|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Системы обработки информации и управления

**Отчёт по лабораторной работе №5**

По дисциплине:

«Технологии машинного обучения»

Выполнил:

Студент группыИУ5-62 **\_\_\_ \_ \_ \_\_\_\_ \_ Андреев И.М.\_\_**

(Подпись, дата) (Фамилия И.О.)

Проверил:

**\_\_\_\_\_\_\_\_\_ \_ Гапанюк Ю. Е.\_**

(Подпись, дата) (Фамилия И.О.)

Москва, 2020

**Задание**

1. Выберите набор данных (датасет) для решения задачи классификации или регрессии.
2. В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков.
3. С использованием метода train\_test\_split разделите выборку на обучающую и тестовую.
4. Обучите следующие модели:
   * одну из линейных моделей;
   * SVM;
   * дерево решений.
5. Оцените качество моделей с помощью двух подходящих для задачи метрик. Сравните качество полученных моделей.

**Дополнительные задания:**

* Проведите эксперименты с важностью признаков в дереве решений.
* Визуализируйте дерево решений.

**Jupyter Notebook**

Импорт библиотек и подготовка данных для моделей была произведена в лабораторной работе №4.

## Логистическая регрессия

In [41]:

**from** **sklearn.linear\_model** **import** LogisticRegression

In [42]:

LogRegression = LogisticRegression()

In [43]:

LogRegression.fit(X\_train,y\_train)

Out[43]:

LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,

intercept\_scaling=1, l1\_ratio=None, max\_iter=100,

multi\_class='auto', n\_jobs=None, penalty='l2',

random\_state=None, solver='lbfgs', tol=0.0001, verbose=0,

warm\_start=False)

In [44]:

y\_predicted\_lr = LogRegression.predict(X\_test)

In [45]:

LogRegression.score(X\_test,y\_test)

Out[45]:

0.8044692737430168

In [46]:

classification\_report(y\_test, y\_predicted\_lr, output\_dict = **True**)

Out[46]:

{'0': {'precision': 0.8598130841121495,

'recall': 0.8214285714285714,

'f1-score': 0.8401826484018264,

'support': 112},

'1': {'precision': 0.7222222222222222,

'recall': 0.7761194029850746,

'f1-score': 0.7482014388489208,

'support': 67},

'accuracy': 0.8044692737430168,

'macro avg': {'precision': 0.7910176531671859,

'recall': 0.798773987206823,

'f1-score': 0.7941920436253735,

'support': 179},

'weighted avg': {'precision': 0.80831259390754,

'recall': 0.8044692737430168,

'f1-score': 0.805753927507722,

'support': 179}}

## SVM

In [47]:

**from** **sklearn.svm** **import** SVC

In [48]:

SVC\_ = SVC()

In [49]:

SVC\_.fit(X\_train,y\_train)

Out[49]:

SVC(C=1.0, break\_ties=False, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape='ovr', degree=3, gamma='scale', kernel='rbf',

max\_iter=-1, probability=False, random\_state=None, shrinking=True,

tol=0.001, verbose=False)

In [50]:

y\_predicted\_svc = SVC\_.predict(X\_test)

y\_predicted\_svc

Out[50]:

array([1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1,

0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1,

1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,

1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,

0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0,

0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1,

0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1,

0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0,

0, 0, 0], dtype=int64)

In [51]:

SVC\_.score(X\_test,y\_test)

Out[51]:

0.8659217877094972

In [52]:

classification\_report(y\_test,y\_predicted\_svc, output\_dict= **True**)

Out[52]:

{'0': {'precision': 0.8548387096774194,

'recall': 0.9464285714285714,

'f1-score': 0.8983050847457628,

'support': 112},

'1': {'precision': 0.8909090909090909,

'recall': 0.7313432835820896,

'f1-score': 0.8032786885245902,

'support': 67},

'accuracy': 0.8659217877094972,

'macro avg': {'precision': 0.8728739002932551,

'recall': 0.8388859275053304,

'f1-score': 0.8507918866351765,

'support': 179},

'weighted avg': {'precision': 0.868339913825587,

'recall': 0.8659217877094972,

'f1-score': 0.8627365453780612,

'support': 179}}

## Дерево решений

In [53]:

**from** **sklearn.tree** **import** DecisionTreeClassifier

In [54]:

DTClassifier = DecisionTreeClassifier(random\_state=1)

In [55]:

DTClassifier.fit(X\_train,y\_train)

Out[55]:

DecisionTreeClassifier(ccp\_alpha=0.0, class\_weight=None, criterion='gini',

max\_depth=None, max\_features=None, 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, presort='deprecated',

random\_state=1, splitter='best')

In [56]:

y\_predicted\_DT = DTClassifier.predict(X\_test)

In [57]:

DTClassifier.score(X\_test,y\_test)

Out[57]:

0.8100558659217877

In [58]:

classification\_report(y\_predicted\_DT,y\_test, output\_dict= **True**)

Out[58]:

{'0': {'precision': 0.7589285714285714,

'recall': 0.9239130434782609,

'f1-score': 0.8333333333333331,

'support': 92},

'1': {'precision': 0.8955223880597015,

'recall': 0.6896551724137931,

'f1-score': 0.7792207792207794,

'support': 87},

'accuracy': 0.8100558659217877,

'macro avg': {'precision': 0.8272254797441365,

'recall': 0.806784107946027,

'f1-score': 0.8062770562770563,

'support': 179},

'weighted avg': {'precision': 0.8253177448749867,

'recall': 0.8100558659217877,

'f1-score': 0.8070328182060025,

'support': 179}}

In [59]:

**from** **sklearn.tree** **import** DecisionTreeClassifier, DecisionTreeRegressor, export\_graphviz

**import** **graphviz**

### Визуализация дерева

In [60]:

dot\_data = export\_graphviz(DTClassifier, out\_file=**None**,

feature\_names=data.columns,

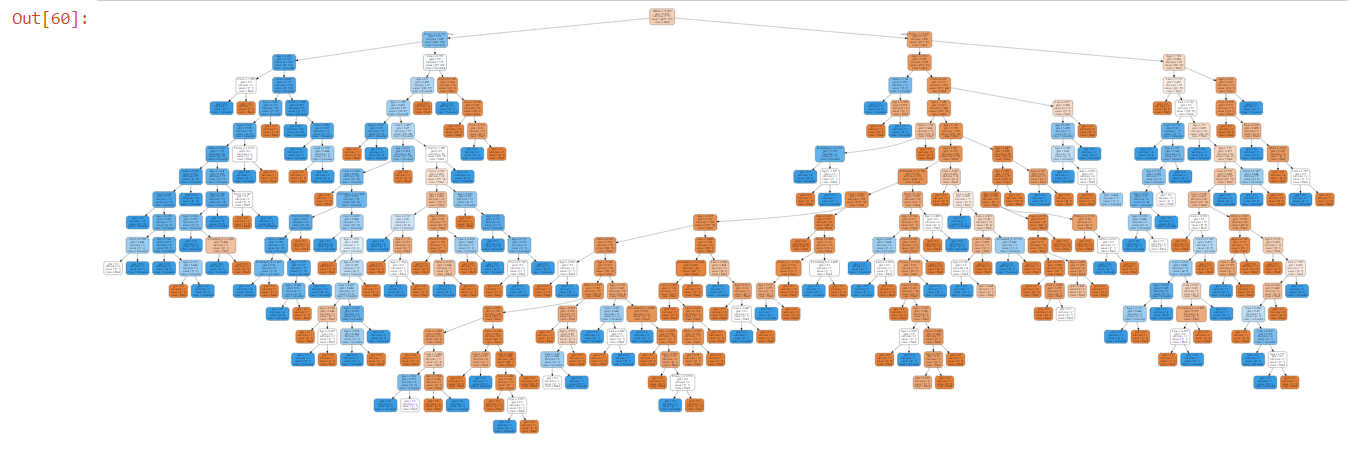
class\_names=('Died','Survived'),

filled=**True**, rounded=**True**, special\_characters=**True**)

graph = graphviz.Source(dot\_data)

graph

Out[60]:



### Важность признаков

In [61]:

**from** **operator** **import** itemgetter

importance = list(zip(data.columns,DTClassifier.feature\_importances\_))

importance\_sort = sorted(importance, key=itemgetter(1), reverse = **True**)

importance\_sort

Out[61]:

[('IsMale', 0.2871670739507118),

('Fare', 0.24321144496159736),

('Age', 0.23672193810675546),

('Pclass\_3', 0.06826073943367003),

('Parch', 0.045973586652106976),

('SibSp', 0.0426733601709724),

('Pclass\_1', 0.03307673997235574),

('Embarked\_S', 0.016283435200530178),

('Pclass\_2', 0.01224728102886664),

('Embarked\_C', 0.009450244136703022),

('Embarked\_Q', 0.004934156385730447)]

In [62]:

DTClassifier2 = DecisionTreeClassifier(random\_state=1)

DTClassifier2.fit(X\_train[['Age']],y\_train)

DTClassifier2.score(X\_test[['Age']],y\_test)

Out[62]:

0.6145251396648045

In [63]:

DTClassifier2 = DecisionTreeClassifier(random\_state=1)

DTClassifier2.fit(X\_train[['IsMale']],y\_train)

DTClassifier2.score(X\_test[['IsMale']],y\_test)

Out[63]:

0.8212290502793296

In [64]:

y\_predicted\_DT2 = DTClassifier2.predict(X\_test[['IsMale']])

classification\_report(y\_predicted\_DT2,y\_test, output\_dict= **True**)

Out[64]:

{'0': {'precision': 0.8660714285714286,

'recall': 0.8508771929824561,

'f1-score': 0.8584070796460177,

'support': 114},

'1': {'precision': 0.746268656716418,

'recall': 0.7692307692307693,

'f1-score': 0.7575757575757576,

'support': 65},

'accuracy': 0.8212290502793296,

'macro avg': {'precision': 0.8061700426439233,

'recall': 0.8100539811066128,

'f1-score': 0.8079914186108876,

'support': 179},

'weighted avg': {'precision': 0.8225676287358101,

'recall': 0.8212290502793296,

'f1-score': 0.8217923537545825,

'support': 179}}

In [65]:

dot\_data1 = export\_graphviz(DTClassifier2, out\_file=**None**,

feature\_names=['IsMale'],

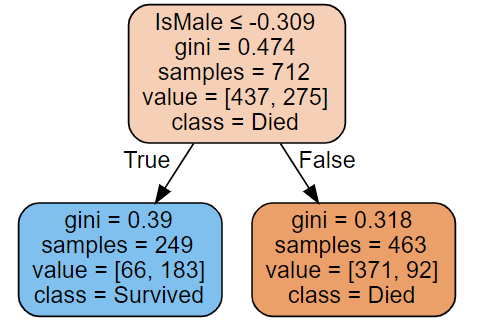
class\_names=('Died','Survived'),

filled=**True**, rounded=**True**, special\_characters=**True**)

graph = graphviz.Source(dot\_data1)

graph

Out[65]:



## Вывод: Лучшей из построенных моделей является модель SVM. Дерево, построенное на признаке "IsMale" тоже показывает хорошие результаты.