Week 10 Statistical Data Mining

Theory and Practice

Ying Lin, Ph.D

March 27, 2020

Overview of the Topics

- Linear regression
- Logistic regression and activation function
- Model coefficient estimation and interpretation
- Loss function, gradient descent, optimization
- R/Python demo

Regression Motivation

- Linear Regression: model numerical outcome
  - Simple vs. multiple regression

$$\begin{aligned} f(x) &= \beta_0 + \beta_1 \cdot x_1 + \ldots + \beta_i \cdot x_i + \ldots + \beta_n \cdot x_n \\ &= \sum_{i=1}^{n} \beta_i x_i \\ &= \boldsymbol{\beta}^T \boldsymbol{x} \end{aligned}$$

- Logistic regression: Generalized linear regression
  - Model binary variable with two possible outcomes
  - Model multinomial categorical variables
- Key: estimation of model parameters which are coefficients in the regression model
  - Interpretation of coefficients

Key Assumptions of Linear Regression

- Linear relationship between IVs and DVs
  - Checked by scatterplots for linear or curvlinear relationship
  - Tranformation techniques such as normalizing DV
- Homoscedasticity: similar variance of error terms across space of IVs
  - Checked by scatterplot of residuals vs. predicted values
- Multivariate normality: residues are normally distributed
- Residual/error terms should be independent: no multicollinearity
  - Tested with Variance Inflation Factor (VIF) values

4/41

R-Squared

- Percent of variance that could be explained by the model

$$TSS = \sum_i (y_i - \bar{y})^2$$

$$SSE = \sum_i (f(x_i) - y_i)^2$$

$$SSM = TSS - SSE$$

$$R^2 = \frac{SSM}{TSS}$$

5/41

Ordinary Least Square (OLS)

- Minimize sum of squared error between the data points and the regression line/model
- How to estimate regression coefficients
  - Solve the model parameters analytically (closed-form solution)
  - Use the iterative optimization algorithms (Gradient Descent, Stochastic Gradient Descent, Newton's method, etc.)

Ordinary Least Square (OLS)

- How to estimate the model coefficients (in specifications) that minimizes such squared difference between real value and predicted values

$$\frac{\partial S}{\partial a} = \frac{\partial\left(\sum(y-ax-b)^2\right)}{\partial a} = 2\sum\left((y-ax-b)\cdot(0-x-0)\right)$$

the parameters of regression model $\hat{y} = ax + b$ are $a = \dfrac{n\bar{x}\,\bar{y} - \sum xy}{\left(n\bar{x}^2 - \sum x^2\right)}$ and $b = \bar{y} - a\bar{x}$

7/41

Objective Function

- Objective function: machine learning algorithms rely on minimizing or maximizing objective functions
- Loss function: the group of objective functions that are minimized; a measure of how good a prediction model does in terms of predicting the expected outcome
    - Regression loss
    - Classification loss

Regression Loss Function

- Mean squared error (quadratic loss, L2 loss)

$$MSE = \frac{\sum_{i=1}^{n} (y_i - y_i^p)^2}{n}$$

  - Root Mean Squared Error (RMSE)

$$RMSE = \sqrt{MSE}$$

- Mean absolute error (L1 loss)

$$MAE = \frac{\sum_{i=1}^{n} |y_i - y_i^p|}{n}$$

9/41

Comparison between L1 and L2 Loss Function

- MSE/RMSE is more sensitive towards outliers than MAE
  - If outliers represent anomaly that are important, then use MSE/RMSE
  - If outliers are due to erroneous data, then use MAE
  - L2 loss function works well in most of situations and consider removing outliers first
- MAE has larger and constant gradients unlike the dynamic gradient of MSE/RMSE which adjusts accordingly to the size of error
  - MAE requires an adaptive learning rate to complement

Other Loss Functions

- Huber loss (Smooth Mean Absolute Error)

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x)^2) \; for |y - f(x)| \leq \delta \\ \delta|y - f(x)| - \frac{1}{2}\delta^2 \; otherwise \end{cases}$$

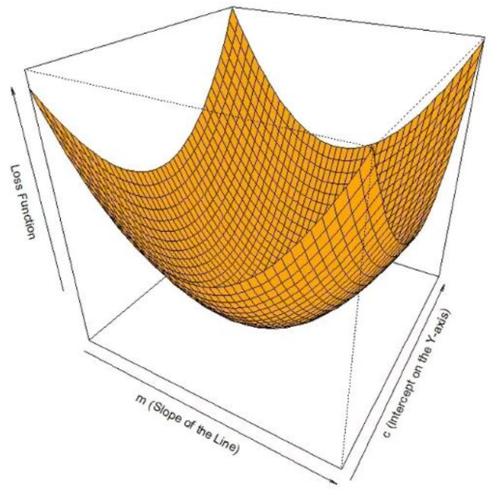
- Other considerations
  - Choice of hyperparameter δ determines outliers: residues larger than δ are minimized with L1; residues smaller than δ are minimized with L2
  - Huber loss curves around the minima which decreases the gradient; and it is more robust to outliers than MSE

11/41

Gradient Descent

- Popular optimization algorithm to find the local minimum for a differentiable function, e.g. loss function
- An iterative process of minimizing a function by following the gradient (slope) of the cost function

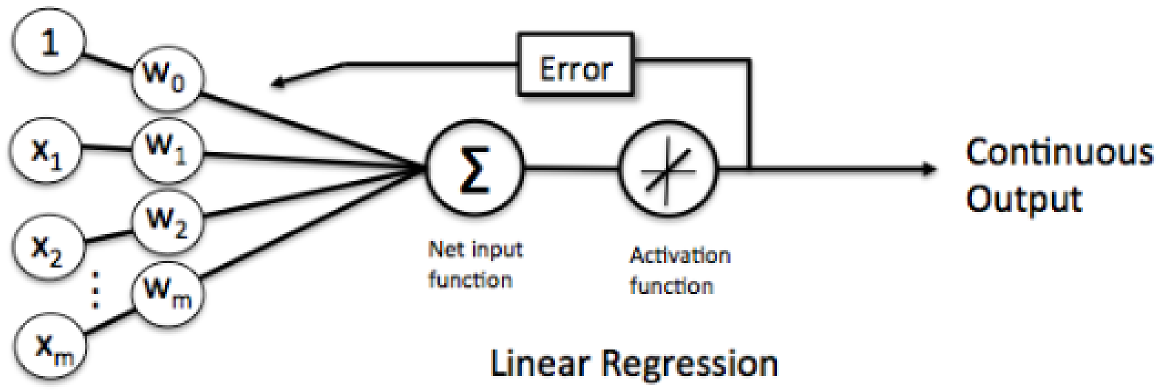
$$m_{n+1} = m_n - \alpha \frac{\partial}{\partial m_n} LF_{m_n}$$



12/41

Regression Model

- Linear regression



- How is a logistic regression different?
    - Activation function
    - Loss function

Logistic/Sigmoid Function

- Logistic function

$$f(x) = \frac{1}{1 + e^{-k(x-x_0)}}$$

- Sigmoid function: a special case of logistic function or standard logistic function where $K = 1$ and $x_0 = 0$

$$S(x) = \frac{1}{1 + e^{-x}}$$

- Logit function: inverse of sigmoid function

$$logit(p) = log(\frac{p}{1-p})$$

$$odd = \frac{p}{1-p}$$

Interpretation of Logistic Regression Coefficients

$$logit(p) = \sum_{i}^{n} \beta_i \cdot x_i + \beta_0$$

$$p = \frac{1}{1 + e^{-\sum_i^n \beta_i \cdot x_i + \beta_0}}$$

- How to interpret coefficient β_i?

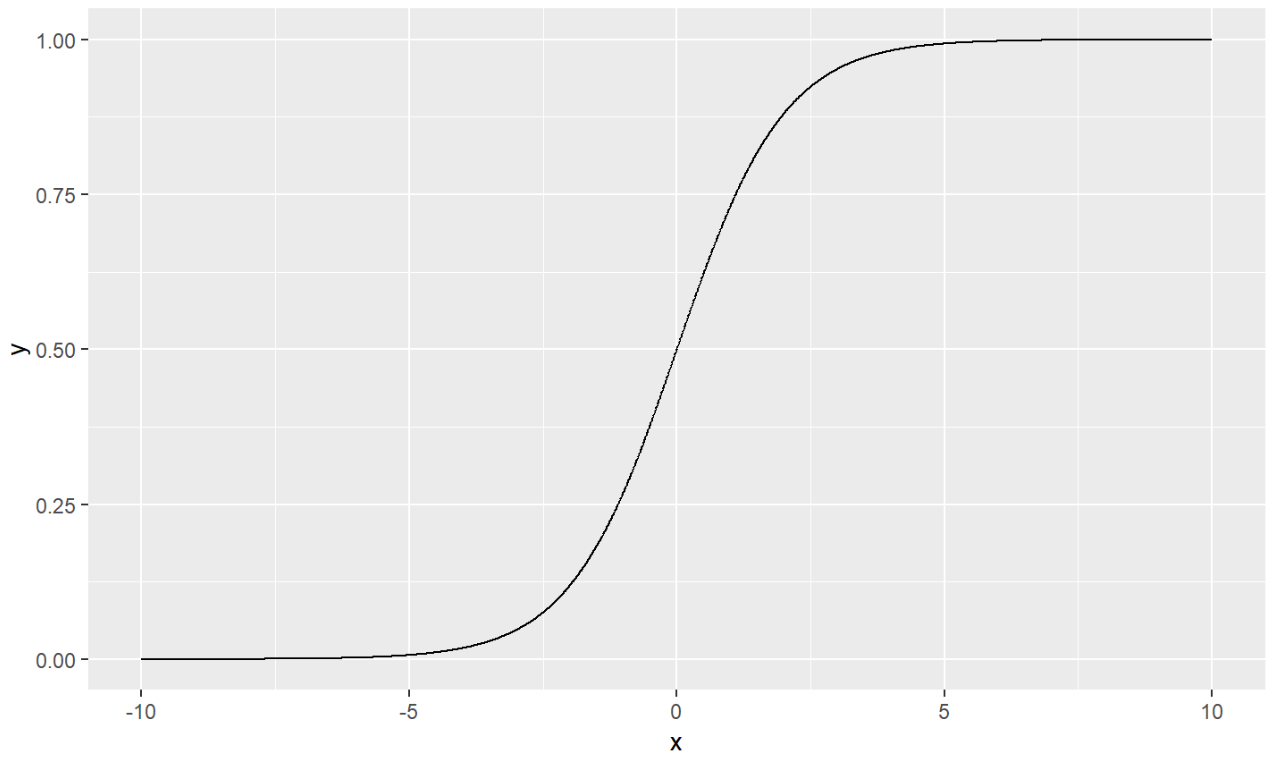
$$log(odd_1) = \beta_0 + \beta_1 \cdot x_1 + \ldots + \beta_i \cdot x_i + \ldots + \beta_n \cdot x_n$$

$$log(odd_2) = \beta_0 + \beta_1 \cdot x_1 + \ldots + \beta_i \cdot (x_i + 1) + \ldots + \beta_n \cdot x_n$$

$$\log(odd_2) - \log(odd_1) = \log \frac{odd_2}{odd_1} = \beta_i$$

15/41

Sigmoid Function: S-Shape and its Properties



Classification Loss Function

- Cross entropy loss function: negative log likelihood
  - $CrossEntropyLoss(y_i, \hat{y_i}) = -(y_i \log(\hat{y_i}) + (1 - y_i) \log(1 - \hat{y_i}))$
  - Increases as the predicted probability diverges from the actual label
  - Property: penalize heavily the predictions that are but
- Gradient of cross entropy loss (or log loss)
  - $\partial E / \partial p_i = - c_i / p_i + (1 - c_i)/(1 - p_i)$

Regression with Regularization: Lasso vs Ridge

- Intuition: add a penalty term to loss function that is based on magnitude of coefficients as a way to reduce model complexity and overfitting, i.e. regularization
  - $RLS = LS + \lambda \sum_{i=1}^{n} {\beta_i}^q$
- Lasso regularization: L1 loss function ($q = 1$)
  - Dimension reduction and feature selection
- Ridge regularization: L2 loss function ($q = 2$)
  - Feature ranking and relative importance; but can’t zero out coefficients like Lasso
- Shrinkage parameter: $\lambda \geq 0$
  - $\lambda = 0$: no regularization, just regular regression
  - As λ increases, the coefficients decrease

Estimation of Coefficients of Logistic Regression

- Statistical approach: Maximum Liklihood Estimation (MLE)
  - Estimate the distribution parameters (θ) that maximize the product of probabilities of observing all the data points
- Machine learning approach
  - Loss function: cross-entropy function
  - Optimization algorithm, e.g. stochastic gradient descent (SGD)
- Difference between two approaches

Demo Data: Boston Housing Price

```
library(caret)
library(mlbench)
data("BostonHousing")
str(BostonHousing)
```

```
## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : num  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ b      : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

Train Linear Model

```
model_lm <- train(medv ~ ., data = BostonHousing, method = "lm")
print(model_lm)
```

```
## Linear Regression
##
## 506 samples
##  13 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 506, 506, 506, 506, 506, 506, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   4.731782  0.7207107  3.378688
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

Model Specification

```
model_lm$finalModel
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Coefficients:
## (Intercept)         crim           zn        indus        chas1
##   3.646e+01   -1.080e-01    4.642e-02    2.056e-02    2.687e+00
##         nox           rm          age          dis          rad
##  -1.777e+01    3.810e+00    6.922e-04   -1.476e+00    3.060e-01
##         tax      ptratio            b        lstat
##  -1.233e-02   -9.527e-01    9.312e-03   -5.248e-01
```

Examine the Linear Regression Model

```
names(summary(model_lm))
```

```
##  [1] "call"          "terms"         "residuals"     "coefficients"
##  [5] "aliased"       "sigma"         "df"            "r.squared"
##  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
```

```
summary(model_lm)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.595  -2.730  -0.518   1.777  26.199
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.646e+01  5.103e+00   7.144 3.28e-12 ***
## crim        -1.080e-01  3.286e-02  -3.287 0.001087 **
## zn           4.642e-02  1.373e-02   3.382 0.000778 ***
## indus        2.056e-02  6.150e-02   0.334 0.738288
```

Examine the Linear Regression Model (2)

```
coef <- summary(model_lm)$coefficients
coef <- coef[coef[, 4] < 0.05, ]
coef[order(coef[, 4]), ]
```

```
##                 Estimate  Std. Error    t value     Pr(>|t|)
## lstat        -0.524758378 0.050715278 -10.347146 7.776912e-23
## rm            3.809865207 0.417925254   9.116140 1.979441e-18
## dis          -1.475566846 0.199454735  -7.398004 6.013491e-13
## ptratio      -0.952747232 0.130826756  -7.282511 1.308835e-12
## (Intercept)  36.459488385 5.103458811   7.144074 3.283438e-12
## nox         -17.766611228 3.819743707  -4.651257 4.245644e-06
## rad           0.306049479 0.066346440   4.612900 5.070529e-06
## b             0.009311683 0.002685965   3.466793 5.728592e-04
## zn            0.046420458 0.013727462   3.381576 7.781097e-04
## crim         -0.108011358 0.032864994  -3.286517 1.086810e-03
## tax          -0.012334594 0.003760536  -3.280009 1.111637e-03
## chas1         2.686733819 0.861579756   3.118381 1.925030e-03
```

Information of the Attributes

- MEDV - Median value of owner-occupied homes in $1000’s
- CRIM - per capita crime rate by town
- ZN - proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS - proportion of non-retail business acres per town.
- CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
- NOX - nitric oxides concentration (parts per 10 million)
- RM - average number of rooms per dwelling
- AGE - proportion of owner-occupied units built prior to 1940
- DIS - weighted distances to five Boston employment centres
- RAD - index of accessibility to radial highways
- TAX - full-value property-tax rate per $10,000
- PTRATIO - pupil-teacher ratio by town
- B - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- LSTAT - % lower status of the population

Model Details

```
summary(model_lm)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.595  -2.730  -0.518   1.777  26.199
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.646e+01  5.103e+00   7.144 3.28e-12 ***
## crim        -1.080e-01  3.286e-02  -3.287 0.001087 **
## zn           4.642e-02  1.373e-02   3.382 0.000778 ***
## indus        2.056e-02  6.150e-02   0.334 0.738288
## chas1        2.687e+00  8.616e-01   3.118 0.001925 **
## nox         -1.777e+01  3.820e+00  -4.651 4.25e-06 ***
## rm           3.810e+00  4.179e-01   9.116  < 2e-16 ***
## age          6.922e-04  1.321e-02   0.052 0.958229
## dis         -1.476e+00  1.995e-01  -7.398 6.01e-13 ***
## rad          3.060e-01  6.635e-02   4.613 5.07e-06 ***
## tax         -1.233e-02  3.760e-03  -3.280 0.001112 **
```

Model Residuals' Plot

```
plot(model_lm$finalModel)
```

Run LM Model with Standardized Data

```
model_lm2 <- train(medv ~ ., data = BostonHousing, method = "lm", preProcess = c("center", "sca
print(model_lm2)
```

```
## Linear Regression
##
## 506 samples
##  13 predictor
##
## Pre-processing: centered (13), scaled (13)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 506, 506, 506, 506, 506, 506, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   4.913235  0.7197477  3.450471
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

Calculate R Square and Compare with Output (1)

```
predict_lm <- predict(model_lm, newdata = BostonHousing, type = "raw")
real_predict <- data.frame(cbind(real = BostonHousing$medv, predict = predict_lm,
                                 square_diff = (BostonHousing$medv - predict_lm)^2))
str(real_predict)
```

```
## 'data.frame':    506 obs. of  3 variables:
##  $ real       : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
##  $ predict    : num  30 25 30.6 28.6 27.9 ...
##  $ square_diff: num  36 11.7 17.1 23 68.2 ...
```

Calculate R Square and Compare with Output (2)

```
SSE <- sum(real_predict$square_diff)
TSS <- var(BostonHousing$medv) * (length(BostonHousing$medv) + 1)
R_squared <- 1 - SSE/TSS
R_squared
```

```
## [1] 0.7416658
```

```
summary(model_lm)$r.squared
```

```
## [1] 0.7406427
```

Logistic Regression

```
library(arules)
BostonHousing2 <- BostonHousing
BostonHousing2$medv <- discretize(BostonHousing2$medv,
                                  method = "frequency",
                                  breaks = 2,
                                  labels = c("low", "high"))
table(BostonHousing2$medv)
```

```
##
##  low high
##  251  255
```

```
model_glm <- train(medv ~ ., data = BostonHousing2,
                   method = "glm", family = "binomial")
```

```
print(model_glm)
```

```
## Generalized Linear Model
##
## 506 samples
##  13 predictor
##   2 classes: 'low', 'high'
```

31/41

Logistic Regression Model Output

```
summary(model_glm)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0015  -0.3574   0.0085   0.2981   3.3286
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) 12.065836   4.007916   3.011  0.00261 **
## crim        -0.082159   0.074950  -1.096  0.27300
## zn           0.012275   0.013525   0.908  0.36411
## indus        0.029454   0.043259   0.681  0.49594
## chas1        1.659713   0.664634   2.497  0.01252 *
## nox         -7.283784   2.733645  -2.664  0.00771 **
## rm           1.617226   0.440282   3.673  0.00024 ***
## age         -0.028603   0.010451  -2.737  0.00620 **
## dis         -0.711035   0.168061  -4.231 2.33e-05 ***
## rad          0.237470   0.060920   3.898 9.70e-05 ***
## tax         -0.008461   0.002919  -2.899  0.00374 **
```

Relative Importance of Variables by Logistic Regression

```
varImp(model_glm)
```

```
## glm variable importance
##
##          Overall
## lstat   100.000
## ptratio  86.431
## dis      68.821
## rad      62.370
## rm       58.010
## tax      43.000
## age      39.859
## nox      38.455
## chas1    35.212
## b        17.748
## crim      8.051
## zn        4.394
## indus     0.000
```

Import Python Libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.pipeline import make_pipeline
```

Data Loading and Preprocessing

```
boston = load_boston()
boston_data = pd.DataFrame(boston.data, columns = boston.feature_names)
boston_data['Price'] = boston.target
X = boston_data.drop('Price', axis=1)
y = boston_data['Price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=16)
boston_data.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 506 entries, 0 to 505
## Data columns (total 14 columns):
## CRIM       506 non-null float64
## ZN         506 non-null float64
## INDUS      506 non-null float64
## CHAS       506 non-null float64
## NOX        506 non-null float64
## RM         506 non-null float64
## AGE        506 non-null float64
## DIS        506 non-null float64
## RAD        506 non-null float64
## TAX        506 non-null float64
## PTRATIO    506 non-null float64
## B          506 non-null float64
```

Linear Regression Training and Testing

```
lr_pipe = LinearRegression()
lr_pipe.fit(X_train, y_train)
```

```
## LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
y_pred = lr_pipe.predict(X_test)
print(f"RMSE: {round(np.sqrt(metrics.mean_squared_error(y_test, y_pred)), 3)}")
```

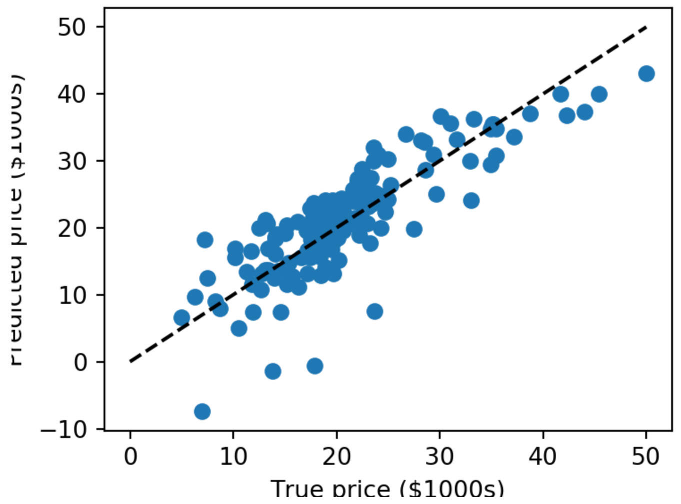
```
## RMSE: 4.614
```

Visualize Regression Line

```
plt.figure(figsize=(4, 3))
plt.scatter(y_test, y_pred)
plt.plot([0, 50], [0, 50], '--k')
plt.axis('tight')
```

```
## (-2.500483870967742, 52.51016129032258, -10.282281648953647, 52.87058484042636)
```

```
plt.tight_layout()
plt.xlabel('True price ($1000s)')
plt.ylabel('Predicted price ($1000s)')
plt.show()
```



Linear Model Coefficient Estimation

```
for idx, col_name in enumerate(X_train.columns):
  print(f"The coefficient for {col_name} is {lr_pipe.coef_[idx]}")
```

```
## The coefficient for CRIM is -0.135721389191917
## The coefficient for ZN is 0.03531958631459615
## The coefficient for INDUS is 0.0020350011834421146
## The coefficient for CHAS is 2.425658595197517
## The coefficient for NOX is -15.996609853279406
## The coefficient for RM is 3.997523842035774
## The coefficient for AGE is 0.022861021101290676
## The coefficient for DIS is -1.3542588201878618
## The coefficient for RAD is 0.346279075866271
## The coefficient for TAX is -0.011431344325432802
## The coefficient for PTRATIO is -1.0161040881448886
## The coefficient for B is 0.010906415198789543
## The coefficient for LSTAT is -0.6351622221570083
```

```
plt.scatter(lr_pipe.predict(X_train), lr_pipe.predict(X_train)-y_train, c='b', s=40, alpha=0.5
plt.hlines(y=0, xmin=0, xmax=50)
plt.xlabel("Fitted")
plt.ylabel("Residuals")
plt.title("Residals vs. fitted")
plt.show()
```

Evaluate Linear Regression Model

```
data_tuples = list(zip(y_test, y_pred))
real_predict = pd.DataFrame(data_tuples, columns=['Real', 'Predict'])
real_predict['squared_dif'] = (real_predict["Real"]-real_predict["Predict"])**2
real_predict.head(3)
```

```
##     Real    Predict  squared_dif
## 0   23.3  27.406211    16.860968
## 1   12.8  13.179976     0.144382
## 2    6.3   9.676904    11.403484
```

```
SSE = sum(real_predict['squared_dif'])
TSS = np.var(y_test) * (len(y_test)+1)
real_predict.head(3)
```

```
##     Real    Predict  squared_dif
## 0   23.3  27.406211    16.860968
## 1   12.8  13.179976     0.144382
## 2    6.3   9.676904    11.403484
```

```
Rsquared = 1 - SSE/TSS
Rsquared
```

Prepare for Logistic Regressin Dataset

```
boston_data2 = boston_data.copy()
boston_data2['Price'] = pd.qcut(boston_data2['Price'], 2, labels=["low", "high"])
X2 = boston_data2.drop('Price', axis=1)
y2 = boston_data2['Price']
X_train2, X_test2, y_train2, y_test2 = train_test_split(X2, y2, test_size=0.3, random_state=16
boston_data2.Price.value_counts()
```

```
## low     256
## high    250
## Name: Price, dtype: int64
```

Logistic Regression Model Training and Testing

```
logr_pipe = make_pipeline(StandardScaler(), LogisticRegression(solver='lbfgs'))
logr_pipe.fit(X_train2, y_train2)
```

```
## Pipeline(memory=None,
##          steps=[('standardscaler',
##                  StandardScaler(copy=True, with_mean=True, with_std=True)),
##                 ('logisticregression',
##                  LogisticRegression(C=1.0, class_weight=None, dual=False,
##                                     fit_intercept=True, intercept_scaling=1,
##                                     l1_ratio=None, max_iter=100,
##                                     multi_class='warn', n_jobs=None,
##                                     penalty='l2', random_state=None,
##                                     solver='lbfgs', tol=0.0001, verbose=0,
##                                     warm_start=False))],
##          verbose=False)
```

```
y_pred2 = logr_pipe.predict(X_test2)
y_pred2
```

```
## array(['high', 'low', 'low', 'high', 'low', 'high', 'high', 'low', 'low',
##        'high', 'low', 'low', 'low', 'high', 'low', 'high', 'low', 'high',
##        'low', 'low', 'low', 'low', 'low', 'low', 'low', 'low', 'low',
```