0.452247	0.441430	0.403679
min	4.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	5865.250000	221.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	11673.500000	305.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	17401.750000	414.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	0.000000
max	22881.000000	569.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	2.000000	1.000000

8 rows × 137 columns

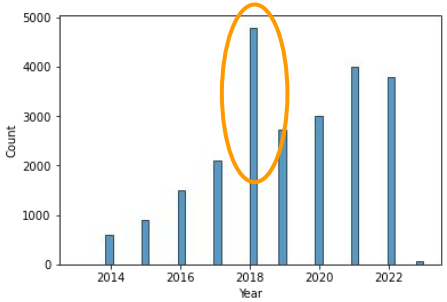
Что получилось Исследование данных

Распределение значений в данных

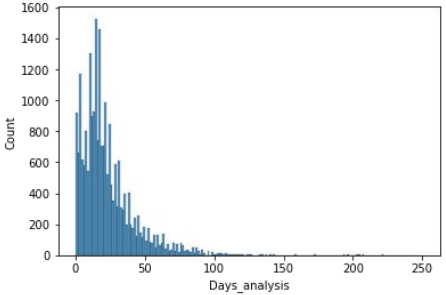


Компоненты

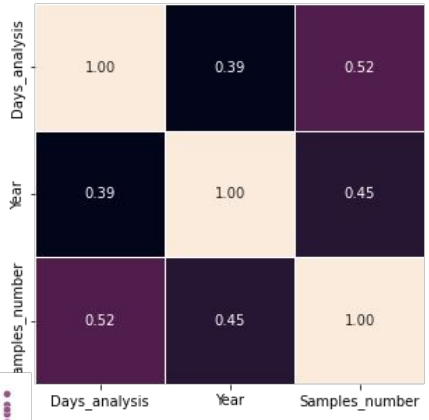
Год анализа

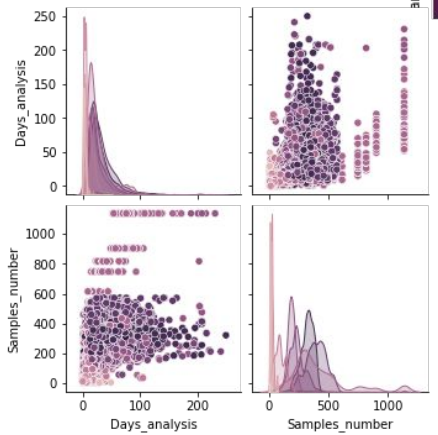


Распределение целевой переменной

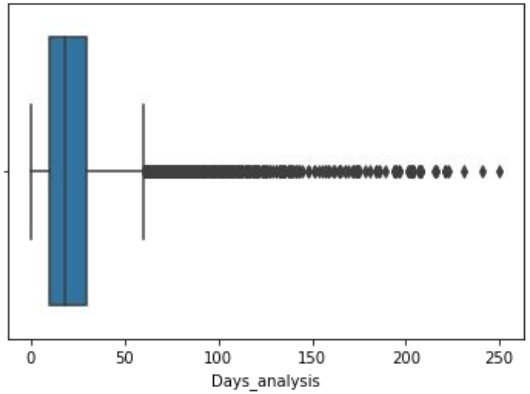


Слабая корреляция



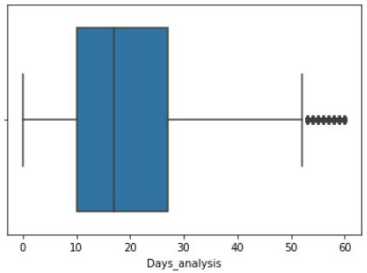


Что получилось Борьба с аномалиями




```
In [26]: perc_75, perc_25 = np.percentile(data['Days_analysis'],[75, 25])
         IQR = perc_75 - perc_25
         k = 1.5
         np.round(data.drop(data[data['Days_analysis'] > perc_75 + k*IQR].index,
                            axis=0).shape[0] / data.shape[0], 2)
Out[26]: 0.94
```

```
In [10]: sns.boxplot(x='Days_analysis', data=data.drop(data[data['Days_analysis'] > perc_75 + k*IQR].index,
                            axis=0))
         plt.show();
```



Итоговая потеря данных

```
In [13]: np.round(data.drop(data[(data['Days_analysis'] > perc_75 + k*IQR) |
                                 (data['Samples_number'] > 650)].index,
                          axis=0).shape[0] / data.shape[0], 2)
Out[13]: 0.93
```

```
In [11]: data[data['Days_analysis'] > perc_75 + k*IQR].groupby('Year').count()
```

Out[11]:

Year	Protocol_number	Product	Component_1	Component_2	Component_3	Component_4	Component_5	Component_6	Component_7	Component_8	...	C
2017	4	4	4	4	4	4	4	4	4	4	...	
2018	436	436	436	436	436	436	436	436	436	436	...	
2019	114	114	114	114	114	114	114	114	114	114	...	
2020	127	127	127	127	127	127	127	127	127	127	...	
2021	457	457	457	457	457	457	457	457	457	457	...	
2022	296	296	296	296	296	296	296	296	296	296	...	
2023	3	3	3	3	3	3	3	3	3	3	...	

7 rows × 135 columns

Что получилось Построение модели

```
In [57]: np.round(data['Days_analysis'].mean(), 2)
Out[57]: 19.31
```

Линейные методы

```
In [16]: from sklearn.model_selection import train_test_split
         from sklearn.preprocessing import StandardScaler
         X = data.drop(['Protocol_number', 'Product', 'Days_analysis'], axis=1)
         Y = data['Days_analysis']
         X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=18)
```

```
In [29]: def rmse(y_pred, y):
             return np.sqrt(mean_squared_error(y_pred, y))
```

```
In [30]: from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
         def err_check(y_train, y_train_pred, y_test, y_pred):
             print('MAE train: ', np.round(mean_absolute_error(y_train, y_train_pred), 2))
             print('MAE test: ', np.round(mean_absolute_error(y_test, y_pred), 2))
             print('RMSE train: ', np.round(rmse(y_train, y_train_pred), 2))
             print('RMSE test: ', np.round(rmse(y_test, y_pred), 2))
             print('R2 train: ', np.round(r2_score(y_train, y_train_pred), 2))
             print('R2 test: ', np.round(r2_score(y_test, y_pred), 2))
```

```
In [22]: from sklearn.linear_model import LinearRegression

         model = LinearRegression()
         model.fit(X_train, y_train)
         y_pred = model.predict(X_test)
         y_train_pred = model.predict(X_train)
         err_check(y_train, y_train_pred, y_test, y_pred)
```

Линейная регрессия

```
MAE train:  7.99
MAE test:  8.25
RMSE train:  10.58
RMSE test:  10.81
R2 train:  0.37
R2 test:  0.35
```

```
In [74]: from sklearn.linear_model import Ridge

         model = Ridge(alpha = 100)
         model.fit(X_train, y_train)
         y_pred = model.predict(X_test)
         y_train_pred = model.predict(X_train)
         err_check(y_train, y_train_pred, y_test, y_pred)
```

```
MAE train:  7.99
MAE test:  8.24
RMSE train:  10.58
RMSE test:  10.81
R2 train:  0.37
R2 test:  0.35
```

Ridge

```
In [76]: from sklearn.linear_model import Lasso

         model = Lasso(alpha = 0.01)
         model.fit(X_train, y_train)
         y_pred = model.predict(X_test)
         y_train_pred = model.predict(X_train)
         err_check(y_train, y_train_pred, y_test, y_pred)
```

```
MAE train:  8.0
MAE test:  8.23
RMSE train:  10.58
RMSE test:  10.8
R2 train:  0.37
R2 test:  0.35
```

Лассо

Метод опорных векторов

```
In [33]: X = data_for_analysis.drop(['Protocol_number', 'Product', 'Days_analysis', 'Year'], axis=1)
         Y = data_for_analysis['Days_analysis']
         X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=18)

         scaler = StandardScaler()
         X_train = scaler.fit_transform(X_train)
         X_test = scaler.transform(X_test)
```

```
In [141]: from sklearn.model_selection import GridSearchCV
          param_grid = {'C':[0.001,0.01,0.1,0.5,1],
                        'kernel':['linear','rbf','poly'],
                        'gamma':['scale','auto'],
                        'degree':[2,3,4],
                        'epsilon':[0,0.01,0.1,0.5,1,2]}

          svr = SVR()
          model = GridSearchCV(model,param_grid=param_grid)
          model.fit(X_train, y_train)
          y_pred = model.predict(X_test)
          y_train_pred = model.predict(X_train)
          err_check(y_train, y_train_pred, y_test, y_pred)
```

```
MAE train:  8.07
MAE test:  7.99
RMSE train:  11.39
RMSE test:  11.27
R2 train:  0.31
R2 test:  0.31
```

```
In [142]: model.best_params_
Out[142]: {'C': 1, 'degree': 2, 'epsilon': 2, 'gamma': 'scale', 'kernel': 'linear'}
```

Что получилось Построение модели

```
In [57]: np.round(data['Days_analysis'].mean(), 2)
Out[57]: 19.31
```

```
In [19]: data_for_analysis = data.drop(data[data['Year'] == 2014].index, axis=0)
         data_for_analysis = data.drop(data[data['Year'] == 2015].index, axis=0)
         data_for_analysis = data.drop(data[data['Year'] == 2016].index, axis=0)
         data_for_analysis = data.drop(data[data['Year'] == 2017].index, axis=0)
         data_for_analysis = data.drop(data[data['Year'] == 2018].index, axis=0)
         data_for_analysis = data.drop(data[data['Year'] == 2019].index, axis=0)
         X = data_for_analysis.drop(['Protocol_number', 'Product', 'Days_analysis', 'Year'], axis=1)
         Y = data_for_analysis['Days_analysis']
         X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=18)
```

```
In [99]: from sklearn.ensemble import GradientBoostingRegressor
         model = GradientBoostingRegressor(n_estimators=500,
             max_depth=4,
             min_samples_split=5,
             learning_rate=0.01,
             loss="squared_error")
         model.fit(X_train, y_train)
         y_pred = model.predict(X_test)
         y_train_pred = model.predict(X_train)
         err_check(y_train, y_train_pred, y_test, y_pred)
```

```
MAE train:  7.5
MAE test:  7.69
RMSE train:  10.09
RMSE test:  10.3
R2 train:  0.43
R2 test:  0.41
```

Пробный запуск

```
In [22]: from sklearn.ensemble import GradientBoostingRegressor
         from sklearn.model_selection import GridSearchCV

         base_model = GradientBoostingRegressor(random_state=15, learning_rate=0.05)
         param_grid = {'n_estimators':[400,500,700],
             'max_depth':[7,10,15],
             'min_samples_split':[4,5,6]}
         model = GridSearchCV(base_model,param_grid=param_grid)
         model.fit(X_train, y_train)
         y_pred = model.predict(X_test)
         y_train_pred = model.predict(X_train)
         err_check(y_train, y_train_pred, y_test, y_pred)
         print(model.best_params_)
```

```
MAE train:  4.29
MAE test:  5.7
RMSE train:  6.79
RMSE test:  8.85
R2 train:  0.75
R2 test:  0.57
{'max_depth': 10, 'min_samples_split': 5, 'n_estimators': 500}
```

Подбор параметров

```
In [32]: from sklearn.model_selection import train_test_split
         X = data.drop(['Protocol_number', 'Product', 'Days_analysis'], axis=1)
         Y = data['Days_analysis']
         X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=18)

         model = GradientBoostingRegressor(random_state=15,
                                           learning_rate=0.05,
                                           n_estimators=500,
                                           max_depth=10,
                                           min_samples_split=5)
         model.fit(X_train, y_train)
         y_pred = model.predict(X_test)
         y_train_pred = model.predict(X_train)
         err_check(y_train, y_train_pred, y_test, y_pred)
```

```
MAE train:  3.88
MAE test:  5.44
RMSE train:  6.28
RMSE test:  8.55
R2 train:  0.78
R2 test:  0.6
```

На полных данных

Итоговая модель для использования

```
In [68]: X = data.drop(['Protocol_number', 'Product', 'Days_analysis','Year'], axis=1)
         Y = data['Days_analysis']
         X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=18)

         model = GradientBoostingRegressor(random_state=15,
                                           learning_rate=0.05,
                                           n_estimators=500,
                                           max_depth=10,
                                           min_samples_split=5)
         model.fit(X_train, y_train)
         y_pred = model.predict(X_test)
         y_train_pred = model.predict(X_train)
         err_check(y_train, y_train_pred, y_test, y_pred)
```

```
MAE train:  4.48
MAE test:  5.93
RMSE train:  6.98
RMSE test:  9.04
R2 train:  0.73
R2 test:  0.55
```

Что получилось Дополнительные задачи

Текущие временные затраты на анализ разных наименований

```
In [131]: df[(df['Days_analysis']>10) & (df['Size']>10)].sort_values('Days_analysis', ascending=False)
```

Out[131]:

Product	Days_analysis	Size
361	43.971429	35
439	41.238095	42
441	39.547619	42
251	36.090909	11
442	35.846154	13
...	...	...
309	10.659341	182
562	10.613169	243
455	10.581395	43
181	10.440000	25
452	10.052632	19

215 rows × 2 columns

Есть статистические данные, чтобы ориентироваться по срокам

Вклад компонентов в итоговое время анализа

```
In [104]: feature_importance = pd.DataFrame(data=model.coef_, index=X.columns, columns=['Importance'])
          feature_importance = feature_importance.sort_values('Importance', axis=0, ascending=False)
          feature_importance.head(20)
```

Out[104]:

	Importance
Component_94	14.743702
Component_16	11.759182
Component_90	11.747588
Component_130	11.116116
Component_103	11.001787
Component_117	9.654633
Component_27	7.907178
Component_112	7.382501
Component_37	7.323867
Component_126	7.263495
Component_29	6.891115
Component_100	6.369808
Component_102	6.369808
Component_101	6.369808
Component_116	6.314122
Component_129	5.925920
Component_92	5.132974
Component_26	5.018018
Component_72	4.865634
Component_110	4.804286

Необходимо изучить методики для определения этих компонентов и, по возможности, оптимизировать по времени

Выводы и планы по развитию

1. Цель по построению приемлемой модели достигнута (но, применить на практике пока нельзя: нет данных о количестве образцов в лаборатории)
2. Получена дополнительная информация о затратах времени на анализы
3. Обнаружены возможные пути оптимизации деятельности лаборатории
4. В перспективе возможно построение более сложной модели, но учитывая специфику исследуемого процесса, маловероятно получение радикально улучшенных результатов

Спасибо за внимание!