任务	逻辑回归模型	组别
2	Machine Learning	#3

机器学习报告

小组#3

小组成员：帅灿宇 20354240

万俊麟 20354123

廖逸霖 20354079

成员贡献度：1:1:1

任务完成概况：

1 描述10折交叉验证对数据集的处理。

2 使用多项式线性回归模型拟合出最佳的学习器。

3 描述训练模型所使用的算法，推导训练模型中的数学公式。

4 分析模型训练结果，包括训练误差和测试误差

5 模型训练结果

（1） 得出的线性模型

构成 y 的各项系数	常数 C	x_1	x_2	x_3	x_1^2
	5.06	0.0531	3.90×10^{-5}	3.02×10^{-3}	-1.12×10^{-4}
	x_1x_2	x_1x_3	x_2^2	x_2x_3	x_3^2
	1.10×10^{-3}	-3.80×10^{-5}	6.40×10^{-3}	8.20×10^{-5}	4.20×10^{-5}

（2）对应的均方误差及 R^2

得到的训练样本 $MSE=0.810581$， $R^2=0.971148$；

得到的测试样本 $MSE=0.925924$， $R^2=0.965510$

- jupyter测试接口**用法：输入：**估计器，(n, 4)np矩阵

  输出：(n, 1)预测值，均方差

（拟合的学习器代码及测试见附件）

目录

1 数据处理

1.1 模型评估

在利用现有数据训练学习器时，需要对训练得到的多个模型进行择优选择，通常可以通过实验测试来对学习器泛化误差进行评估并进而做出选择。

为达到择优的目的，在使用训练集数据获得学习器模型后，我们需要使用一个“测试集”来测试模型的拟合程度，测试集也必须是符合真实情况的数据，因此需要利用现有数据来划分出测试集和训练集数据。需要注意的是，训练集与测试集的数据必须互斥，这样才能达到更好的泛化能力。类似的评估方法有留出法，交叉验证法及自助法，在此问题处理中我们使用交叉验证法。

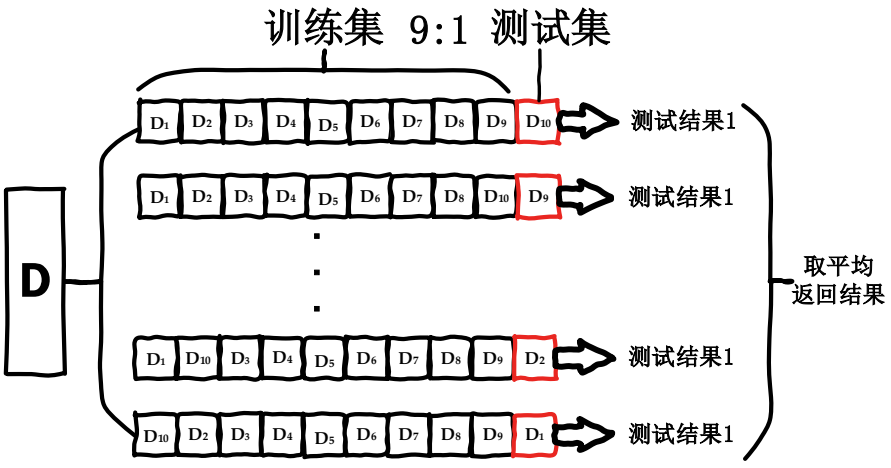
1.2十折交叉验证法

交叉验证法，也叫循环验证法，是对数据的处理即先将数据集D划分为k个大似的互斥子集，即

$$D = D_1 \cup D_2 \cup ... \cup D_k$$

再在划分出的子集中取得k-m个子集作为训练集，余下的m个子集作为测试集，这样就能得到k组不同的训练集及测试集的组合，最后取k组测试结果的均值。采用交叉验证法可以充分利用现有数据来训练模型，也可以通过不同的数据组合来达到更好的模型泛化效果，而交叉验证法的评估结果的稳定性和保真性在很大程度上取决于划分的比例即k的取值

在对此任务处理时，我们使用10折交叉验证法，即取k=10，划分出10组测试集对训练集的比例为9:1的数据集来对训练得到的学习器进行评估选择。

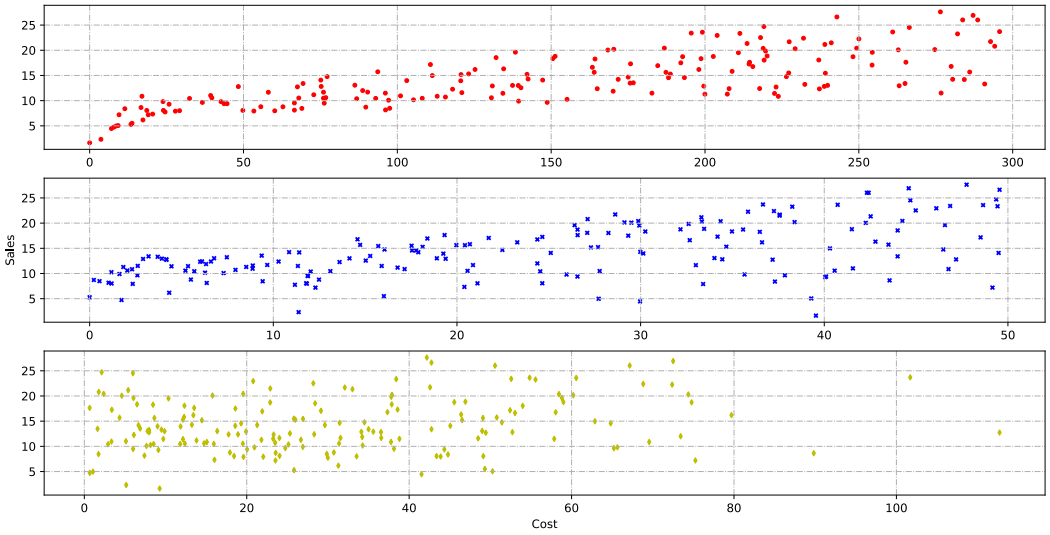


2 线性模型

2.1 单变量分析

散点图

我们首先对三个变量对销量的影响分别绘制得出以下散点图：



对应得到的系数矩阵如下:

	TV	radio	Newspaper	Sales
TV	1.000000	0.042934	0.021903	0.775351
radio	0.042934	1.000000	0.372282	0.563953
newspaper	0.021903	0.372282	1.000000	0.217399
sales	0.775351	0.563953	0.217399	1.000000

由散点图可以直观看出“TV”，“radio”的广告投放预算对销量的影响大致呈线性关系，但“newspaper”相对就比较分散， 与销量没有明显的联系，在得出的系数矩阵数据中也可以看出。

2.2 多元线性回归模型

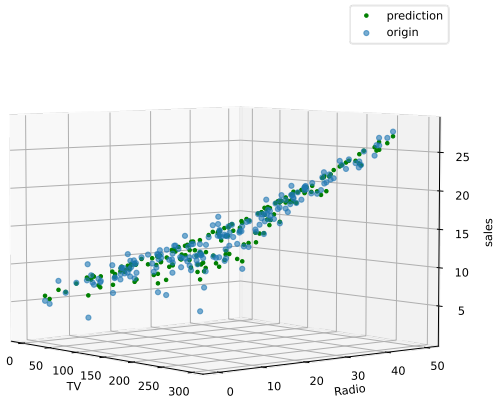
现对商品投放广告和销售数据中，将在“TV”、“radio”、“newspaper”三种渠道投放广告的预算分别设为x_1,x_2,x_3，对应的商品销量“sales”设为y。

2.2.1多元一次线性回归模型

首先我们将三个变量 x_1, x_2, x_3 以一次项的形式拟合出

$$y = \omega_1 x_1 + \omega_2 x_2 + \omega_2 x_2$$

这样的多元一次线性回归模型，通过最小二乘法，得到模型的**拟合优度**（亦“判断系数”） $R^2 < 0.9$，拟合优度小，未达到预期拟合效果。由数据分析得到此结果的原因在于“newspaper”对应的变量与销量关联度低，我们先画出该结果下“TV”，及“Radio”影响下的销量关系如下：



由绘制出的单变量散点图也可以大致看出“newspaper”对应的数据与销量之间并没有明显的联系，因此若将 x_3 代入线性模型中其对应的系数过小，若直接用一次线性回归建模则会让拟合优度变低，因此我们考虑选择使用**多元二次线性回归**来进行再次拟合优化。

2.2.2 多项式线性回归

多项式线性模型拟合，即三种渠道对应的广告投放预算值与商品的销售量不仅是简单的一次线性关系，亦或是不同商品广告投放预算之间不再是互不影响，而是会有某种联系影响到商品的总销售量。

由此我们通过将不同变量组合以及由一次项转为多次项进行求解，此时变量则有 $x_1x_2, x_1x_3, ..., x_1^2, ...$ 等多个二次项变量，则需要求解的新训练模型为：

$$y = \omega_1 x_1 + ...\omega_k x_1^a x_2^b x_3^c + ...$$

在多种多项式的组合训练中，我们最终选择以

$$1,\ x_1,\ x_2,\ x_3,\ x_1^2,\ x_1x_2,\ x_1x_3,\ x_2^2,\ x_2x_3,\ x_3^2$$

十个变量组合的线性回归模型拟合得出最优模型。

为便于求解我们将这十组多项式用向量矩阵：

$$t=[1,\ x_1,\ x_2,\ x_3,\ x_1^2,\ x_1x_2,\ x_1x_3,\ x_2^2,\ x_2x_3,\ x_3^2]$$

表示，最后通过最小二乘法求得最终模型。

3 模型训练使用的算法

在训练模型时我们使用的损失函数为均方误差：

$$MSE=\frac{1}{m}\sum_{i=1}^{m}(y_i-\boldsymbol{\omega}^T\boldsymbol{t}_i)^2$$

该值越小，说明拟合误差越小，拟合效果越好，所以我们的算法需要使该误差最小，即目标函数为

$$\min MSE=\min\sum_{i=1}^{m}(y_i-\boldsymbol{\omega}^T\boldsymbol{t}_i)^2$$

对于其代表的函数，可写成

$$L(\boldsymbol{\omega}^T,\boldsymbol{t}_i,y_i)=\min\sum_{i=1}^{m}(y_i-\boldsymbol{\omega}^T\boldsymbol{t}_i)^2=\min\left\|\boldsymbol{y}-\boldsymbol{\omega}^T\boldsymbol{X}\right\|_2^2=\min(\boldsymbol{y}-\boldsymbol{\omega}^T\boldsymbol{X})^T(\boldsymbol{y}-\boldsymbol{\omega}^T\boldsymbol{X})$$

其中列向量 $\boldsymbol{y}=(y_1;y_2;\cdots;y_m)$，$\boldsymbol{\omega}$ 包含 $\omega_1,\omega_2\cdots\omega_9,b$ 十个未知量，

$\boldsymbol{X}=\begin{pmatrix}\boldsymbol{t}_1 & 1\\ \boldsymbol{t}_2 & 1\\ \vdots & 1\\ \boldsymbol{t}_m & 1\end{pmatrix}$。利用多元函数求极值的方法，让损失函数 L 对未知量求偏导数，并解由偏导数等于0构成的方程，即可求出未知量的值，即：

$$\frac{\partial L}{\partial\boldsymbol{\omega}}=2\boldsymbol{X}^T(\boldsymbol{y}-\boldsymbol{\omega}^T\boldsymbol{X})=0$$

由于此处构成 $\boldsymbol{X}$ 的元素 $\boldsymbol{X}^T\boldsymbol{X}$ 为满秩矩阵（由于学习器第一个系数代表的元素为常数，故满秩应为9），即可解得 $\boldsymbol{\omega}$：

$$\boldsymbol{\omega}^T = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

也即我们的最小二乘算法。此处还用到了另一个系数来判断我们拟合的优度，总平方和 $SST=\sum_{i=1}^{m}(y_i-\bar{y})^2$，而拟合数据与原始数据的误差平方和 $SSE=\sum_{i=1}^{m}(y_i-\boldsymbol{\omega}^T\boldsymbol{t}_i)^2$，我们力求误差小，也即 SSE 需要相对 SST 足够小，据此可利用两者的比值判断，即得到判断系数 R^2：

$$R^2=\frac{SST-SSE}{SST}=1-\frac{SSE}{SST}$$

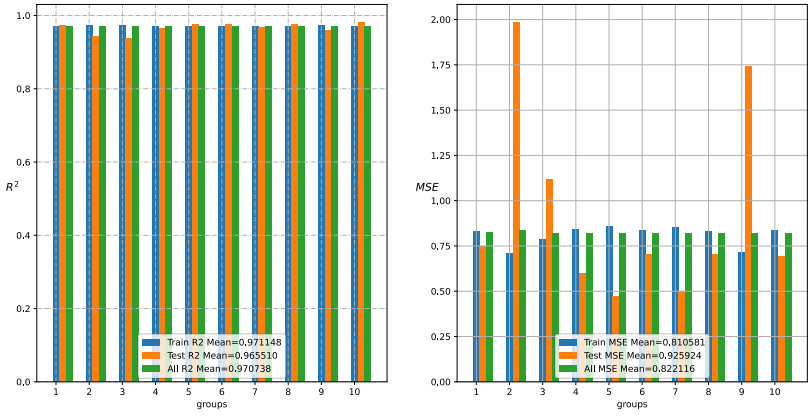
R^2 越靠近1，说明学习器的拟合效果越好。

4 模型训练结果分析

经过训练，我们得到了关于电视，和报纸三种媒体销售量 y 和预算 x_i 的关系，训练得到的训练器的各个系数如下表，此处仅取3位有效数字：

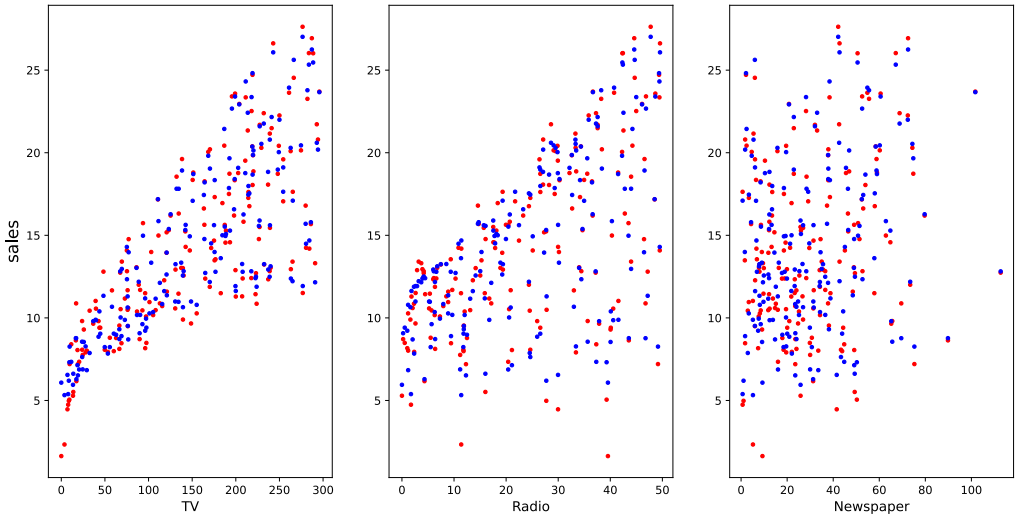
构成 y 的	常数 C	x_1	x_2	x_3	x_1^2
	5.06	0.0531	3.90×10^{-5}	3.02×10^{-3}	-1.12×10^{-4}
各项系数	x_1x_2	x_1x_3	x_2^2	x_2x_3	x_3^2
	1.10×10^{-3}	-3.80×10^{-5}	6.40×10^{-3}	8.20×10^{-5}	4.20×10^{-5}

在交叉十折验证中，我们得到的训练误差、测试误差如下图：



最终得到的训练样本 $MSE=0.810581$，得到的训练样本 $R^2=0.971148$；得到的测试样本 $MSE=0.925924$，得到的测试样本 $R^2=0.965510$。整体的 MSE 都控制在0.85以内，R^2 都控制在0.96以上，由之前的分析知我们的训练器能较好地拟合这些数据。

最后再根据所得到的学习器的预测值以及真实值进行比对，绘制出散点图如下：



由此散点图可看出训练学得的学习器拟合程度较好。

5 模型训练收获

在分析此类实际问题时，往往能获得的真实数据集有限，为了能够通过训练得到一个能充分预测出真实值的学习器，我们要对现有数据采用一些方式充分利用以训练得到拟合程度高的学习器，交叉验证法即是一种值得采用的方法，这种方法能有效利用数据来进行训练，而且该方法每次划分出的不同训练集即测试集，保证每次都能从不同的数据中获得新的学习器参数，有很好的数据泛化能力。

实际问题中一个数据通常会有很多影响因素，例如此例中某产品的销售量与在电视，收音机，报纸上的广告投入的分别会有不同程度上影响，并且也不一定只是简单的一次关系的影响，这要考虑到现实因素来进行模型的优化训练。因为在此例中我们在得出销量与三种广告投放途径的投入的一次关系模型后，分析得出模型的拟合程度虽然可以大致接受，但并未达到预期的效果，因此我们选择使用多项式的回归模型进行拟合，最终才得出最佳的拟合效果。不同的因素之间会因现实因素产生某些非线性的关系，这与现实情况也是符合的。

6 代码

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn.linear_model as lm
from sklearn.model_selection import train_test_split, cross_validate, KFold
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
import joblib
path = u"第二次作业数据集/3_train.csv"
df = pd.read_csv(path, index_col=0)
m, n = df.shape
X = df.iloc[:, 0 : n - 1]
Y = df.iloc[:, n - 1]
X_type = list(X.columns)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.1, random_state=1)
def DataVisualize(df):
    plt.figure()
    plt.subplot(311)
    plt.scatter(df.TV, df.sales,   c='r',marker='o',s=10)
    plt.grid(linestyle='-.')
    plt.subplot(312)
    plt.scatter(df.radio, df.sales,   c='b', marker='x',s=10)
    plt.grid(linestyle='-.')
    plt.ylabel("Sales")
    plt.subplot(313)
    plt.scatter(df.newspaper, df.sales,   c='y', marker='d',s=10)
    plt.xlabel('Cost')
    plt.grid(linestyle='-.')
    plt.show()

DataVisualize(df)
# 输出相关系数矩阵（最后一行为三种参数（TV，radio，newspaper）对sales的相关系数（0~1之间），越小则说明越不相关
df.corr()
# Now is (1, X1, X2, X3, X1 ^ 2, X1 * X2, X1 * X3, X2 ^ 2, X2 * X3, X3 ^ 2) 10 Features in total, so do coefs
# reg = lm.LinearRegression(copy_X=True)
reg0 = make_pipeline(PolynomialFeatures(degree=2), lm.LinearRegression(copy_X=True))
reg0.fit(X_train, Y_train)
coef = reg0[1].coef_
intercept = reg0[1].intercept_
train_score = reg0.score(X_train, Y_train)
test_score = reg0.score(X_test, Y_test)
train_mse = mean_squared_error(Y_train, reg0.predict(X_train))
test_mse = mean_squared_error(Y_test, reg0.predict(X_test))
print("Coef :", coef, "\tIntercept :", intercept)
print("Train R2 :", train_score, "\tTrain MSE :", train_mse)
```

```
print("Test R2 :", test_score, "\tTest MSE :", test_mse)
# Cross Validation Method 1
cv_results = cross_validate(reg0, X, Y, cv=KFold(n_splits=10, shuffle=True, random_state=1),
scoring=['neg_mean_squared_error','r2'], return_train_score=True, return_estimator=True)
cv_results = pd.DataFrame(cv_results)
train_mse_mean = cv_results['train_neg_mean_squared_error'].mean() * (-1.0)
test_mse_mean = cv_results['test_neg_mean_squared_error'].mean() * (-1.0)
train_score_mean = cv_results['train_r2'].mean()
test_score_mean = cv_results['test_r2'].mean()
cv_results.rename(columns={'train_neg_mean_squared_error' : 'train_negMSE',
'test_neg_mean_squared_error': 'test_negMSE'}, inplace=True)
print(cv_results.drop(columns=['estimator']),"\n")
print("Mean Train MSE: %.6f, Mean Test MSE: %.6f" % (train_mse_mean, test_mse_mean))
print("Mean Train R2: %.6f, Mean Test R2: %.6f\n" % (train_score_mean, test_score_mean))

estimators = cv_results['estimator']
coef_list = []
intercept_list = []
for i in estimators:
    coef_list.append(i[1].coef_)
    intercept_list.append(i[1].intercept_)
coef_mean = np.array(coef_list).mean(axis=0)
intercept_mean = np.array(intercept_list).mean()
print("Coef :", coef_mean, "\tIntercept :", intercept_mean, "\n")
# Cross Validation Method 2
kf = KFold(n_splits=10, shuffle=True, random_state=1)
i = 1
coef_list = []
intercept_list = []
test_score_list = []
test_mse_list = []
train_score_list = []
train_mse_list = []
all_score_list = []
all_mse_list = []
for train_index, test_index in kf.split(X, Y):
    print("%d out of KFold %d" % (i, kf.n_splits), end='\t')
    print("TEST_INDEX :", test_index)
    X_train, X_test = X.iloc[train_index, :], X.iloc[test_index, :]
    Y_train, Y_test = Y.iloc[train_index], Y.iloc[test_index]
    # reg = lm.LinearRegression(copy_X=True)
    reg = make_pipeline(PolynomialFeatures(degree=2), lm.LinearRegression(copy_X=True))
    reg.fit(X_train, Y_train)

    test_score = reg.score(X_test, Y_test)
    train_score = reg.score(X_train, Y_train)
    all_score = reg.score(X, Y)
    test_mse = mean_squared_error(Y_test, reg.predict(X_test))
    train_mse = mean_squared_error(Y_train, reg.predict(X_train))
```

```
        all_mse = mean_squared_error(Y, reg.predict(X))
        print("Train R2: %.6f, Test R2: %.6f, All R2: %.6f, Train MSE: %.6f, Test MSE: %.6f,    All MSE: %.6f" %
(
        train_score, test_score, all_score, train_mse, test_mse, all_mse))

        coef_list.append(reg[1].coef_)
        intercept_list.append(reg[1].intercept_)

        test_score_list.append(test_score)
        test_mse_list.append(test_mse)
        train_score_list.append(train_score)
        train_mse_list.append(train_mse)
        all_score_list.append(all_score)
        all_mse_list.append(all_mse)

        i += 1

coef_mean = np.array(coef_list).mean(axis=0)
intercept_mean = np.array(intercept_list).mean()

test_score_mean = np.array(test_score_list).mean()
test_mse_mean = np.array(test_mse_list).mean()
train_score_mean = np.array(train_score_list).mean()
train_mse_mean = np.array(train_mse_list).mean()
all_score_mean = np.array(all_score_list).mean()
all_mse_mean = np.array(all_mse_list).mean()
# Parameters
print("\nCoef Mean: ", coef_mean)
print("Intercept Mean: ", intercept_mean, "\n")
# Mean Scores
print("Train R2 Mean: ", train_score_mean)
print("Test R2 Mean: ", test_score_mean)
print("All R2 Mean: ", all_score_mean, "\n")
print("Train MSE Mean: ", train_mse_mean)
print("Test MSE Mean: ", test_mse_mean)
print("All MSE Mean: ", all_mse_mean)
bar_width = 0.2
N = 10
plt.subplot(1, 2, 1)
plt.grid(linestyle='-.')
plt.bar(np.arange(N), height=train_score_list, width=bar_width, label='Train R2 Mean=%.6f' %
train_score_mean)
plt.bar(np.arange(N) + bar_width, height=test_score_list, width=bar_width, align='center', label='Test R2
Mean=%.6f' % test_score_mean)
plt.bar(np.arange(N) + 2 * bar_width, height=all_score_list, width=bar_width, align='center', label='All R2
Mean=%.6f' % all_score_mean)
plt.xlabel('groups')
xt=range(0,10,1)
lb=['1','2','3','4','5','6','7','8','9','10']
```

```
plt.xticks(xt,lb)
plt.ylabel('$R^2$',rotation=0,fontsize=12)
plt.legend(loc='lower center')

plt.subplot(1, 2, 2)
plt.grid()
plt.bar(np.arange(N), height=train_mse_list, width=bar_width, label='Train MSE Mean=%.6f' %
train_mse_mean)
plt.bar(np.arange(N) + bar_width, height=test_mse_list, width=bar_width, align='center', label='Test MSE
Mean=%.6f' % test_mse_mean)
plt.bar(np.arange(N) + 2 * bar_width, height=all_mse_list, width=bar_width, align='center', label='All
MSE Mean=%.6f' % all_mse_mean)
plt.xlabel('groups')
xt=range(0,10,1)
lb=['1','2','3','4','5','6','7','8','9','10']
plt.xticks(xt,lb)
plt.ylabel('$MSE$',rotation=0,fontsize=12)
plt.legend(loc='lower center')

plt.show()


def TenTimesTenFoldsCV(X, Y, polynomial=False):
    import sklearn.linear_model as lm
    from sklearn.model_selection import cross_validate, KFold
    from sklearn.preprocessing import PolynomialFeatures
    from sklearn.pipeline import make_pipeline
    import numpy as np

    cv_results_dict = {}
    coef_dict = {}
    intercept_dict = {}
    estimator_dict = {}
    scoring_dict = {}

    if polynomial == True:
        reg = make_pipeline(PolynomialFeatures(degree=2), lm.LinearRegression(copy_X=True))
    else:
        reg = lm.LinearRegression()

    for i in range(10):
        # Cross Validation Method 1
        print("Times %d:" % (i + 1))
        cv_results = cross_validate(reg, X, Y, cv=KFold(n_splits=10, shuffle=True, random_state=i),
                                    scoring=['neg_mean_squared_error', 'r2'],
return_train_score=True,
                                    return_estimator=True)
        cv_results = pd.DataFrame(cv_results)
        cv_results_dict[i] = cv_results
```

```
        train_mse_mean = cv_results['train_neg_mean_squared_error'].mean() * (-1.0)
        test_mse_mean = cv_results['test_neg_mean_squared_error'].mean() * (-1.0)
        train_score_mean = cv_results['train_r2'].mean()
        test_score_mean = cv_results['test_r2'].mean()
        scoring_dict[i] = {'train_mse_mean': train_mse_mean, 'test_mse_mean': test_mse_mean,
                           'train_score_mean': train_score_mean, 'test_score_mean': test_score_mean}

        cv_results.rename(
            columns={'train_neg_mean_squared_error': 'train_negMSE',
'test_neg_mean_squared_error': 'test_negMSE'},
            inplace=True)
        # If print these, results will be too long to read :(
        # print(cv_results.drop(columns=['estimator']),"\n")

        print("Mean Train MSE: %.6f, Mean Test MSE: %.6f" % (train_mse_mean, test_mse_mean))
        print("Mean Train R2: %.6f, Mean Test R2: %.6f\n" % (train_score_mean, test_score_mean))

        estimators = cv_results['estimator']
        estimator_dict[i] = estimators

        coef_list = []
        intercept_list = []
        if polynomial == True:
            for e in estimators:
                coef_list.append(e[1].coef_)
                intercept_list.append(e[1].intercept_)
        else:
            for e in estimators:
                coef_list.append(e.coef_)
                intercept_list.append(e.intercept_)

        coef_mean = np.array(coef_list).mean(axis=0)
        intercept_mean = np.array(intercept_list).mean()

        coef_dict[i] = coef_mean
        intercept_dict[i] = intercept_mean

        print("Coef :", coef_mean, "\tIntercept :", intercept_mean, "\n")
    return (cv_results_dict, coef_dict, intercept_dict, estimator_dict, scoring_dict)


_, coef_dict, intercept_dict, _, scoring_dict = TenTimesTenFoldsCV(X, Y, polynomial=True)

# coef_mean = np.array(pd.DataFrame(coef_dict).mean(axis=1))
coef_mean = np.array(list(coef_dict.values())).mean(axis=0)
intercept_mean = np.array(list(intercept_dict.values())).mean()

print("Final Coef Mean :", coef_mean)
```

```
print("Final Intercept Mean :", intercept_mean)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.1, random_state=1)
reg0.coef_ = coef_mean
reg0.intercept_ = intercept_mean
# reg0.singular_ = singular_mean
Y_pred = reg0.predict(X)
print("After changing weights and bias to meaned edition:")
print("Train R2 :", reg0.score(X_train, Y_train))
print("Test R2 :", reg0.score(X_test, Y_test))
print("All R2 :", reg0.score(X, Y))
print("Train MSE :", mean_squared_error(Y_train, reg0.predict(X_train)))
print("Test MSE :", mean_squared_error(Y_test, reg0.predict(X_test)))
print("All MSE :", mean_squared_error(Y, Y_pred))
print("The rank of the variation:",reg0[1].rank_)
polynomial_items = [i.replace('X1', X_type[0]).replace('X2', X_type[1]).replace('X3', X_type[2]) for i in ['1',
'X1', 'X2', 'X3', 'X1 ^ 2', 'X1 * X2', 'X1 * X3', 'X2 ^ 2', 'X2 * X3', 'X3 ^ 2']]
finalEqStr = ""
for i in range(1, len(polynomial_items)):
    finalEqStr += "(%.6f) * %s + " % (coef_mean[i], polynomial_items[i])
finalEqStr += str(intercept_mean)
print("Final Equation : Sales =", finalEqStr, "\n")
joblib.dump(reg0, "LinearRegressor.pkl")
reg0 = joblib.load("LinearRegressor.pkl")


def tester(reg, data):
    """
    Input:
    reg: Regressor from sklearn with predict() method, e.g.
sklearn.linear_model._base.LinearRegression() or Pipeline()
    data: np.array of (n, 4), e.g. array([[X1, X2, X3, Y]])
          or list e.g. [(X1, X2, X3, Y1), (X4, X5, X6, Y2)]

    Return: (Y_pred, mse)
    Y_pred: Predicted Values of Y
    mse: Mean Squared Error Value of (Y, Y_pred)
    """
    from sklearn.metrics import mean_squared_error
    data = np.array(data)    # Ensure that input data is an np.ndarray
    m, n = data.shape
    X, Y = data[:, 0: n - 1], data[:, n - 1]
    Y_pred = reg.predict(X)
    mse = mean_squared_error(Y, Y_pred)
    return (Y_pred, mse)
# Get Data Ready For Function tester()
X1 = np.array(X_test)
Y1 = np.array(Y_test)
print(X1.shape, Y1.shape)
# Y1 should be 2-dim array of (n, 1), change it now
```

```
Y1 = Y1.reshape(Y1.shape[0], 1)
data = np.hstack([X1, Y1])
print(data.shape)
#Try tester()
Y1_pred, mse1 = tester(reg0, data)
print(Y1_pred, mse1)
Y2 = Y1_pred - Y1.reshape(-1)
loss = np.sum(Y2**2) / Y2.shape[0]
print(loss)
print(mse1==loss)

fig = plt.figure()
ax1 = fig.add_subplot(projection='3d')
ax1.scatter3D(X.iloc[:, 0], X.iloc[:, 1], Y, cmap='Blues', alpha=0.6,label='origin')
ax1.plot3D(X.iloc[:, 0], X.iloc[:, 1], Y_pred, '.g', alpha=1,label='prediction')
plt.xlabel('TV')
plt.ylabel('Radio')
ax1.set_zlabel('sales')
plt.legend(loc='best')
ax1.view_init(5, -40)
plt.show()

fig = plt.figure()
ax2 = fig.add_subplot(131)
ax2.plot(X.iloc[:, 0], Y, '.r', X.iloc[:, 0], Y_pred, '.b')
plt.xlabel('TV',fontsize=12)
plt.ylabel('sales',fontsize=15)
ax2 = fig.add_subplot(132)
ax2.plot(X.iloc[:, 1], Y, '.r', X.iloc[:, 1], Y_pred, '.b')
plt.xlabel('Radio',fontsize=12)
ax2 = fig.add_subplot(133)
ax2.plot(X.iloc[:, 2], Y, '.r', X.iloc[:, 2], Y_pred, '.b')
plt.xlabel('Newspaper',fontsize=12)
plt.show()
```