Московский авиационный институт

(Национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу «Машинное обучение»

Тема: Создание линейных моделей

Студент: Харьков Павел Александрович

Группа: М80-406Б-19

Дата:

Оценка:

Москва, 2022

1. Постановка задачи

- Реализовать алгоритмы машинного обучения: Linear/ Logistic Regression, SVM, KNN, Naive Bayes в отдельных классах
- Данные классы наследовать от BaseEstimator и ClassifierMixin, иметь методы fit и predict
- Организовать весь процесс предобработки, обучения и тестирования с помощью Pipeline
- Настроить гиперпараметры моделей с помощью кросс валидации, вывести и сохранить эти гиперпараметры в файл, вместе с обученными моделями
- Проделать аналогично с коробочными решениями
- Для каждой модели получить оценки метрик: Confusion Matrix, Accuracy, Recall, Precision, ROC_AUC curve
- Проанализировать полученные результаты и сделать выводы о применимости моделей
- Загрузить полученные гиперпараметры модели и обученные модели в формате pickle на гит вместе с jupyter notebook ваших экспериментов

2. Ход работы

В качестве датасета я возьму уже обработанный мной в предыдущей лабораторной работе набор данных.

In [ ]:
```
import numpy as np
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

import pandas as pd
import pickle
```

Загрузим наш датасет и разделим его на обучающую и тестовую выборки:

In [ ]:
```
data = pd.read_csv('weatherAfter.csv', index_col=0)
x = data.drop(['RainTomorrow'], axis=1).to_numpy()
y = np.array(data['RainTomorrow'])
scaler = StandardScaler()

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, shuffle
```

Функция для оценки качества предсказаний:

In [ ]:
```
def print_res(preds, test):
    print("Accuracy: {}".format(accuracy_score(preds, test)))
    print("Recall: {}".format(recall_score(preds, test)))
    print("Precision: {}".format(precision_score(preds, test)))
    print("Roc-auc: {}".format(roc_auc_score(preds, test)))
    print("Confusion matrix:\n{}".format(confusion_matrix(preds, test)))
```

Логистическая регрессия

In [ ]:
```
class MyLogisticRegression(BaseEstimator, ClassifierMixin):
    def __init__(self, n = 200, lr = 0.1, threshold = 0.5):
        self.lr = lr
        self.n = int(n)
        self.threshold = threshold

    def get_params(self, deep = True):
        return {"lr": self.lr, "n" : self.n, "n" : self.threshold}

    def set_params(self, **parameters):
        for parameter, value in parameters.items():
            setattr(self, parameter, value)
        return self

    def sigmoid(self, x, weight):
        z = np.dot(x, weight)
        return 1 / (1 + np.exp(-z))
```

```
    def fit(self, x, y):
        self.intercept = np.ones((x.shape[0], 1))
        self.x = np.concatenate((self.intercept, x), axis=1)
        self.weight = np.zeros(self.x.shape[1])
        self.y = y
        for i in range(self.n):
            sigma = self.sigmoid(self.x, self.weight)
            grad = np.dot(self.x.T, (sigma - self.y)) / self.y.shape[0]
            self.weight -= self.lr * grad

    def predict(self, x_new):
        test_intercept = np.ones((x_new.shape[0], 1))
        x_new = np.concatenate((test_intercept, x_new), axis=1)
        result = self.sigmoid(x_new, self.weight)
        result = result >= self.threshold
        y_pred = np.zeros(result.shape[0])
        for i in range(len(y_pred)):
            if result[i] == True:
                y_pred[i] = 1
            else:
                continue

        return y_pred
```

Подберем параметры для логистической регрессии через GridSearch:

In [ ]:
```
parameters = { "model__n": [100, 200], "model__lr": np.linspace(0.1, 0.00001, 5)}
logreg = Pipeline(steps=[
    ("preprocess", scaler),
    ("model", MyLogisticRegression())
])
gs = GridSearchCV(estimator=logreg, param_grid=parameters, scoring='accuracy')
gs.fit(X_train, y_train)
bp = gs.best_params_
```

Обучим нашу модель и сделаем предсказания, оценим качество:

In [ ]:
```
logreg = Pipeline(steps=[
    ("preprocess", scaler),
    ("model", MyLogisticRegression(n=bp["model__n"], lr=bp["model__lr"]))
])
logreg.fit(X_train, y_train)
preds = logreg.predict(X_test)
print_res(preds, y_test)
```

```
Accuracy: 0.8436441982148354
Recall: 0.7112068965517241
Precision: 0.48162162162162164
Roc-auc: 0.7889562406479189
Confusion matrix:
[[31178  4795]
 [ 1809  4455]]
```

Сохраним обученную модель:

In [ ]:
```
pickle.dump(logreg, open('logreg.pkl', 'wb'))
```

Обучим и сделаем предсказания с помощью Логистической регрессии в sklearn:

In [ ]:
```
logreg_sk = Pipeline(steps=[
```

```
    ("preprocess", scaler),
    ("model", LogisticRegression(solver= 'liblinear'))
])
logreg_sk.fit(X_train, y_train)
preds = logreg_sk.predict(X_test)
print_res(preds, y_test)
```

```
Accuracy: 0.8456092999029287
Recall: 0.7145102971230939
Precision: 0.49135135135135133
Roc-auc: 0.7916820634907475
Confusion matrix:
[[31171  4705]
 [ 1816  4545]]
```

Как видно, качество метрик моей логистической регрессии и из sklearn приблизительно одинаковы

KNN

In [ ]:
```
class MyKNNClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, k = 5, metric = 'euclidian'):
        self.k = k
        self.metric = metric

    def fit(self, X, y):
        self.x = X
        self.y = y
        return self

    def get_params(self, deep = True):
        return {"k": self.k, "metric" : self.metric}

    def set_params(self, **parameters):
        for parameter, value in parameters.items():
            setattr(self, parameter, value)
        return self

    def predict(self, X_test):
        def euclidian(s):
            dist = np.sum(((self.x - s) ** 2), axis = 1)
            return dist

        def manhatten(s):
            dist = np.sum(abs(self.x - s), axis = 1)
            return dist

        if self.metric == 'euclidean':
            dist = np.apply_along_axis(euclidian, 1, X_test)
        else:
            dist = np.apply_along_axis(manhatten, 1, X_test)

        s_dist = np.argsort(dist, axis = 1)
        n_dist = s_dist[:, :self.k]
        preds = []
        for row in n_dist:
            pred = np.argmax(np.bincount(self.y[row]))
            preds.append(pred)
        return np.array(preds)
```

Так как моя реализация KNN работает очень долго на полном датасете, то я беру лишь

часть этого датасета:

In [ ]:
```
tr, te = 3000, 1000
X_train_small = X_train[:tr]
y_train_small = y_train[:tr]
X_test_small = X_test[:te]
y_test_small = y_test[:te]

knn_k = [1, 2, 3, 4]
knn_metrics = ["euclidean", "manhatten"]
```

Подберем параметры для нашей модели KNN:

In [ ]:
```
parameters_knn = {"model__k": knn_k, "model__metric": knn_metrics}
knn = Pipeline(steps=[
    ("preprocess", scaler),
    ("model", MyKNNClassifier())
])
gs_knn = GridSearchCV(knn, parameters_knn, scoring='accuracy')
gs_knn.fit(X_train_small, y_train_small)
bp = gs_knn.best_params_
```

Также обучим ее и сделаем предсказания, оценим качество:

In [ ]:
```
knn = Pipeline(steps=[
    ("preprocess", scaler),
    ("model", MyKNNClassifier(k = bp["model__k"], metric = bp["model__metric"]))
])
knn.fit(X_train_small, y_train_small)
preds = knn.predict(X_test_small)
print_res(preds, y_test_small)
```

```
Accuracy: 0.806
Recall: 0.6190476190476191
Precision: 0.18660287081339713
Roc-auc: 0.7188087614981958
Confusion matrix:
[[767 170]
 [ 24  39]]
```

In [ ]:
```
pickle.dump(knn, open('knn.pkl', 'wb'))
```

Проделаем то же самое для модели из sklearn:

In [ ]:
```
knn_sk = Pipeline(steps=[
    ("preprocess", scaler),
    ("model", KNeighborsClassifier(n_neighbors = 4, metric = "minkowski"))
])
knn_sk.fit(X_train_small, y_train_small)
preds = knn_sk.predict(X_test_small)
print_res(preds, y_test_small)
```

```
Accuracy: 0.794
Recall: 0.5283018867924528
Precision: 0.1339712918660287
Roc-auc: 0.6685860014743679
Confusion matrix:
[[766 181]
 [ 25  28]]
```

SVM

In [ ]:
```
class MySVMClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, n = 100, lr = 0.01, lam = 0.01):
        self.n = n
        self.lr = lr
        self.lam = lam

    def get_params(self, deep = True):
        return {"n": self.n, "lr": self.lr, "lam": self.lam}

    def set_params(self, **parameters):
        for parameter, value in parameters.items():
            setattr(self, parameter, value)
        return self

    def fit(self, X, y):
        self.n, self.m = X.shape
        self.X = X
        self.y = y

        self.W = np.zeros(self.m)
        self.b = 0
        for _ in range(self.n):
            for i, x in enumerate(X):
                if y[i] * (np.dot(x, self.W) - self.b) >= 1:
                    self.W -= self.lr * (2 * self.W * self.lam)
                else:
                    self.W -= self.lr * (2 * self.W * self.lam - np.dot(x, y[i]))
                    self.b -= self.lr * y[i]

        return self

    def predict(self, X):
        return np.sign(np.dot(X, self.W) - self.b)
```

Моя SVM тоже долго работает на большом датасете, так что я буду использовать маленький:

In [ ]:
```
parameters_svm = {"model__n": [50, 100], "model__lr": [0.01], "model__lam": [0.01]}
svm = Pipeline(steps=[
    ("preprocess", scaler),
    ("model", MySVMClassifier())
])
gs_svm = GridSearchCV(svm, parameters_svm, scoring='accuracy')
gs_svm.fit(X_train_small, np.where(y_train_small > 0, 1, -1))
bp = gs_svm.best_params_
```

In [ ]:
```
svm = Pipeline(steps=[
    ("preprocess", scaler),
    ("model", MySVMClassifier(n = bp["model__n"], lr = bp["model__lr"], lam = bp["m
])
svm.fit(X_train_small, np.where(y_train_small > 0, 1, -1))
preds = svm.predict(X_test_small)
print_res(np.where(preds > 0, 1, 0), y_test_small)
```

```
Accuracy: 0.838
Recall: 0.6577181208053692
Precision: 0.4688995215311005
Roc-auc: 0.7636416690983368
Confusion matrix:
[[740 111]
 [ 51  98]]
```

In [ ]:
```
pickle.dump(svm, open('svm.pkl', 'wb'))
```

In [ ]:
```
svm_sk = logreg = Pipeline(steps=[
    ("preprocess", scaler),
    ("model", SVC(gamma='auto'))
])
svm_sk.fit(X_train_small, np.where(y_train_small > 0, 1, -1))
preds = svm_sk.predict(X_test_small)
print_res(np.where(preds > 0, 1, 0), y_test_small)
```

```
Accuracy: 0.848
Recall: 0.7355371900826446
Precision: 0.4258373205741627
Roc-auc: 0.7995092093757933
Confusion matrix:
[[759 120]
 [ 32  89]]
```

Bayes

In [ ]:
```
import math

class MyNaiveBayes(BaseEstimator, ClassifierMixin):
    def __init__(self):
        self.classes = None
        self.mean = None
        self.variance = None

    def fit(self, X, y):
        self.classes = np.unique(y)
        self.mean = np.zeros((len(self.classes), X.shape[1]), dtype=np.float64)
        self.variance = np.zeros((len(self.classes), X.shape[1]), dtype=np.float64)

        for j in self.classes:
            self.mean[j] = X[j==y].mean(axis=0)
            self.variance[j] = X[j==y].var(axis=0)

        return self

    def predict(self, X):
        preds = []
        for x in X:
            probs = [1. for _ in range(len(self.classes))]
            for i in range(len(self.classes)):
                mean = self.mean[i]
                variance = self.variance[i]
                e = np.exp(- (x - mean)**2 / (2 * variance))
                probs[i] = np.prod((1 / ((2 * math.pi) * variance)**0.5) * e)
            preds.append(self.classes[np.argmax(probs)])
        return preds
```

In [ ]:
```
bayes = Pipeline(steps=[
    ("preprocess", scaler),
```

```
    ("model", MyNaiveBayes())
])
bayes.fit(X_train, y_train)
preds = bayes.predict(X_test)
print_res(preds, y_test)
```

```
Accuracy: 0.7139001349527665
Recall: 0.40873373695736187
Precision: 0.6860540540540541
Roc-auc: 0.6500072413587678
Confusion matrix:
[[23807  2904]
 [ 9180  6346]]
```

In [ ]:
```
pickle.dump(bayes, open('bayes.pkl', 'wb'))
```

In [ ]:
```
bayes_sk = Pipeline(steps=[
    ("preprocess", scaler),
    ("model", GaussianNB())
])
bayes_sk.fit(X_train, y_train)
preds = bayes_sk.predict(X_test)
print_res(preds, y_test)
```

```
Accuracy: 0.7424769751639557
Recall: 0.43831981196451586
Precision: 0.6249729729729729
Roc-auc: 0.6594483940020872
Confusion matrix:
[[25579  3469]
 [ 7408  5781]]
```

In [ ]:
```

```

3. Результаты работы моделей

Эти таблицы не являются точными, так как для моделей KNN и SVM использовался уменьшенный набор данных из-за медленной скорости работы.

Для удобного сравнения я разделил полученные результаты на таблицы по каждой метрике:

Accuracy:

	Моя модель	Модель из sklearn
Logistic Regression	0.8436	0.8456
KNN	0.806	0.794
SVM	0.838	0.848
Naive Bayes	0.7139	0.7424

Recall:

	Моя модель	Модель из sklearn
Logistic Regression	0.7112	0.7145
KNN	0.6190	0.5283
SVM	0.6577	0.7355
Naive Bayes	0.4087	0.4383

Precision:

	Моя модель	Модель из sklearn
Logistic Regression	0.4816	0.4913
KNN	0.1866	0.1339
SVM	0.4688	0.4258
Naive Bayes	0.6860	0.6249

Roc-auc score:

	Моя модель	Модель из sklearn
Logistic Regression	0.7889	0.7916
KNN	0.7188	0.6685
SVM	0.7636	0.7995
Naive Bayes	0.6500	0.6594

4. Выводы

По полученным результатам моих моделей можно увидеть, что с задачей лучше всего справляется логистическая регрессия, поскольку по всем метрикам, кроме precision она показывает наилучшее качество. Но для задачи прогнозирования дождя показатели recall нам важнее, чем precision, так как нам важнее не допустить ошибку 2 рода, то есть предсказать, что дождя не будет, когда на самом деле он будет, нежели допустить ошибку 1 рода: предсказать дождь, при том, что на самом деле дождя не будет.

В случае с моделями sklearn лучше всего с задачей справляется SVM, поскольку он лидирует по всем метрикам, кроме precision, но Логистическую регрессию также можно использовать, так как ее показатели по recall и accuracy не сильно ниже, чем показатели SVM.