Московский государственный технический университет им. Н.Э. Баумана
Кафедра «Системы обработки информации и управления»

Лабораторная работа №6
по дисциплине
«Методы машинного обучения»
на тему
«Ансамбли моделей машинного обучения»

Выполнил:
студент группы ИУ5-23М
Наседкин И. А.

Москва — 2020 г.

1. Ансамбли моделей машинного обучения

Цель лабораторной работы: изучение ансамблей моделей машинного обучения. 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 две ансамблевые модели. Оцените качество моделей с помощью одной из подходящих для задачи метрик. Сравните качество полученных моделей.
5. Произведите для каждой модели подбор значений одного гиперпараметра. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать перебор параметров в цикле, или использовать другие методы.
6. Повторите пункт 4 для найденных оптимальных значений гиперпараметров. Сравните качество полученных моделей с качеством моделей, полученных в пункте 4.

1. Подготовка данных; датасет - https://www.kaggle.com/ronitf/heart-disease-uci/version/1
2. age;—возраст;
3. sex;—пол;
4. chest pain type (4 values);—Тип боли;
5. resting blood pressure;—Кровяное давление в покое;
6. serum cholestoral in mg/dl;—Холестерин;
7. fasting blood sugar > 120 mg/dl;—Сахар в крови;
8. resting electrocardiographic results (values 0,1,2);—Электрокардиография в покое;
9. maximum heart rate achieved;—Максимальный сердечный ритм;
10. exercise induced angina;—Стенокардия вызванная физической нагрузкой;
11. oldpeak = ST depression induced by exercise relative to rest;—депрессия вызванная физ упражнениями;
12. the slope of the peak exercise ST segment;—Наклон пика упражнений;
13. number of major vessels (0-3) colored by flourosopy;—Кол-во крупных сосоудов по цвету thal: 3 = normal; 6 = fixed defect; 7 = reversable defect;

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19:
FutureWarning: pandas.util.testing is deprecated. Use the functions in
  ↪the
public API at pandas.testing instead.
  import pandas.util.testing as tm
```

[0]:
```
data = pd.read_csv('heart.csv')
```

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

[3]:
```
   age  sex  cp  trestbps  chol  fbs  …  exang  oldpeak  slope  ca  thal
target
0   63    1   3       145   233    1  …      0      2.3      0   0     1
1
1   37    1   2       130   250    0  …      0      3.5      0   0     2
1
2   41    0   1       130   204    0  …      0      1.4      2   0     2
1
3   56    1   1       120   236    0  …      0      0.8      2   0     2
1
4   57    0   0       120   354    0  …      1      0.6      2   0     2
1

[5 rows x 14 columns]
```

[4]:
```
data.shape
```

[4]:
```
(303, 14)
```

[5]:
```
data.isnull().sum()
```

[5]:
```
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
```

Датасет без пустых значений

1.0.1. Feature Scaling

[6]:
```
from sklearn.preprocessing import MinMaxScaler
import warnings

from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc
import pylab as pl
import matplotlib.pyplot as plt
```

```
warnings.filterwarnings('ignore')

# Create the scaler object with a range of 0-1
scaler = MinMaxScaler(feature_range=(0, 1))
# Fit on data, transform  data
scaler.fit_transform(data)
```

[6]:
```
array([[0.70833333, 1.        , 1.        , …, 0.        , 0.33333333,
        1.        ],
       [0.16666667, 1.        , 0.66666667, …, 0.        , 0.66666667,
        1.        ],
       [0.25      , 0.        , 0.33333333, …, 0.        , 0.66666667,
        1.        ],
       …,
       [0.8125    , 1.        , 0.        , …, 0.5       , 1.        ,
        0.        ],
       [0.58333333, 1.        , 0.        , …, 0.25      , 1.        ,
        0.        ],
       [0.58333333, 0.        , 0.33333333, …, 0.25      , 0.66666667,
        0.        ]])
```

Разделим датасет на тестовую и обучающую выборки

[0]:
```
X = data.drop('target',axis = 1).values
y = data['target'].values
```

1.0.2. Ансамблевые модели

[0]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score

from sklearn.model_selection import GridSearchCV
```

[0]:
```
kfold = 5 #количество подвыборок для валидации
```

[0]:
```
itog_val = {} #список для записи результатов кросс валидации разных
 ↪алгоритмов
```

[0]:
```
ROCtrainTRN, ROCtestTRN, ROCtrainTRG, ROCtestTRG = train_test_split(X,
 ↪y, test_size=0.20)
```

[0]:
```
model_rfc = RandomForestClassifier(n_estimators = 75) #в параметре
 ↪передаем кол-во деревьев
```

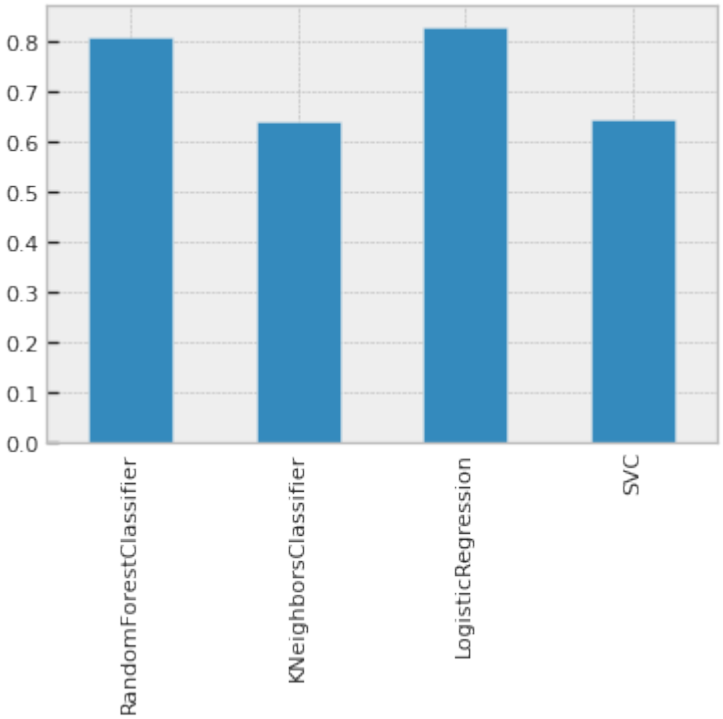
```
model_knc = KNeighborsClassifier(n_neighbors = 20) #в параметре
 ↪передаем кол-во соседей
model_lr = LogisticRegression(penalty='l2', tol=0.01) #l1
model_svc = svm.SVC() #по умолчанию kernel='rbf'
```

1. SVM - метод опорных векторов(SVC)
2. Метод k-ближайших соседей(KNeighborsClassifier)
3. Случайный лес(RandomForestClassifier)
4. Логистическая регрессия (LogisticRegression)

[0]:
```
scores = cross_val_score(model_rfc, X, y, cv = kfold)
itog_val['RandomForestClassifier'] = scores.mean()
scores = cross_val_score(model_knc, X, y, cv = kfold)
itog_val['KNeighborsClassifier'] = scores.mean()
scores = cross_val_score(model_lr, X, y, cv = kfold)
itog_val['LogisticRegression'] = scores.mean()
scores = cross_val_score(model_svc, X, y, cv = kfold)
itog_val['SVC'] = scores.mean()
```

[14]:
```
plt.style.use('bmh')
data.from_dict(data = itog_val, orient='index').plot(kind='bar',
 ↪legend=False)
```

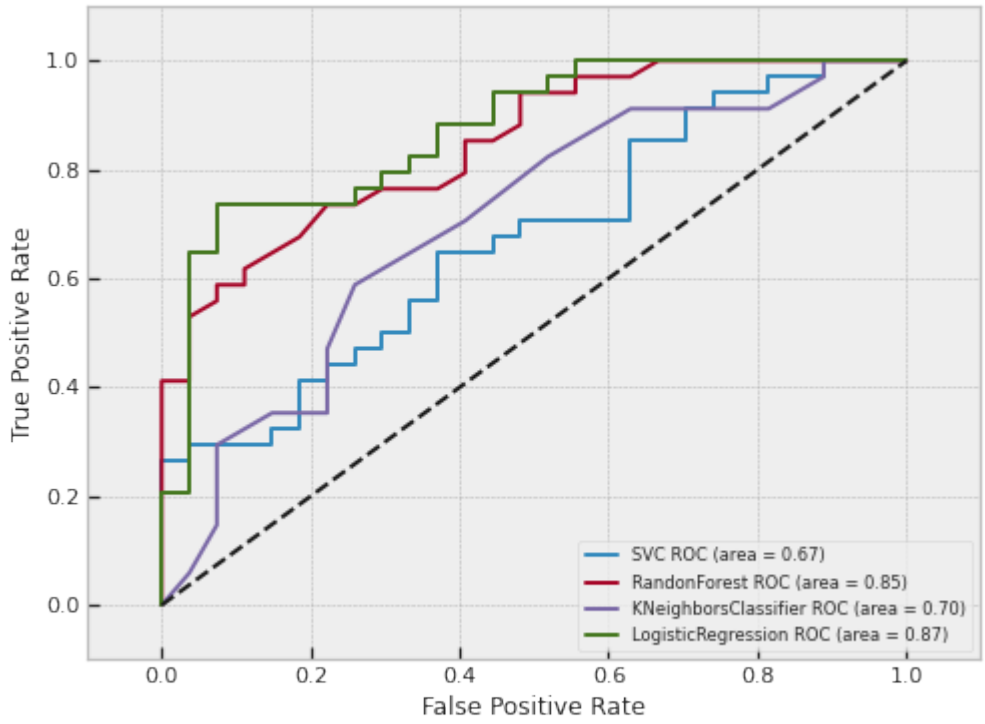
[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5d112094a8>
```



[15]:
```
pl.clf()
plt.figure(figsize=(8,6))
```

```
#SVC
model_svc.probability = True
probas = model_svc.fit(ROCtrainTRN, ROCtrainTRG).
 ↪predict_proba(ROCtestTRN)
fpr, tpr, thresholds = roc_curve(ROCtestTRG, probas[:, 1])
roc_auc  = auc(fpr, tpr)
pl.plot(fpr, tpr, label='%s ROC (area = %0.2f)' % ('SVC', roc_auc))
#RandomForestClassifier
probas = model_rfc.fit(ROCtrainTRN, ROCtrainTRG).
 ↪predict_proba(ROCtestTRN)
fpr, tpr, thresholds = roc_curve(ROCtestTRG, probas[:, 1])
roc_auc  = auc(fpr, tpr)
pl.plot(fpr, tpr, label='%s ROC (area = %0.2f)' %
 ↪('RandonForest',roc_auc))
#KNeighborsClassifier
probas = model_knc.fit(ROCtrainTRN, ROCtrainTRG).
 ↪predict_proba(ROCtestTRN)
fpr, tpr, thresholds = roc_curve(ROCtestTRG, probas[:, 1])
roc_auc  = auc(fpr, tpr)
pl.plot(fpr, tpr, label='%s ROC (area = %0.2f)' %
 ↪('KNeighborsClassifier',roc_auc))
#LogisticRegression
probas = model_lr.fit(ROCtrainTRN, ROCtrainTRG).
 ↪predict_proba(ROCtestTRN)
fpr, tpr, thresholds = roc_curve(ROCtestTRG, probas[:, 1])
roc_auc  = auc(fpr, tpr)
pl.plot(fpr, tpr, label='%s ROC (area = %0.2f)' %
 ↪('LogisticRegression',roc_auc))
pl.plot([0, 1], [0, 1], 'k--')
pl.xlim([-0.1, 1.1])
pl.ylim([-0.1, 1.1])
pl.xlabel('False Positive Rate')
pl.ylabel('True Positive Rate')
pl.legend(loc=0, fontsize='small')
pl.show()
```

```
<Figure size 432x288 with 0 Axes>
```



[0]:
```
# Функция train_test_split разделила исходную выборку таким образом,
#чтобы в обучающей и тестовой частях сохранились пропорции классов.
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.35, random_state=1)
```

[17]:
```
from sklearn.preprocessing import MinMaxScaler
warnings.filterwarnings('ignore')

# Create the scaler object with a range of 0-1
scaler = MinMaxScaler(feature_range=(0, 1))
# Fit on data, transform  data
scaler.fit_transform(X)
scaler.fit_transform(X_train)
scaler.fit_transform(X_test)
```

[17]:
```
array([[0.77777778, 0.        , 0.        , …, 0.        , 0.75      ,
        1.        ],
       [0.61111111, 1.        , 0.33333333, …, 1.        , 0.        ,
        1.        ],
       [0.38888889, 1.        , 0.        , …, 1.        , 0.5       ,
        1.        ],
       …,
       [0.52777778, 1.        , 0.        , …, 0.        , 0.        ,
        1.        ],
       [0.66666667, 1.        , 0.33333333, …, 0.5       , 1.        ,
        1.        ],
       [0.38888889, 1.        , 0.33333333, …, 0.        , 0.        ,
        1.        ]])
```

[0]:
```
rfc = RandomForestClassifier().fit(X_train, y_train)
predicted_rfc = rfc.predict(X_test)
```

[19]:
```
accuracy_score(y_test, predicted_rfc)
```

[19]:
```
0.7476635514018691
```

[20]:
```
balanced_accuracy_score(y_test, predicted_rfc)
```

[20]:
```
0.7484210526315789
```

[21]:
```
(precision_score(y_test, predicted_rfc, average='weighted'),
 recall_score(y_test, predicted_rfc, average='weighted'))
```

[21]:
```
(0.7493681302533357, 0.7476635514018691)
```

[22]:
```
f1_score(y_test, predicted_rfc, average='weighted')
```

[22]:
```
0.7479284020432736
```

[0]:
```
abc = AdaBoostClassifier().fit(X_train, y_train)
predicted_abc = abc.predict(X_test)
```

[24]:
```
accuracy_score(y_test, predicted_abc)
```

[24]:
```
0.7289719626168224
```

[25]:
```
balanced_accuracy_score(y_test, predicted_abc)
```

[25]:
```
0.7284210526315789
```

[26]:
```
(precision_score(y_test, predicted_abc, average='weighted'),
 recall_score(y_test, predicted_abc, average='weighted'))
```

[26]:
```
(0.7293842770753162, 0.7289719626168224)
```

[27]:
```
f1_score(y_test, predicted_abc, average='weighted')
```

[27]:
```
0.7291144464706996
```

[28]:
```
rfc_n_range = np.array(range(5,100,5))
rfc_tuned_parameters = [{'n_estimators': rfc_n_range}]
rfc_tuned_parameters
```

[28]:
```
[{'n_estimators': array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55,
  ↪60, 65, 70,
75, 80, 85,
         90, 95])}]
```

[29]:
```
warnings.filterwarnings('ignore')

gs_rfc = GridSearchCV(RandomForestClassifier(), rfc_tuned_parameters,
 ↪cv=5,
```

```
                      scoring='accuracy')
gs_rfc.fit(X_train, y_train)
```

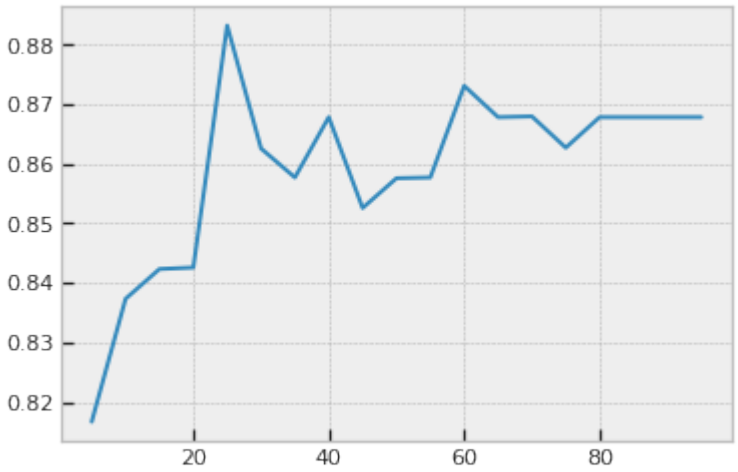
[29]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True,
  ↪ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini',
  ↪max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=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=100,
  ↪n_jobs=None,
                                              oob_score=False,
                                              random_state=None,
  ↪verbose=0,
                                              warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid=[{'n_estimators': array([ 5, 10, 15, 20, 25,
  ↪30, 35, 40,
       45, 50, 55, 60, 65, 70, 75, 80, 85,
              90, 95])}],
             pre_dispatch='2*n_jobs', refit=True,
  ↪return_train_score=False,
             scoring='accuracy', verbose=0)
```

[30]:
```
gs_rfc.best_params_
```

[30]:
```
{'n_estimators': 25}
```

[31]:
```
plt.plot(rfc_n_range, gs_rfc.cv_results_['mean_test_score'])
```

[31]:
```
[<matplotlib.lines.Line2D at 0x7f5d10b90828>]
```



[32]:
```
abc_n_range = np.array(range(5,100,5))
abc_tuned_parameters = [{'n_estimators': abc_n_range}]
abc_tuned_parameters
```

[32]:
```
[{'n_estimators': array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55,
  ↪60, 65, 70,
75, 80, 85,
         90, 95])}]
```

[33]:
```
gs_abc = GridSearchCV(AdaBoostClassifier(), abc_tuned_parameters, cv=5,
                      scoring='accuracy')
gs_abc.fit(X_train, y_train)
```

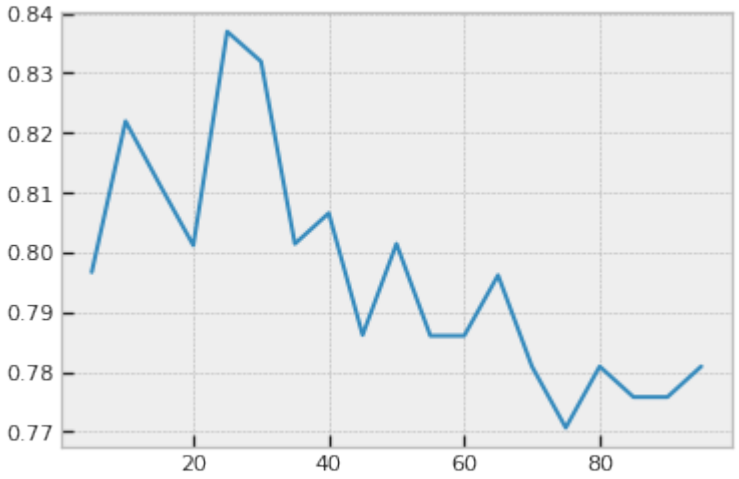
[33]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=AdaBoostClassifier(algorithm='SAMME.R',
                                          base_estimator=None,
                                          learning_rate=1.0,
  ↪n_estimators=50,
                                          random_state=None),
             iid='deprecated', n_jobs=None,
             param_grid=[{'n_estimators': array([ 5, 10, 15, 20, 25,
  ↪30, 35, 40,
45, 50, 55, 60, 65, 70, 75, 80, 85,
       90, 95])}],
             pre_dispatch='2*n_jobs', refit=True,
  ↪return_train_score=False,
             scoring='accuracy', verbose=0)
```

[34]:
```
gs_abc.best_params_
```

[34]:
```
{'n_estimators': 25}
```

[35]:
```
plt.plot(abc_n_range, gs_abc.cv_results_['mean_test_score'])
```

[35]:
```
[<matplotlib.lines.Line2D at 0x7f5d10b16f28>]
```



[0]:
```
rfc_optimized = RandomForestClassifier(n_estimators=gs_rfc.
  ↪best_params_['n_estimators']).fit(X_train, y_train)
predicted_rfc_opt = rfc_optimized.predict(X_test)
```

[37]:
```
accuracy_score(y_test, predicted_rfc_opt)
```

[37]:
```
0.7009345794392523
```

[38]:
```
balanced_accuracy_score(y_test, predicted_rfc_opt)
```

[38]:
```
0.6984210526315789
```

[39]:
```
(precision_score(y_test, predicted_rfc_opt, average='weighted'),
 recall_score(y_test, predicted_rfc_opt, average='weighted'))
```

[39]:
```
(0.7004791699667353, 0.7009345794392523)
```

[40]:
```
f1_score(y_test, predicted_rfc_opt, average='weighted')
```

[40]:
```
0.7004610416515945
```

[0]:
```
abc_optimized = RandomForestClassifier(n_estimators=gs_abc.
  ↪best_params_['n_estimators']).fit(X_train, y_train)
predicted_abc_opt = abc_optimized.predict(X_test)
```

[42]:
```
accuracy_score(y_test, predicted_abc_opt)
```

[42]:
```
0.7383177570093458
```

[43]:
```
balanced_accuracy_score(y_test, predicted_abc_opt)
```

[43]:
```
0.7371929824561403
```

[44]:
```
(precision_score(y_test, predicted_abc_opt, average='weighted'),
 recall_score(y_test, predicted_abc_opt, average='weighted'))
```

[44]:
```
(0.7383177570093458, 0.7383177570093458)
```

[45]:
```
f1_score(y_test, predicted_abc_opt, average='weighted')
```

[45]:
```
0.7383177570093458
```

Результаты для разных методов примерно одинаковые.

[0]:
```

```