**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра ИУ5.

Курс «Технологии машинного обучения»

Отчет по лабораторной работе №5

«Линейные модели, SVM и деревья решений»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-62 |  | преподаватель каф. ИУ5 |
| Король Константин |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2019 г.

Задание

1. Выберите набор данных (датасет) для решения задачи классификации или регресии.
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 1) одну из линейных моделей, 2) SVM и 3) дерево решений. Оцените качество моделей с помощью трех подходящих для задачи метрик. Сравните качество полученных моделей.
5. Произведите для каждой модели подбор одного гиперпараметра с использованием GridSearchCV и кросс-валидации.
6. Повторите пункт 4 для найденных оптимальных значений гиперпараметров. Сравните качество полученных моделей с качеством моделей, полученных в пункте 4.

**Код программы**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

from sklearn.metrics import accuracy\_score, recall\_score, f1\_score

from sklearn.metrics import make\_scorer

from sklearn.model\_selection import GridSearchCV

import matplotlib.pyplot as plt

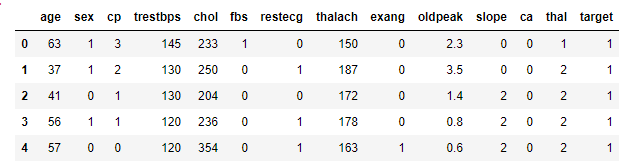
from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

data = pd.read\_csv('data/heart.csv', sep=',')

data.head()



data.isnull().sum()

OUT: age 0

sex 0

cp 0

trestbps 0

chol 0

fbs 0

restecg 0

thalach 0

exang 0

oldpeak 0

slope 0

ca 0

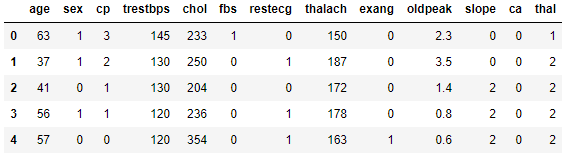
thal 0

target 0

dtype: int64

X = data.loc[:, data.columns != 'target']

X.head()



Y = data['target']

Y.head()

OUT: 0 1

1 1

2 1

3 1

4 1

Name: target, dtype: int64

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, Y, test\_size = 0.2, random\_state = 1)

X\_train.shape, y\_train.shape

OUT: ((242, 13), (242,))

X\_test.shape, y\_test.shape

OUT: ((61, 13), (61,))

logistic = LogisticRegression()

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

logistic\_y\_test = logistic.predict(X\_test)

classification\_report(y\_test, logistic\_y\_test, output\_dict=True)["0"], \

classification\_report(y\_test, logistic\_y\_test, output\_dict=True)["1"]

OUT: ({'precision': 0.8,

'recall': 0.6666666666666666,

'f1-score': 0.7272727272727272,

'support': 30},

{'precision': 0.7222222222222222,

'recall': 0.8387096774193549,

'f1-score': 0.7761194029850746,

'support': 31})

svc = SVC(kernel="rbf", C=0.5)

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

OUT: SVC(C=0.5, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape='ovr', degree=3, gamma='auto\_deprecated',

kernel='rbf', max\_iter=-1, probability=False, random\_state=None,

shrinking=True, tol=0.001, verbose=False)

svc\_y\_test = svc.predict(X\_test)

svc\_y\_test

OUT: array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int64)

classification\_report(y\_test, svc\_y\_test, output\_dict=True)["0"], \

classification\_report(y\_test, svc\_y\_test, output\_dict=True)["1"]

OUT: ({'precision': 0.0, 'recall': 0.0, 'f1-score': 0.0, 'support': 30},

{'precision': 0.5081967213114754,

'recall': 1.0,

'f1-score': 0.673913043478261,

'support': 31})

# при помощи решетчатого поиска и кросс-валидации найдем оптимальное значение гиперпараметра C

scoring = {

'recall': make\_scorer(recall\_score),

'f1': make\_scorer(f1\_score),

'accuracy': make\_scorer(accuracy\_score)

}

svc\_n\_range = [i/10 for i in np.array(range(1, 10, 1))]

svc\_tuned\_parameters = [{'C': svc\_n\_range}]

svc\_tuned\_parameters

OUT: [{'C': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]}]

svc\_gs = GridSearchCV(SVC(kernel="rbf"), svc\_tuned\_parameters, cv=5, scoring=scoring, refit='f1')

svc\_gs.fit(X\_train, y\_train)

OUT: GridSearchCV(cv=5, error\_score='raise-deprecating',

estimator=SVC(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape='ovr', degree=3, gamma='auto\_deprecated',

kernel='rbf', max\_iter=-1, probability=False, random\_state=None,

shrinking=True, tol=0.001, verbose=False),

fit\_params=None, iid='warn', n\_jobs=None,

param\_grid=[{'C': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]}],

pre\_dispatch='2\*n\_jobs', refit='f1', return\_train\_score='warn',

scoring={'recall': make\_scorer(recall\_score), 'f1': make\_scorer(f1\_score), 'accuracy': make\_scorer(accuracy\_score)},

verbose=0)

# лучшая модель

best\_svc = svc\_gs.best\_estimator\_

best\_svc

OUT: SVC(C=0.1, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape='ovr', degree=3, gamma='auto\_deprecated',

kernel='rbf', max\_iter=-1, probability=False, random\_state=None,

shrinking=True, tol=0.001, verbose=False)

# лучшее f1

svc\_gs.best\_score\_

OUT: 0.7127554565117651

# лучшее k

svc\_gs.best\_params\_

OUT: {'C': 0.1}

# на начальном разбиении проверим метрики при новом значении c

best\_svc.fit(X\_train, y\_train)

predicted\_best\_svc = best\_svc.predict(X\_test)

predicted\_best\_svc

OUT: array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int64)

classification\_report(y\_test, predicted\_best\_svc, output\_dict=True)["0"], \

classification\_report(y\_test, predicted\_best\_svc, output\_dict=True)["1"]

OUT: ({'precision': 0.0, 'recall': 0.0, 'f1-score': 0.0, 'support': 30},

{'precision': 0.5081967213114754,

'recall': 1.0,

'f1-score': 0.673913043478261,

'support': 31})

tree = DecisionTreeClassifier(random\_state=1, max\_depth=5)

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

OUT: DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=5,

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=False, random\_state=1,

splitter='best')

tree\_y\_test = tree.predict(X\_test)

tree\_y\_test

OUT: array([0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1,

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

0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0], dtype=int64)

classification\_report(y\_test, tree\_y\_test, output\_dict=True)["0"], \

classification\_report(y\_test, tree\_y\_test, output\_dict=True)["1"]

OUT: ({'precision': 0.7, 'recall': 0.7, 'f1-score': 0.7, 'support': 30},

{'precision': 0.7096774193548387,

'recall': 0.7096774193548387,

'f1-score': 0.7096774193548389,

'support': 31})

# при помощи решетчатого поиска и кросс-валидации найдем оптимальное значение гиперпараметра C

tree\_n\_range = np.array(range(1, 20))

tree\_tuned\_parameters = [{'max\_depth': tree\_n\_range}]

tree\_tuned\_parameters

OUT: [{'max\_depth': array([ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,

18, 19])}]

tree\_gs = GridSearchCV(DecisionTreeClassifier(random\_state=1), tree\_tuned\_parameters, cv=5, scoring=scoring, refit='f1')

tree\_gs.fit(X\_train, y\_train)

OUT: GridSearchCV(cv=5, error\_score='raise-deprecating',

estimator=DecisionTreeClassifier(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=False, random\_state=1,

splitter='best'),

fit\_params=None, iid='warn', n\_jobs=None,

param\_grid=[{'max\_depth': array([ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,

18, 19])}],

pre\_dispatch='2\*n\_jobs', refit='f1', return\_train\_score='warn',

scoring={'recall': make\_scorer(recall\_score), 'f1': make\_scorer(f1\_score), 'accuracy': make\_scorer(accuracy\_score)},

verbose=0)

# лучшая модель

best\_tree = tree\_gs.best\_estimator\_

best\_tree

OUT: DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=4,

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=False, random\_state=1,

splitter='best')

# лучшее значение f1

tree\_gs.best\_score\_

OUT: 0.8615494578662056

# на начальном разбиении проверим метрики при новом значении c

best\_tree.fit(X\_train, y\_train)

predicted\_best\_tree = best\_tree.predict(X\_test)

predicted\_best\_tree

OUT: array([0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0,

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

0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0], dtype=int64)

classification\_report(y\_test, predicted\_best\_tree, output\_dict=True)["0"], \

classification\_report(y\_test, predicted\_best\_tree, output\_dict=True)["1"]

OUT: ({'precision': 0.7666666666666667,

'recall': 0.7666666666666667,

'f1-score': 0.7666666666666667,

'support': 30},

{'precision': 0.7741935483870968,

'recall': 0.7741935483870968,

'f1-score': 0.7741935483870968,

'support': 31})

# таким образом из трех моделей лучший результат показал метод Логистической регрессии

classification\_report(y\_test, logistic\_y\_test, output\_dict=True)["0"], \

classification\_report(y\_test, logistic\_y\_test, output\_dict=True)["1"]

OUT: ({'precision': 0.8,

'recall': 0.6666666666666666,

'f1-score': 0.7272727272727272,

'support': 30},

{'precision': 0.7222222222222222,

'recall': 0.8387096774193549,

'f1-score': 0.7761194029850746,

'support': 31})