Домашнее задание по дисциплине «Методы машинного обучения»

Домашнее задание по дисциплине направлено на решение комплексной задачи машинного обучения.

Домашнее задание включает выполнение следующих шагов:
1. Поиск и выбор набора данных для построения моделей машинного обучения. На основе выбранного набора данных студент должен построить модели машинного обучения для решения или задачи классификации, или задачи регрессии.
2. Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.
3. Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.
4. Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения. В зависимости от набора данных, порядок выполнения пунктов 2, 3, 4 может быть изменен.
5. Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее двух метрик и обосновать выбор.
6. Выбор наиболее подходящих моделей для решения задачи классификации или регрессии. Необходимо использовать не менее трех моделей, хотя бы одна из которых должна быть ансамблевой.
7.Формирование обучающей и тестовой выборок на основе исходного набора данных.
8. Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.
9. Подбор гиперпараметров для выбранных моделей. Рекомендуется подбирать не более 1-2 гиперпараметров. Рекомендуется использовать методы кросс-валидации. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать перебор параметров в цикле, или использовать другие методы.
10. Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.
11. Формирование выводов о качестве построенных моделей на основе выбранных метрик.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
%matplotlib inline
```

Загрузка данных

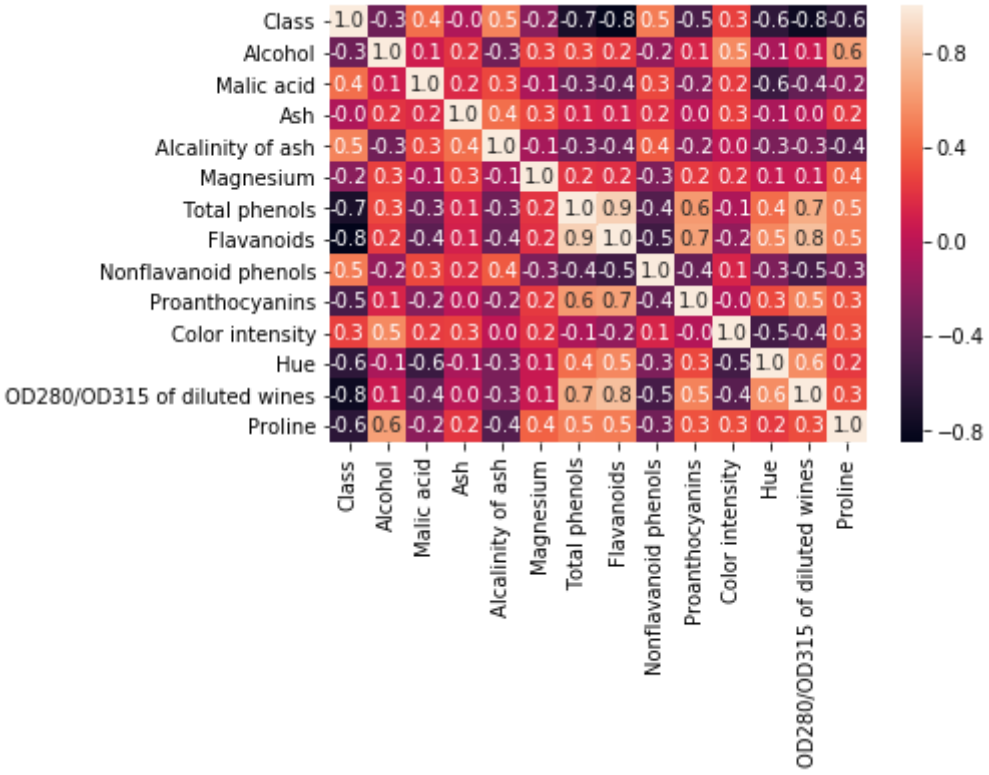
```
data  = pd.read_csv("Wine.csv", sep=";")
```

```
data.head()
```

	Class	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonfla p
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	

```
sns.heatmap(data.corr(method='pearson'), annot=True, fmt='.1f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f70831ea550>
```




```
for col in data.columns:
  temp=data[data[col].isnull()].shape[0]
  print('{}-{}'.format(col, temp))
print("----------------------")
data.dtypes
```

```
Class-0
Alcohol-0
Malic acid-0
Ash-0
Alcalinity of ash-0
Magnesium-0
Total phenols-0
Flavanoids-0
Nonflavanoid phenols-0
Proanthocyanins-0
Color intensity-0
Hue-0
OD280/OD315 of diluted wines-0
Proline-0
----------------------
Class                           int64
Alcohol                       float64
Malic acid                    float64
Ash                           float64
Total phenols                 float64
OD280/OD315 of diluted wines  float64
```

Заметим что датасет не содержит категориальных признаков и пропусков

Выбор метрик

Для оценки качества моделей будем использовать следующие метрики: -Средняя абсолютная ошибка -Каппа Коэна

```
from sklearn.metrics import mean_absolute_error, cohen_kappa_score
```

Выбор моделей

В качестве моделей возьмем линейную модель стохастического спуска, дерево решений и ансамблевый метод повышения градиента

```
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

Разделение выборки на обучающую и тестовую

```
CLASS = 'Class'
RANDOM_STATE = 17
TEST_SIZE = 0.3

X = data.drop(CLASS, axis=1).values
Y = data[CLASS].values

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=TEST_SIZE, random_st
print('X_train: {}'.format(X_train.shape))
print('X_test: {}'.format(X_test.shape))
```

```
X_train: (124, 13)
X_test: (54, 13)
```

Построение базового решения без подбора гиперпараметров

```
class Classifier():
  def __init__(self, method, x_train, y_train, x_test, y_test):
    self._method = method
    self.x_train = x_train
    self.y_train = y_train
    self.x_test = x_test
    self.y_test = y_test
    self.tar1 = []
    self.tar2 = []
  def training(self):
    self._method.fit(self.x_train,self.y_train)
    self.tar2 = self._method.predict(self.x_test)
  def result(self,metric):
    print(metric(self.y_test,self.tar2)*100)
```

SGD - реализует регуляризованные линейные модели с обучением по случайному градиентному спуску (SGD): градиент потерь оценивается для каждой выборки за раз, и модель обновляется по мере уменьшения скорости обучения.

```
#Линейные модели
sgdlinear = Classifier(SGDClassifier(), X_train, Y_train, X_test, Y_test)
sgdlinear.training()
sgdlinear.result(mean_absolute_error)
sgdlinear.result(cohen_kappa_score)
```

```
40.74074074074074
44.208809135399676
```

Модель, которая прогнозирует значение целевой переменной путем изучения простых правил принятия решений, выведенных из функций данных.

```
dtc = Classifier(DecisionTreeClassifier(random_state=5), X_train, Y_train, X_test, Y_tes
dtc.training()
dtc.result(mean_absolute_error)
dtc.result(cohen_kappa_score)
```

```
1.8518518518518516
97.20062208398133
```

```
gbc=Classifier(GradientBoostingClassifier(max_features=2), X_train, Y_train, X_test, Y_t
gbc.training()
gbc.result(mean_absolute_error)
gbc.result(cohen_kappa_score)
```

```
0.0
100.0
```

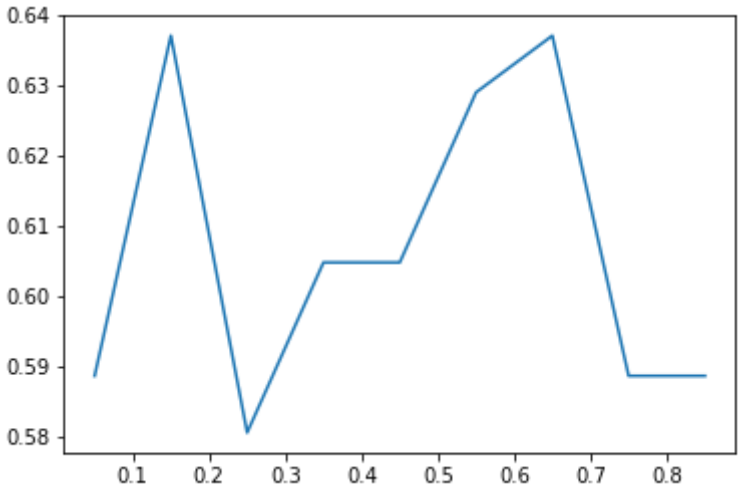
Подбор гиперпараметра K

```
n_range = np.array(range(5,95,10))
n_range = n_range/100
tp=[{'l1_ratio':n_range}]

lgscv = GridSearchCV(SGDClassifier(), tp, scoring='accuracy')
lgscv.fit(X_train, Y_train)

bp1=lgscv.best_params_['l1_ratio']
bp1
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:1978: Fu
  warnings.warn(CV_WARNING, FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_search.py:813: De
  DeprecationWarning)
0.15
```

```
plt.plot(n_range,lgscv.cv_results_['mean_test_score'])
```

```
[<matplotlib.lines.Line2D at 0x7f7085c18860>]
```

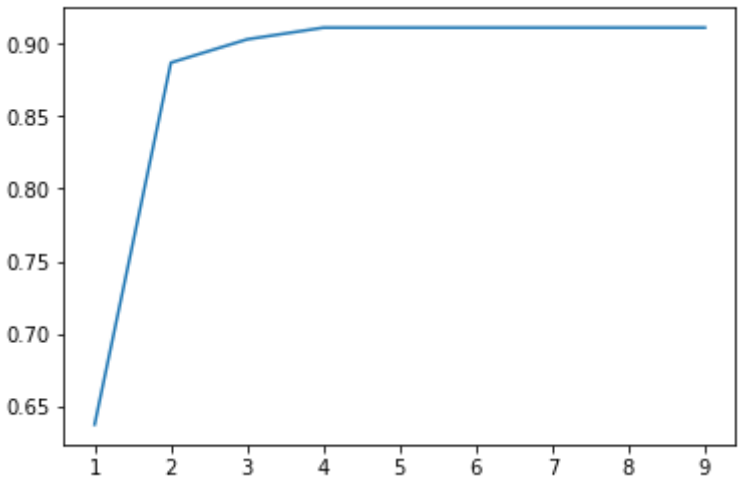



```
n_range = np.array(range(1,10,1))
tp=[{'max_depth':n_range}]

tgscv = GridSearchCV(DecisionTreeClassifier(random_state=1), tp, cv=5, scoring='accuracy
tgscv.fit(X_train, Y_train)

bp2=tgscv.best_params_['max_depth']
bp2
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_search.py:813: De
  DeprecationWarning)
4
```

```
plt.plot(n_range,tgscv.cv_results_['mean_test_score'])
```

```
[<matplotlib.lines.Line2D at 0x7f70858735c0>]
```



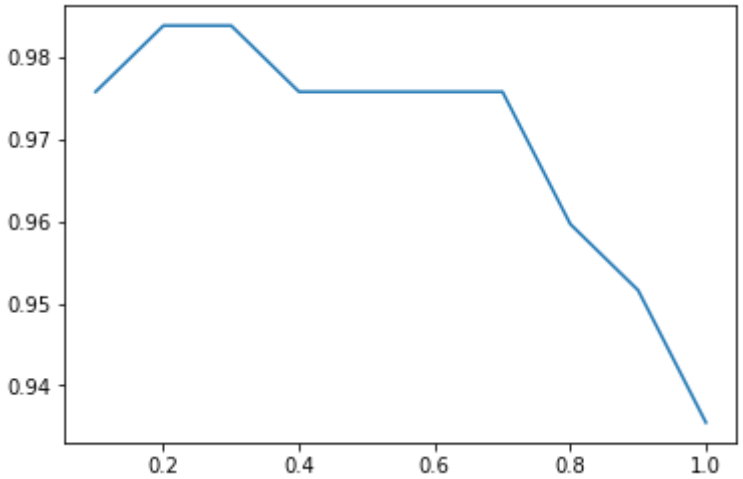

```
n_range = np.array(range(1,11,1))
```

```
n_range = n_range/10
tp=[{'max_features':n_range}]

gbcgscv = GridSearchCV(GradientBoostingClassifier(), tp, cv=5, scoring='accuracy')
gbcgscv.fit(X_train, Y_train)

bp3=gbcgscv.best_params_['max_features']
bp3
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_search.py:813: De
  DeprecationWarning)
0.2
```

```
plt.plot(n_range,gbcgscv.cv_results_['mean_test_score'])
```

```
[<matplotlib.lines.Line2D at 0x7f70832ea240>]
```



Сравнение моделей

```
#Линейные модели
sgdlinear.result(mean_absolute_error)
sgdlinear.result(cohen_kappa_score)
print("_____________________________________")
sgdlinear2 = Classifier(SGDClassifier(l1_ratio=bp1), X_train, Y_train, X_test, Y_test)
sgdlinear2.training()
sgdlinear2.result(mean_absolute_error)
sgdlinear.result(cohen_kappa_score)
```

```
40.74074074074074
44.208809135399676
_____________________________________
55.55555555555556
44.208809135399676
```

```
#DTC
dtc.result(mean_absolute_error)
dtc.result(cohen_kappa_score)
print("_____________________________________")
dtc2 = Classifier(DecisionTreeClassifier(random_state=bp2), X_train, Y_train, X_test, Y_
dtc2.training()
dtc2.result(mean_absolute_error)
dtc2.result(cohen_kappa_score)
```

```
1.8518518518518516
97.20062208398133

_____________________________________
1.8518518518518516
```

```
gbc.result(mean_absolute_error)
gbc.result(cohen_kappa_score)
print("vs")
gbc2=Classifier(GradientBoostingClassifier(max_features=bp3), X_train, Y_train, X_test,
gbc2.training()
gbc2.result(mean_absolute_error)
gbc2.result(cohen_kappa_score)
```

```
0.0
100.0
vs
0.0
100.0
```

Выводы:

По полученным моделям и значениям можно сделать следующие выводы:

1. Наилучшим методом оказался ансамблевский GradiendBoosting показав средние ~100%
2. Несмотря на визуально незначительный прирост после использования расчитанных гиперпараметров использовать случайные гиперпараметры не рекомендуется.

Литература

1. Heart Disease UCI: https://www.kaggle.com/ronitf/heart-disease-uci
2. Scikit-learn docs: https://scikit-learn.org/stable/modules/