机器学习与数据挖掘大作业——实验报告

一、团队信息与分工情况

项目在整个流程中都是由小组成员共同完成，包括项目的前期调研与选题，项目的实现、调试和报告的撰写。只在具体的实现中小组成员有不同的分工，报告中相关部分也由对应成员完成。

学号	姓名	分工	贡献
21311474	张超茂	神经网络模块实现	25%
21311014	周溪石	线性回归模块的实现	25%
21311454	姚隆基	随机森林模块的实现	25%
21311145	孙惠祥	数据的预处理和模型最后的集成与评估	25%

二、选题与背景

在项目初期，我们对机器学习和强化学习的各种应用都进行了考虑，我们假设任何人体都能通过这方面的知识解决或优化，小组成员进行了头脑风暴，想出了包括但不限于以下应用：

电子商务网站用户行为分析及服务推荐、游戏用户偏好分析与活动推荐、分析用户面部特征来预测是否有疾病、鉴定网络评论是否有水军发布、房价或股价预测、用强化学习通关羊了个羊和用强化学习玩简单游戏如警察抓小偷……

但是上面的想法由于种种原因被否决，一个比较普遍的问题是找不到好的数据集，没有可靠优质的数据来源；对于与强化学习相关的任务，感觉学的不多，不是很熟悉；还有的就是感觉一些方法和课程没有关系，只是调包就能解决，即使没有学过这门课也可以做。我们希望尽可能用上学到的知识。经过反复讨论之后，我们最终选择房价预测作为我们的主题。

任何人的生活都离不开房，房地产已是我国的支柱产业之一。房价预测一直是房地产行业以及投资领域的重要任务。准确地预测房价对于购房者、开发商、政府部门和金融机构都具有重要意义。然而，房价受到众多因素的影响，例如地理位置、房屋特征、经济指标、社会环境等。传统的基于经验规则的房价估算方法已经无法满足日益复杂的市场需求。

机器学习作为一种强大的数据分析工具，已经在许多领域取得了显著的成果。将机器学习应用于房价预测任务，可以通过大规模数据的挖掘和建模，发现隐藏在数据中的模式和关联，从而提高预测准确性和效率。

所以，我们选题意义如下：

1. 提供决策支持：房价预测模型可以为购房者、开发商和金融机构等提供决策支持。购房者可以更好地了解市场行情，做出明智的购房决策；开发商可以合理定价和规划开发项目；金融机构可以准确评估风险并制定贷款策略。

2. 优化资源配置：房价预测模型可以帮助政府部门和城市规划者更好地理解市场需求和房地产趋势，从而优化土地利用、公共设施规划和城市建设。

3. 促进行业创新：通过机器学习技术的应用，房地产行业可以加速创新和数字化转型。房价预测模型还可以结合其他数据源，如社交媒体数据和舆情数据，提供更全面的市场洞察。

总之，基于机器学习的房价预测是一个具有广阔应用前景的重要课题。通过充分挖掘数据中的潜在模式和关联，提供准确的房价预测，有助于促进房地产行业的发展。为购房者、开发商、政府部门和金融机构等提供决策支持，推动城市的可持续发展和智慧化建设。

以上是选择房价预测作为我们选题的实际意义。除了对于社会的现实意义，我们也有出于自己的考量：

如今的机器学习发展日新月异，从早期的数据挖掘到传统的机器学习，再到后来的深度学习、人工智能等，机器学习发展的速度和高度已经到了令人咋舌的地步。然而千里之行始于足下，数学的发展从加减乘除开始，物理大厦的建立也是从几个基本定律开始的。因此打好基础，弄清楚轮子是怎么造的、车辆是怎么跑起来的对于后续更深入的学习有重要意义。

综上所述，我们最终决定通过房价预测这个比较简单和传统的回归任务来回顾和总结整个学期的机器学习相关知识，做一个比较完整的机器学习项目来熟悉机器学习与数据挖掘的一般流程，为后续更深入的学习打下基础。

三、模型与方法

出于上面的目的，我们决定尽可能多地用到课程中学到的知识。房价预测很显然应该用监督学习的方法，结合适当的回归算法去做。所以我们决定用以下方法来预测房价并评估我们的解决方案：

1. 对数据进行预处理

数据集采用 kaggle 上一个比赛提供的数据集：House Prices - Advanced Regression Techniques | Kaggle。该数据集描述了爱荷华州埃姆斯市住宅的几乎各个方面，有 79 个属性，接近 3000 行数据，并且每个数据都有标签说明房屋的价格，这就是我们预测的目标。该数据集属性值足够多，可供我们去寻找属性与标记之间的关系，数据项熟练工业很多，足够我们从中学习出比较好的模型。并且数据集的每个属性都有对应的说明，标注十分清楚，缺失值和 0 值也比较少。总体来说是比较优秀的数据集。

对该数据集进行简单观察后发现，该数据集不能直接进行使用。数据中有缺失值和异常值，并且有大量属性的值是字符串而不是直接可用于训练的数字。因此，训练模型之前首先要对数据进行预处理，解决上面出现的问题。

具体的处理包括以下几种方法：

1.1 将字符串类型替换成数字

这里尝试了两种方法，一种是值留下是数字的属性，其他非数字属性全部删除；另一种是用 factorize 方法把非数字的属性根据该属性的个数转换成对应的数字。两种划分方法都进行了尝试，最终选择了第二种方法，因为这样能够保留更多的特征，为模型的训练提供更多信息。

```
# 1. 只挑出是数字的项
# train_num = train_df.select_dtypes(include = ['float64', 'int64'])
# print(train_num.shape)

# 2. 把字符类型的变换成数字
for column in train_df:
    if train_df[column].dtype =='object':
        train_df[column] = pd.factorize(train_df[column])[0]+1
train_num = train_df.copy()
```

1.2 用平均值替换异常值

数据中有两种不希望看到的值，一种是缺失值，一种是异常的 0 值。对数据集进行粗略观察后发现这两种数据出现情况较多，于是对某一列都进行替换操作。先 replace 方法把所有的 0 值替换成空值，然后用 fillna 方法将空值替换成该列的均值。

```
print(train_num.isnull().sum())
#将可能的0值替换成空值
train_num = train_num.replace(0, np.NAN)
#替换空值为平均值
train_num = train_num.fillna(train_num.mean(skipna=True))
print('最终训练集数据：\n',train_num)
```

对于训练集，也做同样的操作;

```
# 测试集做同样的操作
test_df = pd.read_csv('test.csv')
test_df = test_df.drop('Id',axis=1)

# 1. 只挑出是数字的项
# train_num = train_df.select_dtypes(include = ['float64', 'int64'])
# print(train_num.shape)

# 2. 把字符类型的变换成数字
for column in test_df:
    if test_df[column].dtype =='object':
        test_df[column] = pd.factorize(test_df[column])[0]+1
test_num = test_df.copy()

#将可能的0值替换成空值
test_num = test_num.replace(0, np.NAN)
#替换空值为平均值
test_num = test_num.fillna(test_num.mean(skipna=True))
print('最终测试集数据：\n',test_num.shape)
```

1.3 降维

由于该数据集有 79 个属性，如果用所有的属性去预测，对计算资源的消耗可能会非常大，计算时间会很长。为了防止维度灾难，所以要对数据进行降维。

这里采用 PCA 的方式进行降维。首先要对样本做中心化处理：对每一行求均值，然后把这一行减去一行全为这个均值的矩阵，这样就能得到中心化后的样本。接着求训练集的协方差矩阵，再用 np.linalg.eig 方法对协方差矩阵做特征值分解，得出特征值和特征值矩阵。最后选取降维后的维数 k，根据特征值计算方差的贡献率，先将所有特征值按照降序进行排序。

接着要根据需求选取贡献率，这里尝试过 99%和 99.9%两种累计贡献。大于 99%最后结果是 2 维，大于 99.9%是 6 维。我们一开始采用了 6 维的结果。在后续的模块完成后又对 2 维的结果进行了尝试，发现 6 维的结果并没有比 2 维更好，6 维的计算时间反而比 2 维更多一些，所以最终我们采用 2 维的结果进行训练。

选取 k 值后就可以得到投影矩阵 W，用 W 乘原本训练集就可以把原本数据集降到 2 维。对于训练集，也可以直接用这个投影矩阵来降维，因为我们要保持在同一个特征空间中进行训练和预测。

```
# 对数据进行降维
# 去除标签值
X = np.array(train_num.drop('SalePrice',axis=1))
print(X.shape)
X = X.T

# 对样本进行中心化
for i in range(len(X)):
    tmp = np.mean(X[i])
    avg = np.full((1, len(X[i])), tmp)
    X[i] = X[i] - tmp

# 求协方差矩阵
cov = np.dot(X, X.T)
# print('协方差矩阵为：\n',cov)

# 做特征值分解
vals,vecs = np.linalg.eig(cov)
# print("该矩阵的特征值:",vals)
# print("该矩阵的特征向量:",vecs)

# 选取降维后的维数k
idx = np.argsort(-vals)
sortvals = vals[idx]

vsum = np.sum(vals)
tsum = 0
for i in range(len(sortvals)):
    tsum += sortvals[i]
    if tsum/vsum > 0.99:        # 改成>0.999结果就有6维
        k = i+1
        break
print('降维后的维数为：\n',k)
```

```
# 构建投影矩阵
c = [(vals[i], vecs[i]) for i in range(len(vals))]
sorted(c)
# print('排序后特征向量为：\n',c)

W = np.array([c[i][1] for i in range(k)])
# print('投影矩阵：\n',W)

# 得到降维后数据集
tmpY = np.dot(W, X)
X_redu = tmpY.T
print('降维后的训练集形状：\n', X_redu.shape)
```

输出结果为：

```
(1460, 79)
降维后的维数为：
 2
降维后的训练集形状：
 (1460, 2)
```

2. 使用不同的模型来预测房价

由于本数据集的预测任务比较清晰简洁，我们小组提出可采用多种模型完成预测任务，并采用异质集成的方法将各个个体学习器得到的预测结果结合起来。本次任务中，根据数据集特征以及小组特征，我们选择了三种模型分别对房价进行预测，分别是线性回归模型、随机森林模型、以及神经网络模型

2.1 线性回归模型

对于线性回归模型，刚开始我们将其想的过于简单，于是直接使用降维后的特征和原始的房价进行预测。结果在实际训练的过程中，预测的权重值总是无法收敛：

```
第 0 次迭代后的w: [181.9201958904108, 3233.213889157668, -10322.017454156647]
第 1 次迭代后的w: [362.65947158493117, -156786.98613967496, 560397.8842472542]
第 2 次迭代后的w: [543.2180080037668, 8316866.7037393125, -30819865.77466578]
第 3 次迭代后的w: [723.5959858856482, -451103631.2023858, 1691578794.6822515]
第 4 次迭代后的w: [903.793585825833, 24652014775.14814, -92794855091.17517]
第 5 次迭代后的w: [1083.810986096988, -1350452507301.1047, 5089571888979.911]
第 6 次迭代后的w: [1263.6484339672425, 74036047665670.88, -279135371017090.9]
第 7 次迭代后的w: [1443.3014823070862, -4059896748452478.5, 1.5308791320741414e+16]
第 8 次迭代后的w: [1623.0299246898987, 2.2264925309541123e+17, -8.39584685170655e+17]
第 9 次迭代后的w: [1790.8992996898987, -1.2210642734840848e+19, 4.60455152841687e+19]
第 10 次迭代后的w: [2552.271299689899, 6.696676697204628e+20, -2.5252822076741177e+21]
```

可以看出，迭代后的权重值 w 呈倍数增长并且来回波动，无法收敛。

```
第 172 次迭代后的w: [4.2406419264155934e+284, 3.661827384808767e+302, -1.3808588508804552e+303]
第 173 次迭代后的w: [-2.8079745066669404e+286, -inf, inf]
第 174 次迭代后的w: [nan, nan, nan]
第 175 次迭代后的w: [nan, nan, nan]
```

并且在迭代了一百七十多轮后，权重值甚至变得无穷大，说明了无法直接使用原始数据进行训练。

刚开始我们认为是学习率过大，又或者是批量梯度下降法收敛速度较慢。在尝试了较小的学习率，以及将模型调整为随机梯度下降法和 mini-batch 梯度下降法后，仍然无法收敛，于是我们认为是数据本身的问题：降维后的各个特征之间的量级相差较大，且房价本身的数量级较大，导致了在训练的过程中容易出现来回波动、发散的现象。

因此，我们决定对数据进行归一化处理：

对于标签值（即房价），使用 min-max 标准化进行处理。即采用房价的最大值和最小

值之间的差值，将各个房价归一化到 0~1 的区间内，如下图所示：

```
# min-max 标准化
y2 = np.array(train_num['SalePrice']).reshape(-1,1)       # 标签
Y_mean = min(y2[:,0])
Y_std = max(y2[:,0]) - min(y2[:,0])
y2 = (y2 - Y_mean) / Y_std      # 归一化
```

对于特征值，使用 z-score 标准化进行处理。即通过各个特征的均差和标准差，将各个特征值归一化到-1~1 的区间内，如下图所示：

```
# z-score 标准化
X_mean = np.mean(x2, axis=0)        # 均差
X_std = np.std(x2, axis=0)          # 标准差
x2 = (x2 - X_mean) / X_std          # 归一化
```

将特征值和标签值进行标准化处理后，就可以代入模型进行训练了。在训练过程中，我们分别尝试了批量梯度下降法、随机梯度下降法、mini-batch 梯度下降法后，发现使用批量梯度下降法的效果最好，故采用该方法进行训练，如下图所示：

```
# 批量梯度下降
for i in range(epoch):
    temp = 0
    temp_w=[0.0, 0.0, 0.0]
    for j in range(X_redu.shape[0]):
        temp = w[0] + w[1]*x2[j][0] + w[2]*x2[j][1] - y2[j][0]
        temp_w[0] += temp / len(x2)
        temp_w[1] += x2[j][0] * temp / len(x2)
        temp_w[2] += x2[j][1] * temp / len(x2)
    for k in range(len(w)):
        w[k] -= eta * temp_w[k]
```

值得注意的是，由于在训练之前对训练数据进行了归一化处理，因此训练得到的权重值并不是原始数据对应的权重值。要想得到归一化前的数据对应的线性回归模型，还需要对权重 w 进行还原。

由于是线性模型，权重值和特征数据一一对应，因此对权重值进行还原时，只需要执行归一化时的相反操作即可。对于特征值对应的权重值，只需要将权重值乘以相应特征值得到标准差再加上特征值的均值即可；对于偏移量 w0，其与房价有关，只需要将其乘以房价的标准差再加上房价的最小值即可。如下图所示：

```
# 还原w
w[0] = w[0] * Y_std + Y_mean
for i in range(1,len(w)):
    w[i] = w[i] * X_std[i-1] + X_mean[i-1]
```

还原前后的权重值 w 如下图所示：

```
w: [0.20281506589480247, 0.015928391754753333, -0.056846710322820654]
w: [180947.12895084725, 2.3074131350167257, -13.135382159539313]
```

至此，线性回归模型的训练就完成了。

至于对于模型的测试部分，使用数据集中自带的测试集进行测试。由于训练前我们对训

练数据集进行了降维，因此测试前也需要对测试集进行相同的降维操作。

```
# 得到降维后测试数据集Y
tmp_test = np.dot(W,np.array(test_num).T)
test_redu = tmp_test.T
test_redu = np.c_[np.ones(len(test_redu)),test_redu]
```

在训练结束后，我们对权重值进行了还原，因此在测试过程中，不需要对测试集进行归一化处理，只需要直接与还原后的权重值求内积即可。

```
# 以下为对于测试集的测试
ans = np.dot(test_redu,w).reshape(-1,1)                                    # 预测结果
print("测试集的预测结果",ans)
```

线性模型预测出来的结果大致如下：

```
测试集的预测结果 [[187007.88133844]
 [189756.94333509]
 [186325.0255315 ]
 ...
 [188705.53336889]
 [187273.65545301]
 [186517.40838483]]
```

接下来，将预测结果与测试集的样例输出进行比较：

```
# 以下为与提交样例的比较
sample_submission = pd.read_csv('sample_submission.csv')                   # 读取提交样例
sample_ans = np.array(sample_submission['SalePrice']).reshape(-1,1)        # 读取提交样例中的标签
print("测试集的预测差值:",ans-sample_ans)
MSE = np.sum((ans-sample_ans)**2) / len(sample_submission)
print("测试集的均方误差MSE = ",MSE)
```

预测的差值及均方误差如下：

```
测试集的预测差值: [[ 17730.82884004]
 [  1998.54934632]
 [  2741.34196195]
 ...
 [-30516.89003117]
 [  2349.37579401]
 [ -1224.45827265]]
测试集的均方误差MSE =  331471249.44932884
```

由于该房价都是以十万为量级的，因此即使预测结果只有一万的偏差，导致的均方误差也会到达九位数，使得均方误差看起来较大。

为了解决该问题，可以将预测结果和样例输出进行归一化后，再计算均方误差，如下图所示：

```
ans = (ans - min(ans[:,0])) / (max(ans[:,0]) - min(ans[:,0]))
sample_ans = (sample_ans - min(sample_ans[:,0])) / (max(sample_ans[:,0]) - min(sample_ans[:,0]))
MSE = np.sum((ans-sample_ans)**2) / len(sample_submission)
print("归一化后的测试集的均方误差MSE = ",MSE)
```

最后计算出的均方误差如下：

```
归一化后的测试集的均方误差MSE =  0.027529729219235535
```

2.2 随机森林模型

对于多特征结果为连续值的房价预测，回归树无疑是一种可行的预测模型。可以直接使用非数字属性和异常值处理后的数据进行回归树的递归构建。

首先，构建叶节点类：

```
class Node:
    def __init__(self, feature_index=None, threshold=None, value=None, left=None, right=None):
        self.feature_index = feature_index  # 特征索引
        self.threshold = threshold          # 分裂阈值
        self.value = value                  # 叶子节点的预测值
        self.left = left                    # 左子树
        self.right = right                  # 右子树
```

其次，构建回归树类：

```
def __init__(self, max_depth=6):
    self.max_depth = max_depth          # 最大深度
    self.tree = None                    # 决策树
```

为了让回归树有更好的泛化能力同时降低其复杂度，所以将其最大深度设置为 6 层。

用最大方差减少作为最佳特征依据和阈值划分选择：

```
# 计算当前子集y标签的方差
def _calculate_variance(self, y):
    return np.var(y)
```

```
#选出最佳划分特征和划分阈值
def _find_best_split(self, X, y):
    #只剩一行就不分
    m, n = X.shape
    if m <= 1:
        return None, None
    #当前标签的方差
    current_variance = self._calculate_variance(y)
    #初始化划分参数
    best_variance_reduction = 0
    best_feature_index = None
    best_threshold = None
    #遍历当前剩余特征
    for feature_index in range(n):
        thresholds = np.unique(X[:, feature_index])
        #遍历当前剩余特征的特征值，枚举划分阈值
        for threshold in thresholds:
            X_left, y_left, X_right, y_right = self._split_data(X, y, feature_index, threshold)
            #边缘值不算
            if len(y_left) == 0 or len(y_right) == 0:
                continue
            #左右划分权重
            left_weight = len(y_left) / m
            right_weight = len(y_right) / m
            #当前划分能减少的差值
            variance_reduction = current_variance - (left_weight * self._calculate_variance(y_left)
                                                     + right_weight * self._calculate_variance(y_right))
            #比较
            if variance_reduction > best_variance_reduction:
                best_variance_reduction = variance_reduction
                best_feature_index = feature_index
                best_threshold = threshold

    return best_feature_index, best_threshold
```

按照得到的最佳特征和划分阈值得到左右子集，不断递归得到回归树。当子集不可再分或者到了最大深度，将这个子集的平均值作为这个节点的预测值。

带入预测集进行预测，得到单棵回归树的预测结果：

```
def _predict_sample(self, node, sample):
    #节点值不为空，即为叶节点，即预测值
    if node.value is not None:
        return node.value
    elif sample[node.feature_index] <= node.threshold:
        return self._predict_sample(node.left, sample)
    else:
        return self._predict_sample(node.right, sample)

def predict(self, X):
    return np.array([self._predict_sample(self.tree, sample) for sample in X])
```

```
预测:
 [[124109.98319328]
 [140236.48648649]
 [177793.80203046]
 ...
 [164039.46296296]
 [124109.98319328]
 [245489.67567568]]
```

进行 bootstrap 采样，得到 n 个采样得到的训练集，训练得到 n 棵树，预测得到 n 个结果，用简单平均法的得到随机森林的结果：

```
result = np.zeros((test_data.shape[0], 1))
for i in range(n):
    temp1=trees[i].predict(test_data)
    result += trees[i].predict(test_data).reshape(-1, 1)
result /= n
result = result.reshape(-1, 1)
print("\n随机森林结果：\n", result)
```

```
 随机森林结果：
  [[125142.72453705]
  [148960.40037096]
  [170096.23826032]
  ...
  [149526.85784111]
  [124843.87677684]
  [223474.90986671]]
 随机森林的MSE: 4447653456.194012
 随机森林的归一化MSE:  0.10854921585446238
```

2.3 神经网络模型

使用神经网络对房价数据进行拟合是另一种可行的预测模型。直接使用降维后的数据训练神经网络会导致梯度爆炸的问题，原因是房价标签值过大，在反向传播的途中会导致参数

更新过大，从而出现模型预测值为 nan 的现象，为此，需要先对标签进行归一化。

```
#标签归一化[-1,1]
train_y=2*(train_y-min_train_y)/Regular_train_y-1
```

对测试集的标签值处理同上；

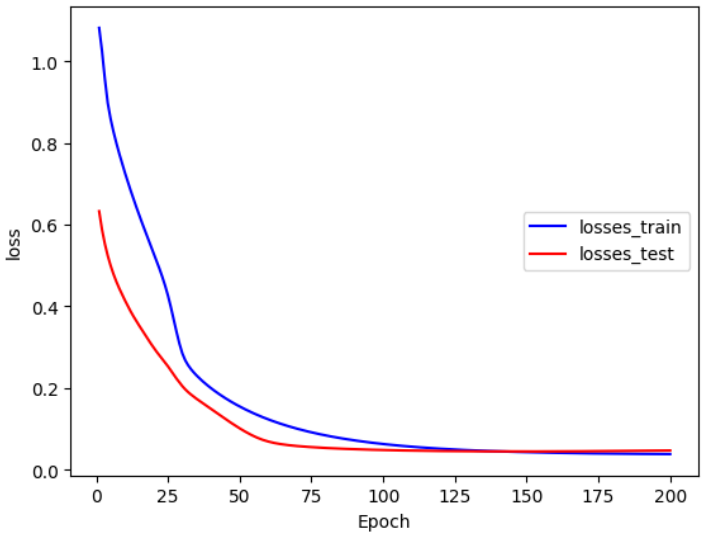
接下来便是搭建合适的神经网络模型。本项目采用 pytorch 搭建神经网络模型，设置三个全连接层，维度分别为 8,12,10，以及一个输出层。损失函数选用均方误差 MSE。激活函数方面，前两个全连接层选择 Relu，最后一个连接层则采用 Tanh。神经网络结构代码如下：

```
class Net(nn.Module):
    def __init__(self, input_dim, dim1, dim2, dim3, output_dim):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(input_dim, dim1)
        self.fc2 = nn.Linear(dim1, dim2)
        self.fc3 = nn.Linear(dim2, dim3)
        self.fc4 = nn.Linear(dim3, output_dim)
        #激活函数relu，用于在全连接层之间加入非线性变换
        self.relu = nn.ReLU()
        self.tanh = nn.Tanh()

    def forward(self, x):
        out0 = self.fc1(x)
        out1 = self.relu(out0)
        out2 = self.fc2(out1)
        out3 = self.relu(out2)
        out4 = self.fc3(out3)
        out5 = self.tanh(out4)
        out = self.fc4(out5)
        return out5,out
# 创建神经网络模型实例
net = Net(2,8,12,10,1)
```

由于房价数据集特征尺度差距较大，在训练模型的过程中容易遇到梯度消失的问题，这容易导致训练出来的模型对于不同输入都给出相同的预测结果。因此，我们将第三层全连接层的激活函数改用 Tanh 而非与前两层一样选择 Relu，目的是减缓在 0 附近的梯度消失问题。经过调整与比较，2 层 Relu 加 1 层 Tanh 的预测效果比全部采用 Relu 和全部采用 Tanh 的效果略好一些。

接下来，我们记录并绘制训练损失及测试损失，以选择合适的学习率和迭代次数



为了防止梯度爆炸和模型过拟合，这里我们采用较小的学习率 0.001，可以发现，模型收敛速度较快，因此选取 200 轮作为迭代上限是比较合适的。

最后是用测试集进行验证，由于神经网络模型经过归一化，最终预测结果并非我们预期的房价，因此要进行逆归一化操作，得到真正的房价预测值。

```
# 输入测试集，获取预测结果predict
_,predict=net(inputs_test)
# 转为numpy格式
predict=predict.detach().numpy()
```

神经网络预测结果（示例）如下

```
177177.1977443627
163875.51662672186
177823.3711960843
169948.71922512798
177001.9931539379
159914.52589078376
159914.58676194382
163527.02923535428
163523.924806191
178134.1445558521
178129.80531172754
178101.59587697778
178216.7901994593
178257.92171190266
178207.4247366894
175460.8701672523
178219.29461290193
163648.31936972222
175867.1025498208
...
178248.1823262924
175923.28663056006
176513.98036782313
```

定义 MSE 计算函数，得到最终均方误差。同样地，我们也给出归一化后的均方误差作

为参照。

```
# 评估预测结果，使用MSE均方误差
def MSE(y:np.ndarray, y_hat:np.ndarray):
    # y是真实值，y_hat是预测值
    m=len(y)
    return np.sum((y-y_hat)**2)/m

# 真实房价（重新读取）
y=np.array(pd.read_csv("sample_submission.csv").drop("Id",axis=1)).reshape(-1,1)
# 预测房价（解归一化）
y_hat=(predict+1)*Regular_test_y/2+min_test_y
# 计算均方误差
print("均方误差：",MSE(y,y_hat))
```

```
均方误差：  272694682.64528763
归一化后的均方误差: 0.051247140378380955
```

3. 异质集成上面的模型

为了让上述 3 种模型都有贡献，提高模型的稳定性，我们采用简单平均法来集成模型。具体来说就是对于每行数据，都用三种模型进行预测得到结果，然后求出它们的平均值作为最终的结果。

```
# 用简单平均法进行集成
res1 = linear_pre(test_redu,w)
res2 = network_pre(inputs_test)
res3 = randforest_pre(test_data)

result = np.mean(np.concatenate((res1,res2,res3),axis=1),axis=1)
result = result.reshape(-1,1)
print('\n最终结果为:\n',result)
```

```
最终结果为:
 [[164426.95813242]
 [174086.27192471]
 [179802.42996897]
 ...
 [173916.10393705]
 [164537.3755991 ]
 [197793.11756408]]
```

四、实验效果与模型评估

由于最终预测的结果是连续的值，所以采用均方误差对结果进行衡量。

```
# 评估预测结果，使用MSE均方误差
def MSE(y:np.ndarray, y_hat:np.ndarray):
    # y是真实值，y_hat是预测值
    m=len(y)
    return np.sum((y-y_hat)**2)/m
```

```
# 用均方误差评估模型
MSE_df = pd.read_csv('sample_submission.csv')
actual_y = np.array(MSE_df)[:, -1].reshape(-1, 1)
print('均方误差为：',MSE(actual_y, result))

# 预测结果归一化
norm_result = 2*(result-np.min(result)) / (np.max(result)-np.min(result)) -1
print('归一化后的均方误差为：', MSE(test_y, norm_result))
```

输出结果为：

```
均方误差为：  638239846.0942875
归一化后的均方误差为：  0.09193931236192225
```

上面是某一次运行的结果，数量级在 10 的 9 次方。由于采用了异质集成，所以运行结果比较稳定，基本稳定在 6.0*10e9 到 7.0*10e9 的范围。由于该房价都是以十万为量级的，因此即使预测结果只有一万的偏差，导致的均方误差也会到达九位数，使得均方误差看起来较大。所以这个结果应当是可以接受的。

这个房价预测的数据集来源是 kaggle 上的一个持续性的比赛。为了进一步评估模型的效果，我们还随机挑选了一次运行的结果进行了提交。最后可以通过比赛的测试，进入了 leaderboard，并且在榜上的排名比标准的提交基准高。我们观察了其他参赛者的结果，他们的 MSE 大多也在 10 的 9 次方这个数量级，一个获得金奖的推荐模型也在这个数量级。这进一步可以说明我们的结果误差在可接受范围内，基本可以成功地预测房价。

五、实验总结

本次大作业充分运用了这学期所学的机器学习模型及方法，包括了数据的预处理、降维、线性回归模型、随机森林模型、神经网络模型、集成学习以及模型的评估方法，并通过实际案例对机器学习流程有了更加清晰的认识

在训练各个体学习器的过程中，我们遇到了许多当时没遇到的新问题，例如模型不收敛，输出异常值，调参困难等。借助这次作业，我们增长了分析问题、解决问题的能力，收获颇丰。

完整项目见：[HousePrice：基于异质集成的房价预测（gitee.com）]