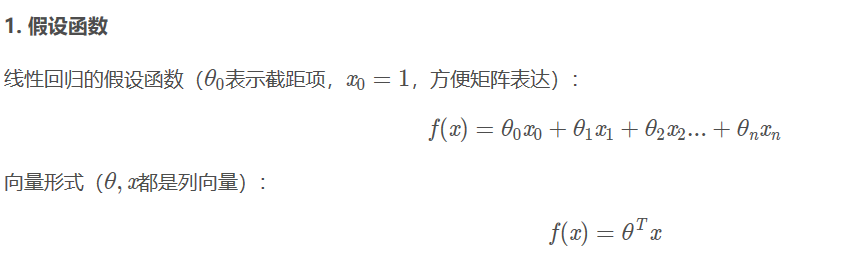
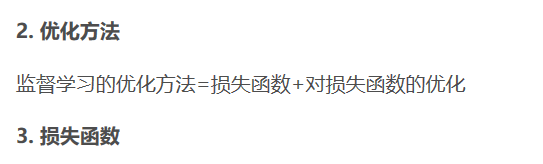
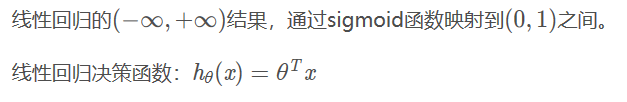
**AI课上代码实现 第八次试验 硬件一班 王倩倩 171491121**

1. **线性回归——python实现**



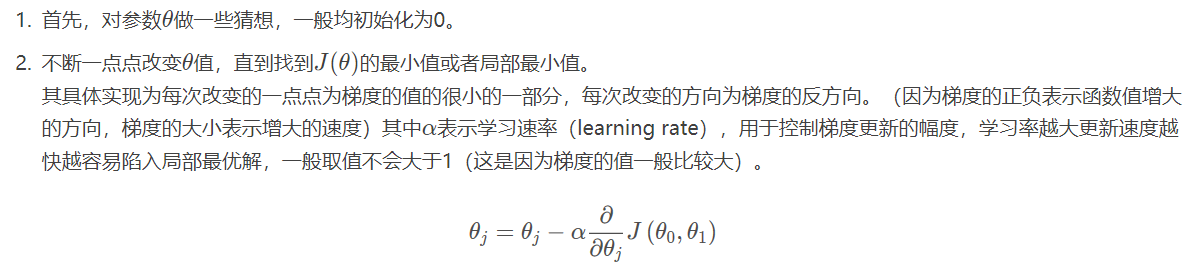




**梯度下降**

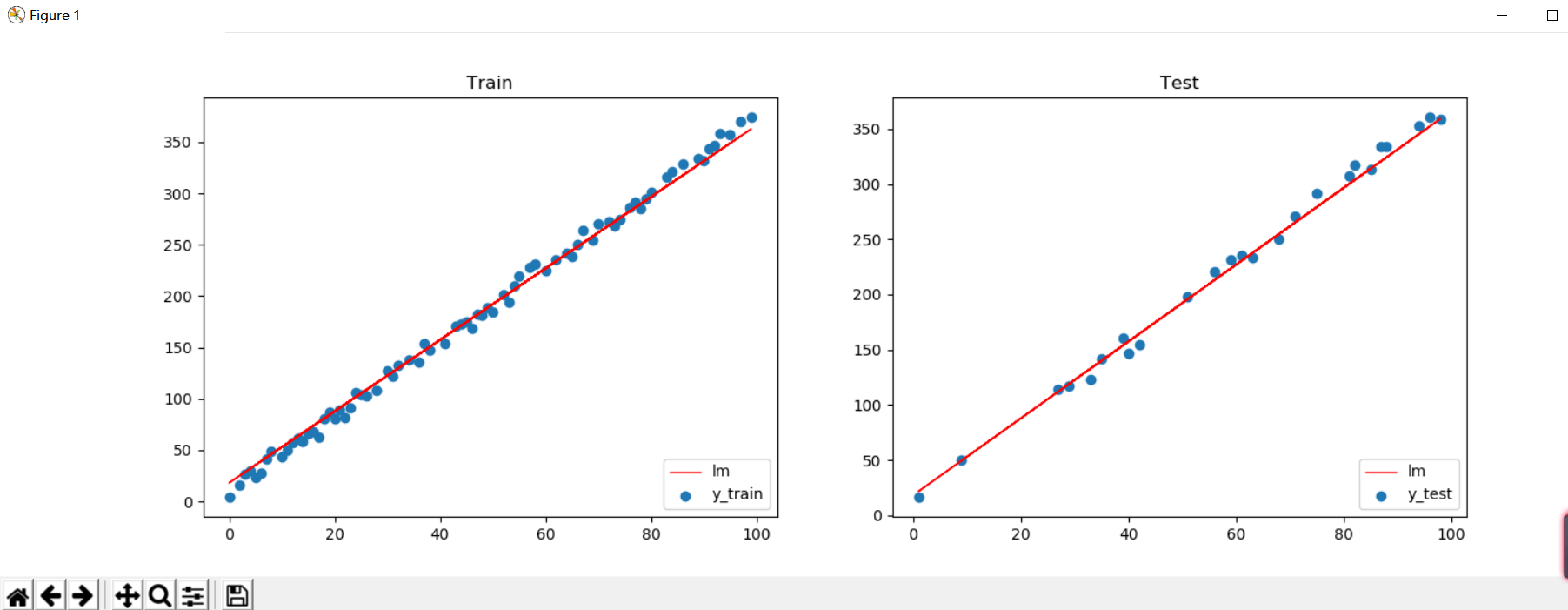
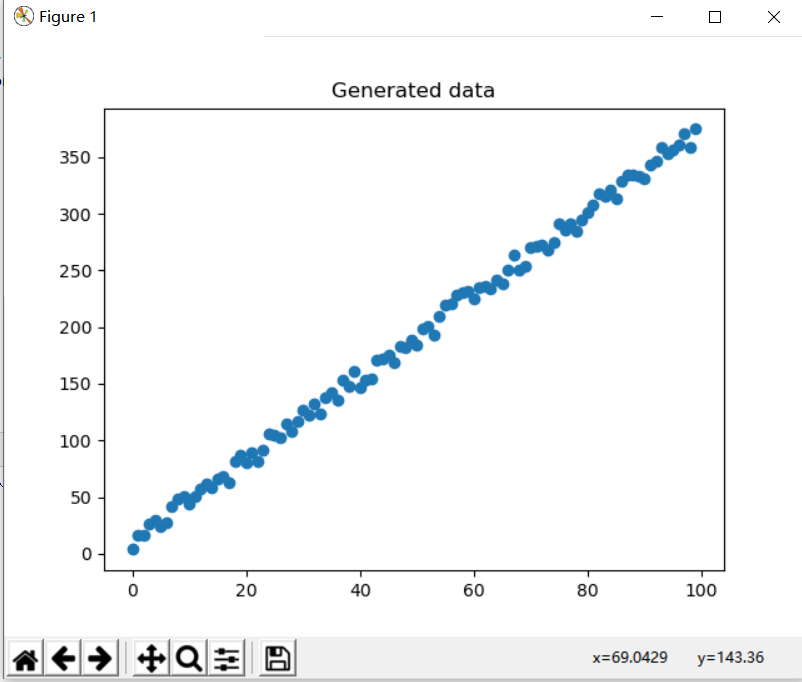
在数学上，凸优化问题有诸多解法（如正规方程，其在特征量很少的情况下有效）。其中，梯度下降算法（Gradient Descent）在机器学习领域使用最为广泛，特别是当参数量极为庞大（如神经网络）时，该算法就显得实用很多。

梯度下降算法的思想如下。



**4.代码实现**  
  
**from** argparse **import** Namespace  
**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
**import** pandas **as** pd  
**from** sklearn.linear\_model.stochastic\_gradient **import** SGDRegressor  
**from** sklearn.preprocessing **import** StandardScaler  
**from** sklearn.model\_selection **import** train\_test\_split  
**import** matplotlib.pyplot **as** plt  
  
*# Arguments*args = Namespace(  
 seed=1234,  
 data\_file=**"sample\_data.csv"**,  
 num\_samples=100,  
 train\_size=0.75,  
 test\_size=0.25,  
 num\_epochs=100,  
)  
  
*# Set seed for reproducability*np.random.seed(args.seed)  
  
*# Generate synthetic data Function***def** generate\_data(num\_samples):  
 X = np.array(range(num\_samples))  
 random\_noise = np.random.uniform(-10,10,size=num\_samples)  
 y = 3.65\*X + 10 + random\_noise *# add some noise* **return** X, y  
  
*# Generate random (linear) data*X, y = generate\_data(args.num\_samples)  
data = np.vstack([X, y]).T  
df = pd.DataFrame(data, columns=[**'X'**, **'y'**])  
df.head()  
  
*# Scatter plot*plt.title(**"Generated data"**)  
plt.scatter(x=df[**"X"**], y=df[**"y"**])  
plt.show()  
  
  
*# Create data splits*X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 df[**"X"**].values.reshape(-1, 1), df[**"y"**], test\_size=args.test\_size,  
 random\_state=args.seed)  
print (**"X\_train:"**, X\_train.shape)  
print (**"y\_train:"**, y\_train.shape)  
print (**"X\_test:"**, X\_test.shape)  
print (**"y\_test:"**, y\_test.shape)  
  
*# Standardize the data (mean=0, std=1) using training data*X\_scaler = StandardScaler().fit(X\_train)  
y\_scaler = StandardScaler().fit(y\_train.values.reshape(-1,1))  
  
*# Apply scaler on training and test data*standardized\_X\_train = X\_scaler.transform(X\_train)  
standardized\_y\_train = y\_scaler.transform(y\_train.values.reshape(-1,1)).ravel()  
standardized\_X\_test = X\_scaler.transform(X\_test)  
standardized\_y\_test = y\_scaler.transform(y\_test.values.reshape(-1,1)).ravel()  
  
  
*# Check*print (**"mean:"**, np.mean(standardized\_X\_train, axis=0),  
 np.mean(standardized\_y\_train, axis=0)) *# mean should be ~0*print (**"std:"**, np.std(standardized\_X\_train, axis=0),  
 np.std(standardized\_y\_train, axis=0)) *# std should be 1  
  
# Initialize the model*lm = SGDRegressor(loss=**"squared\_loss"**, penalty=**"none"**, max\_iter=args.num\_epochs)  
  
*# Train*lm.fit(X=standardized\_X\_train, y=standardized\_y\_train)  
  
*# Predictions (unstandardize them)*pred\_train = (lm.predict(standardized\_X\_train) \* np.sqrt(y\_scaler.var\_)) + y\_scaler.mean\_  
pred\_test = (lm.predict(standardized\_X\_test) \* np.sqrt(y\_scaler.var\_)) + y\_scaler.mean\_  
  
  
*# Train and test MSE*train\_mse = np.mean((y\_train - pred\_train) \*\* 2)  
test\_mse = np.mean((y\_test - pred\_test) \*\* 2)  
print (**"train\_MSE: {0:.2f}, test\_MSE: {1:.2f}"**.format(train\_mse, test\_mse))  
  
*# Figure size*plt.figure(figsize=(15,5))  
  
*# Plot train data*plt.subplot(1, 2, 1)  
plt.title(**"Train"**)  
plt.scatter(X\_train, y\_train, label=**"y\_train"**)  
plt.plot(X\_train, pred\_train, color=**"red"**, linewidth=1, linestyle=**"-"**, label=**"lm"**)  
plt.legend(loc=**'lower right'**)  
  
*# Plot test data*plt.subplot(1, 2, 2)  
plt.title(**"Test"**)  
plt.scatter(X\_test, y\_test, label=**"y\_test"**)  
plt.plot(X\_test, pred\_test, color=**"red"**, linewidth=1, linestyle=**"-"**, label=**"lm"**)  
plt.legend(loc=**'lower right'**)  
  
*# Show plots*plt.show()  
  
*# Feed in your own inputs*X\_infer = np.array((0, 1, 2), dtype=np.float32)  
standardized\_X\_infer = X\_scaler.transform(X\_infer.reshape(-1, 1))  
pred\_infer = (lm.predict(standardized\_X\_infer) \* np.sqrt(y\_scaler.var\_)) + y\_scaler.mean\_  
print (pred\_infer)  
df.head(3)  
  
*# Unstandardize coefficients*coef = lm.coef\_ \* (y\_scaler.scale\_/X\_scaler.scale\_)  
intercept = lm.intercept\_ \* y\_scaler.scale\_ + y\_scaler.mean\_ - np.sum(coef\*X\_scaler.mean\_)  
print (coef) *# ~3.65*print (intercept) *# ~10  
  
# Initialize the model with L2 regularization*lm = SGDRegressor(loss=**"squared\_loss"**, penalty=**'l2'**, alpha=1e-2,  
 max\_iter=args.num\_epochs)  
  
*# Predictions (unstandardize them)*pred\_train = (lm.predict(standardized\_X\_train) \* np.sqrt(y\_scaler.var\_)) + y\_scaler.mean\_  
pred\_test = (lm.predict(standardized\_X\_test) \* np.sqrt(y\_scaler.var\_)) + y\_scaler.mean\_  
  
*# Train and test MSE*train\_mse = np.mean((y\_train - pred\_train) \*\* 2)  
test\_mse = np.mean((y\_test - pred\_test) \*\* 2)  
print (**"train\_MSE: {0:.2f}, test\_MSE: {1:.2f}"**.format(  
 train\_mse, test\_mse))  
  
*# Unstandardize coefficients*coef = lm.coef\_ \* (y\_scaler.scale\_/X\_scaler.scale\_)  
intercept = lm.intercept\_ \* y\_scaler.scale\_ + y\_scaler.mean\_ - (coef\*X\_scaler.mean\_)  
print (coef) *# ~3.65*print (intercept) *# ~10*

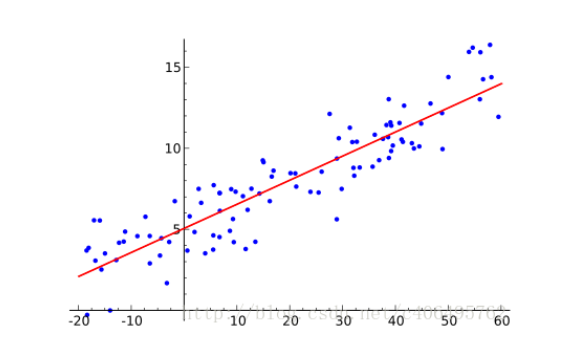
**5.运行结果**



1. **简单了解逻辑回归**

Logistic回归是众多分类算法中的一员。通常，Logistic回归用于二分类问题，例如预测明天是否会下雨。当然它也可以用于多分类问题。

假设现在有一些数据点，我们利用一条直线对这些点进行拟合(该线称为最佳拟合直线)，这个拟合过程就称作为回归，如下图所示：



Logistic回归一种二分类算法，它利用的是Sigmoid函数阈值在[0,1]这个特性。Logistic回归进行分类的主要思想是：根据现有数据对分类边界线建立回归公式，以此进行分类。其实，Logistic本质上是一个基于条件概率的判别模型

所以要想了解Logistic回归，我们必须先看一看Logistic函数。它的公式如下：

