Introduction to Machine Learning Algorithm

1. Outline

- Introduction
- Machine Learning Problem Types
    - Supervised vs. Unsupervised
    - Regression vs. Classification
    - Parametric Model vs. Non-Parametric Model
- Common Evaluation Methods
- Linear Regression
    - Gradient Descent Methods
    - Underfitting vs. Overfitting: First Look
- Logistic Regression

2. Introduction

2.1 Machine Learning Example in Our Life

- Data on attributes of houses in San Francisco and New York.
- Attributes include: Elevation, year built, number of bathrooms and bedrooms, price, square feet, price per sqft.
- Given this dataset, what can we learn from it?
- If a new data case comes in with these attributes, can we predict which city it is in?

2.2 Machine Learning Introduction

- Machine Learning – use the help with computers to perform tasks such as prediction recognition, diagnosis, planning, robot control, etc.
- Machine Learning Essentials:
    - Input Vectors (feature vectors, sample, example, instance, datacase etc.)
    - Outputs
    - Training regime
        - Batch Methods: gain all data points to training our model
        - Online Methods: take samples from data to training model
    - Performance Evaluation

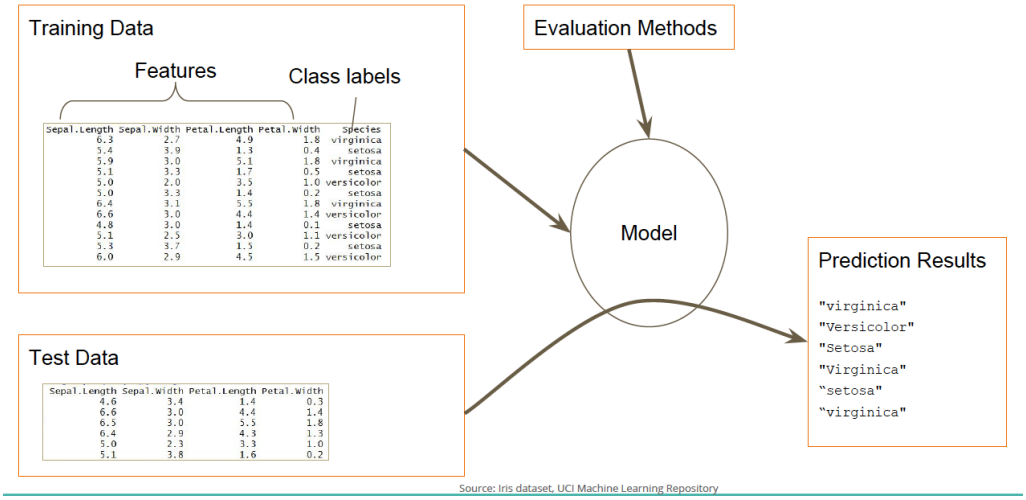
- Example of machine learning problem: decide whether to play tennis at a given day:

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Target Variable
- or -
Class Label
- or -
Goal
- or -
Output Variable

In this example, we called the left 5 features (Day, Outlook, Temperature, Humidity, Wind) as input attributes/input variables/features/attributes, and we called right feature (Play Tennis) as target variable/class label/goal/output variable.

- Example of machine learning problem: estimate the flower type depend on the data



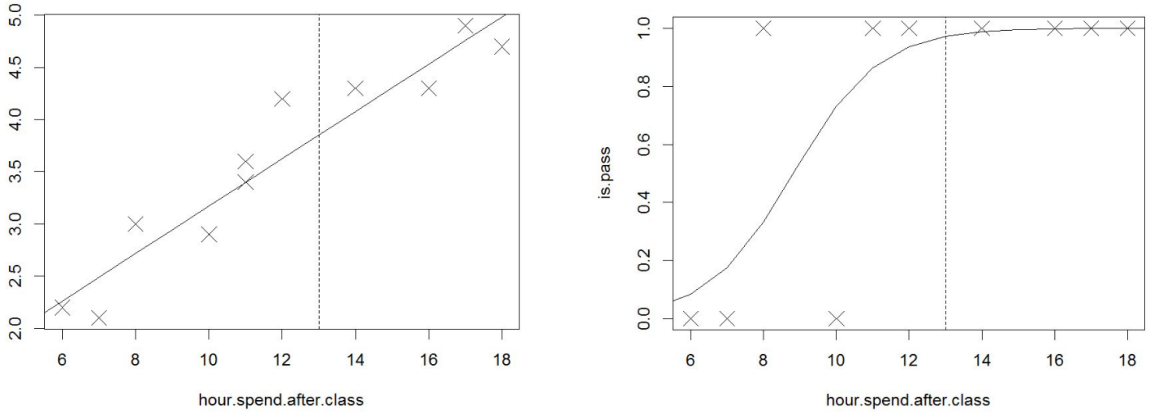
Workflow: Build a model depend on our training data, and create an evaluation method. Then put the test data into our model to get the prediction result.

3Machine Learning Problem Types

3.1 Supervised vs Unsupervised Learning

- Supervised Learning:
  - Output variables (class labels) are given.
  - The relationship between input and output is known.
- Reinforced Learning:
  - Output variables are not known, but actions are rewarded or punished.
- Unsupervised Learning:
  - Learn patterns from data without output variable or feedback.
- Semi-supervised Learning:
  - Only a small amount of data is labeled.

3.2 Regression vs Classification



- Regression: the output variable takes continuous values.
- Classification: the output variable takes class labels.

3.3 Parametric Model vs. Non-Parametric Model

Parametric Model:

- Models can be represented solely by parameters.
- Fixed & finite number of parameters that doesn' t depend on the data size
- Meaningful & interpretable parameters
- Ex: Linear Regression, SVM (primal problem)

Non-Parametric Model:

- Models are represented parameters and data
- Number of the parameters depends on the data size
- Infinite amount of parameters
- Parameters are not meaningful
- Ex: KNN, SVM (dual problem)

4 Evaluation Methods

- Regression: Mean squared error (MSE), sum squared error (SSE), etc.
- Classification: Accuracy, False Positive Rate, etc.
- Variations:
    - With cost matrix
    - With penalty terms to restrict model complexity
        - AIC/BIC
        - L1 regularization (LASSO)
        - L2 regularization (Ridge)

4.1 Regression

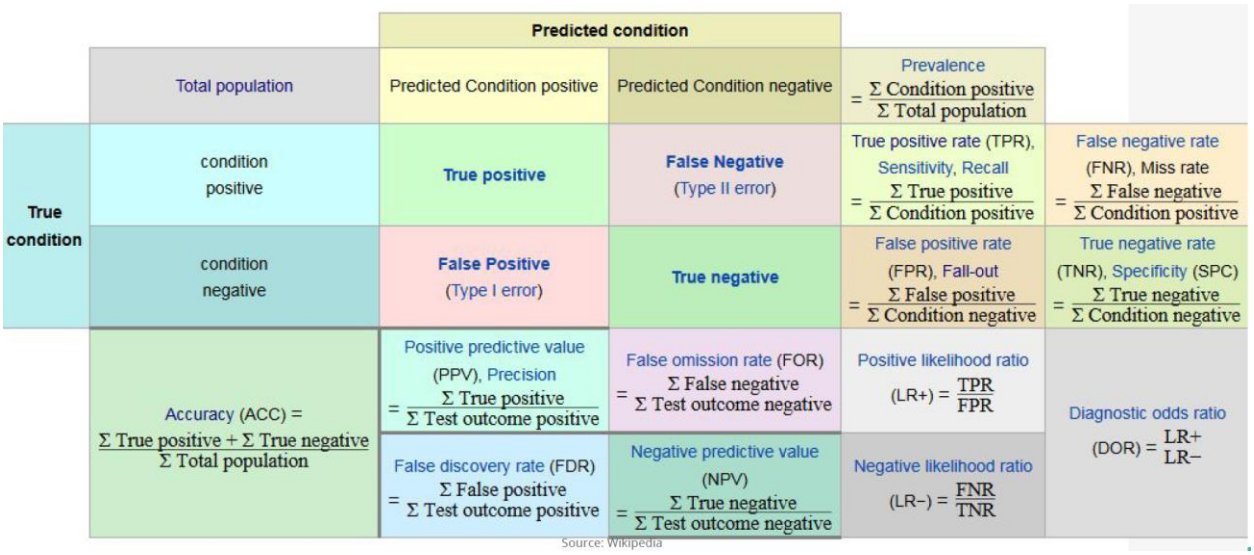
- Mean Squared Error (MSE)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

- Sum Squared Error/Least Squares (SSE)
- Root Mean Squared Error
- Mean Absolute Error
- Coefficient of Determination: is a statistical measure of how well the regression line approximates the real data points.

$$R^2 = \frac{\sum(\hat{y}_i - \bar{y})^2}{\sum(y_i - \bar{y})^2}$$

4.2 Classification

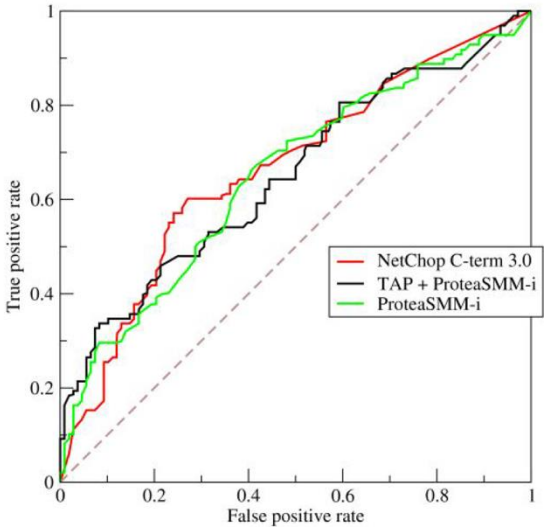


		Predicted condition			
	Total population	Predicted Condition positive	Predicted Condition negative	Prevalence = Σ Condition positive / Σ Total population	
True condition	condition positive	**True positive**	**False Negative** (Type II error)	True positive rate (TPR), Sensitivity, Recall = Σ True positive / Σ Condition positive	False negative rate (FNR), Miss rate = Σ False negative / Σ Condition positive
	condition negative	**False Positive** (Type I error)	**True negative**	False positive rate (FPR), Fall-out = Σ False positive / Σ Condition negative	True negative rate (TNR), Specificity (SPC) = Σ True negative / Σ Condition negative
	Accuracy (ACC) = (Σ True positive + Σ True negative) / Σ Total population	Positive predictive value (PPV), Precision = Σ True positive / Σ Test outcome positive	False omission rate (FOR) = Σ False negative / Σ Test outcome negative	Positive likelihood ratio (LR+) = TPR / FPR	Diagnostic odds ratio (DOR) = LR+ / LR−
		False discovery rate (FDR) = Σ False positive / Σ Test outcome positive	Negative predictive value (NPV) = Σ True negative / Σ Test outcome negative	Negative likelihood ratio (LR−) = FNR / TNR	

Source: Wikipedia

- Confusion Matrix: is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known.
    - true positives (TP): These are cases in which we predicted yes (they have the disease), and they do have the disease.
    - true negatives (TN): We predicted no, and they don't have the disease.
    - false positives (FP): We predicted yes, but they don't actually have the disease. (Also known as a "Type I error.")
    - false negatives (FN): We predicted no, but they actually do have the disease. (Also known as a "Type II error.")
    - Accuracy: Overall, how often is the classifier correct? (TP+TN)/total
- AUC（Area Under Curve）
    - ROC（Receiver operating characteristic）
    - illustrates the performance of a binary classifier as its discrimination threshold is varied.
    - Plot the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

- Properties:
  - Examine how our prediction rank against
  - the true class labels.
  - Freedom to choose or change thresholds for different applications.
  - Independent of the fraction of the test population which is class 0 or class 1.
- Example Applications:
  - Credit Evaluation
  - Advertising Strategies



4.3 Variation

- Cost Matrix: Used for comparing two different models:

	actual negative	actual positive
predict negative	$C(0,0) = c_{00}$	$C(0,1) = c_{01}$
predict positive	$C(1,0) = c_{10}$	$C(1,1) = c_{11}$

- Regularization
  - Regularization terms can be added to (or subtracted from) the model' s loss function (or objective function).
  - Restrict the parameter space in order to prevent a model gets too complicated.
  - Prevent overfitting
    - Overfitting: occurs when a model is excessively complex, such as having too many parameters relative to the number of observations.
    - Underfitting: occurs when a statistical model or machine learning algorithm cannot capture the underlying trend of the data
  - Regularization types:
    - L0 regularization:
      $\mathrm{sum}(|w|^0)$, controls the number of parameters
    - L1 regularization (LASSO: least absolute shrinkage and selection operator): $\mathrm{sum}(|w|^1)$, ability to restrain parameters to be 0.
    - L2 regularization (Ridge):
      $\mathrm{sum}(|w|^2)$

5 Linear Regression

5.1 Notations

- x： training data
- y： target variable or class label
- x^i : i-th data case
- x_j : j-yh feature of x
- m：total number of data case
- n：total number of feature

5.2 Model

Model Function： $h_\theta(x) = \theta_0 + \theta_1 x_1$

Let x_0 be a vector of 1s, $x_0 = [1, 1, 1, \ldots, 1]^T$: $h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x$

The difference between target and predictions： $\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$

Define the cost function： $J(\theta) = \frac{1}{2} \sum_{i=1}^{m} (x^{(i)}) - y^{(i)})^2$

Our goal is to find the parameter that minimizing the cost function J(θ):

- $\hat{\theta} = \arg\min_\theta J(\theta)$
- One method: Gradient descent

5.3 Gradient Descent

- An iterative method to find the max/min* values.
- *: Local max/min; sensitive to starting point
- Intuition: At each step, walk towards the steepest direction.
- Used widely in practice
- Steps:

1. Pick a starting point
2. Repeat
   a. Calculate the gradient of the function at current point
   b. Move a step towards the direction of the gradient to reduce J(θ)
3. Stop when J(θ) is small enough

- **Example of Gradient Descent in Linear Regression**

At each iteration, update the parameter $\theta_i \leftarrow \theta_i - \alpha \frac{\partial J(\theta)}{\partial \theta_i}$, where α is the step size.

$$\frac{\partial J(\theta)}{\partial \theta_i} = \frac{\partial}{\partial \theta_j} \frac{1}{2} \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

$$= \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \frac{\partial}{\partial \theta_j} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

$$= \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \frac{\partial}{\partial \theta_j} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)} \right)$$

$Thus,$

$$\frac{\partial J(\theta)}{\partial \theta_0} = \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

$$\frac{\partial J(\theta)}{\partial \theta_1} = \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}$$

5.4 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions such as (linear) Support Vector Machines and Logistic Regression. Even though SGD has been around in the machine learning community for a long time, it has received a considerable amount of attention just recently in the context of large-scale learning. Stochastic Gradient Descent can be used:

Repeat:

- For i in 1 to m:
  - Perform gradient calculation using only data point i

    $\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$

5.5 Variation

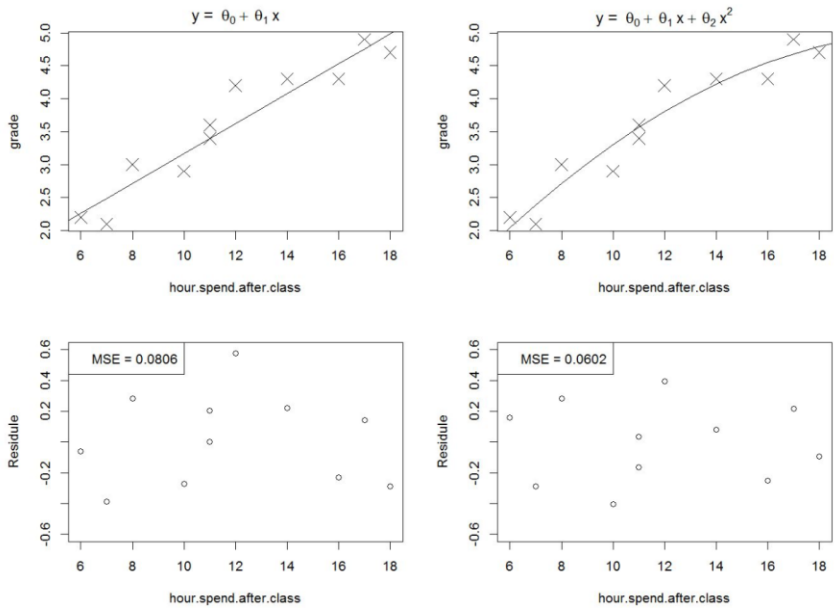
We can apply feature transformation to make the model appear "non-linear".

EX: Polynomial Regression

- Useful when there is reason to believe the relationship between two variables is curvilinear.

- $y = \theta_0 + \theta_1 x + \theta_2 x^2$

- Might lead to overfitting.



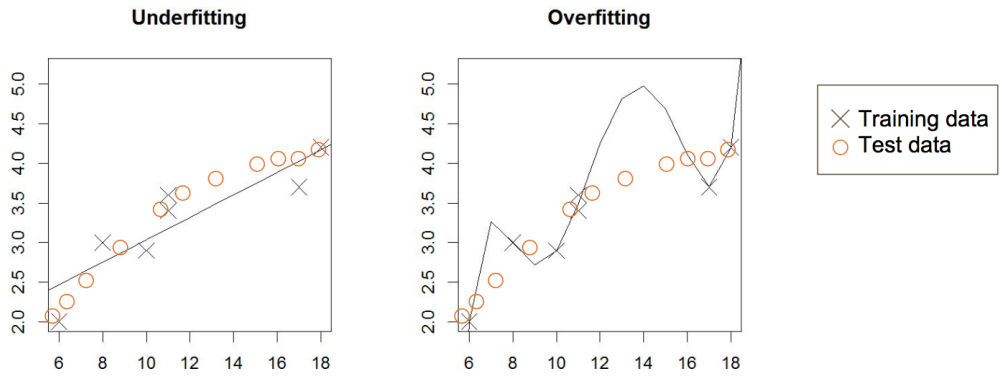
5.6 Bias vs. Variance (Underfitting vs. Overfitting)

Underfitting:

- Refers to a model that can neither model the training data nor generalize to new data.
- An underfit machine learning model is not a suitable model and will be obvious as it will have poor performance on the training data.

Overfitting:

- Overfitting refers to a model that models the training data too well.
- Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model.



6 Logistic Regression

- Logistic function: $\sigma(t) = \frac{e^t}{e^t+1} = \frac{1}{1+e^{-t}}$
- Logistic regression: $F(x) = \frac{1}{1+e^{-(\beta_0+\beta_1 x)}}$, where F(x) is the probability of the target variable if X is "positive" : F(X) = P(Y = 1|X)
- It assumes the probability of the target variable being "positive" can be explained by a combination of the explanatory variable after logistic-transformation.
- Why is logistic regression related to linear regression?

$$F(x) = \frac{1}{1+e^{-(\beta_0+\beta_1 x)}} \Longrightarrow \frac{F(x)}{1-F(x)} = e^{\beta_0+\beta_1 x} \Longrightarrow \ln\left(\frac{F(x)}{1-F(x)}\right) = \beta_0 + \beta_1 x$$

  In $(\frac{F(x)}{1-F(x)})$ is called "odds" . $(\frac{F(x)}{1-F(x)})$ is called "logit" .
- In logistic regression, we assume a linear relationship between explanatory variable and the log odds of the target variable.

7. Other basic Models

- We not be talking about these models but they are worth looking at
  - Naive Bayes
  - K Nearest Neighbor
  - K-Means for clustering
  - Hierarchical clustering

8 Examples in Python

Import the Libraries

In [1]:
```
# imports
import pandas as pd
import matplotlib.pyplot as plt

# this allows plots to appear directly in the notebook
%matplotlib inline
```

Read data into a DataFrame

In [3]:
```
# read data into a DataFrame
data = pd.read_csv('http://www-bcf.usc.edu/~gareth/ISL/Advertising.csv', index_col=0)
data.head()
```

Out[3]:

	TV	Radio	Newspaper	Sales
1	230.1	37.8	69.2	22.1
2	44.5	39.3	45.1	10.4
3	17.2	45.9	69.3	9.3
4	151.5	41.3	58.5	18.5
5	180.8	10.8	58.4	12.9

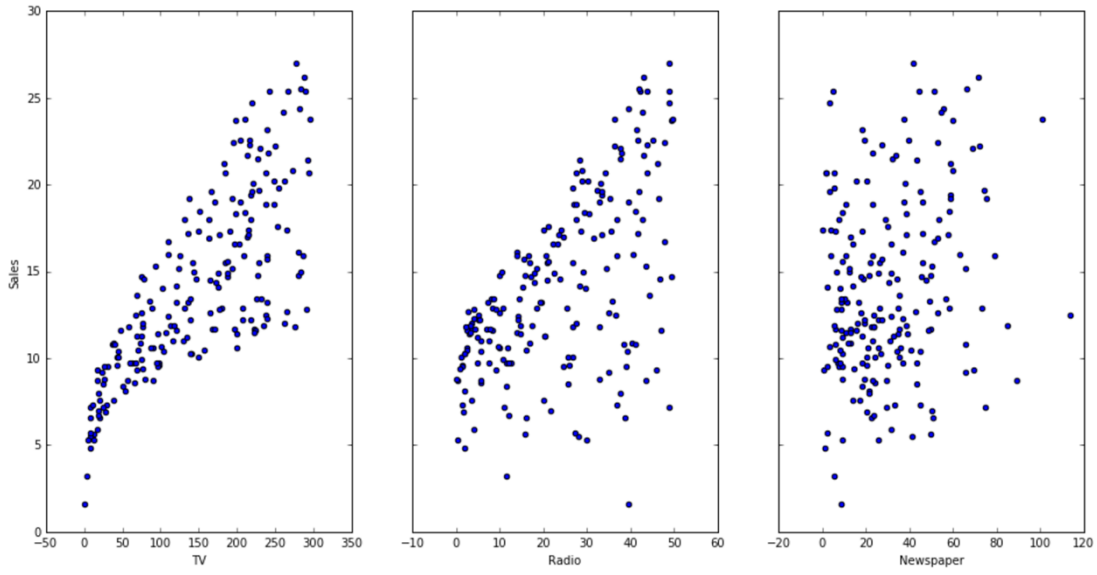
In [4]:
```
# print the shape of the DataFrame
data.shape
```

Out[4]:
```
(200, 4)
```

Plotting the relationship

In [5]:
```
# visualize the relationship between the features and the response using scatterplots
fig, axs = plt.subplots(1, 3, sharey=True)
data.plot(kind='scatter', x='TV', y='Sales', ax=axs[0], figsize=(16, 8))
data.plot(kind='scatter', x='Radio', y='Sales', ax=axs[1])
data.plot(kind='scatter', x='Newspaper', y='Sales', ax=axs[2])
```

Out[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xa0fcf98>
```



Simple Linear Regression

In [6]:
```
import statsmodels.formula.api as smf

# create a fitted model in one line
lm = smf.ols('Sales ~ TV', data=data).fit()

# print the coefficients
lm.params
```

Out[6]:
```
Intercept    7.032594
TV           0.047537
dtype: float64
```

Using the Model for Prediction

In [6]:
```
# manually calculate the prediction
7.032594 + 0.047537*50
```

Out[6]:
```
9.409444
```

In [7]:
```
# you have to create a DataFrame since the Statsmodels formula interface expects it
X_new = pd.DataFrame({'TV': [50]})
X_new.head()
```

Out[7]:

	TV
0	50

In [8]:
```
# use the model to make predictions on a new value
lm.predict(X_new)
```

Out[8]:
```
0    9.409426
dtype: float64
```

Plotting the Least Squares Line

In [9]:
```
# create a DataFrame with the minimum and maximum values of TV
X_new = pd.DataFrame({'TV': [data.TV.min(), data.TV.max()]})
X_new.head()
```

Out[9]:

	TV
0	0.7
1	296.4

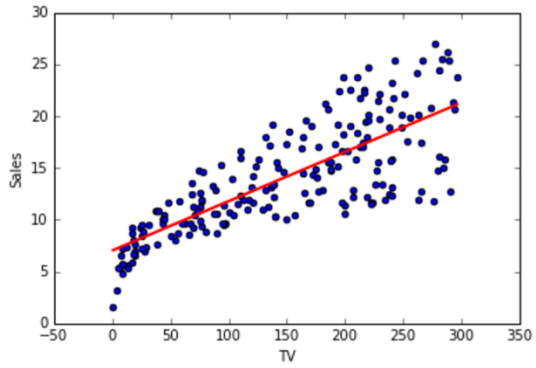
In [10]:
```
# make predictions for those x values and store them
preds = lm.predict(X_new)
preds
```

Out[10]:
```
0     7.065869
1    21.122454
dtype: float64
```

In [11]:
```
# first, plot the observed data
data.plot(kind='scatter', x='TV', y='Sales')

# then, plot the least squares line
plt.plot(X_new, preds, c='red', linewidth=2)
```

Out[11]: [<matplotlib.lines.Line2D at 0xa88ebe0>]



Confidence Intervals

In [12]:
```
# print the confidence intervals for the model coefficients
lm.conf_int()
```

Out[12]:

	0	1
Intercept	6.129719	7.935468
TV	0.042231	0.052843

Hypothesis Testing and p-values

In [13]:
```
# print the p-values for the model coefficients
lm.pvalues
```

Out[13]:
```
Intercept    1.406300e-35
TV           1.467390e-42
dtype: float64
```

How Well Does the Model Fit the data?

In [14]:
```
# print the R-squared value for the model
lm.rsquared
```

Out[14]: 0.61187505085007099

Multiple Linear Regression

In [15]:
```
# create a fitted model with all three features
lm = smf.ols(formula='Sales ~ TV + Radio + Newspaper', data=data).fit()

# print the coefficients
lm.params
```

Out[15]:
```
Intercept    2.938889
TV           0.045765
Radio        0.188530
Newspaper   -0.001037
dtype: float64
```

In [16]:
```
# print a summary of the fitted model
lm.summary()
```

Out[16]:

OLS Regression Results

Dep. Variable:	Sales	**R-squared:**	0.897
Model:	OLS	**Adj. R-squared:**	0.896
Method:	Least Squares	**F-statistic:**	570.3
Date:	Wed, 07 Sep 2016	**Prob (F-statistic):**	1.58e-96
Time:	15:06:55	**Log-Likelihood:**	-386.18
No. Observations:	200	**AIC:**	780.4
Df Residuals:	196	**BIC:**	793.6
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	2.9389	0.312	9.422	0.000	2.324	3.554
TV	0.0458	0.001	32.809	0.000	0.043	0.049
Radio	0.1885	0.009	21.893	0.000	0.172	0.206
Newspaper	-0.0010	0.006	-0.177	0.860	-0.013	0.011

Omnibus:	60.414	**Durbin-Watson:**	2.084
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	151.241
Skew:	-1.327	**Prob(JB):**	1.44e-33
Kurtosis:	6.332	**Cond. No.**	454.

Linear Regression in scikit-learn

In [18]:
```
# create X and y
feature_cols = ['TV', 'Radio', 'Newspaper']
X = data[feature_cols]
y = data.Sales

# follow the usual sklearn pattern: import, instantiate, fit
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(X, y)

# print intercept and coefficients
print lm.intercept_
print lm.coef_
```

```
2.93888936946
[ 0.04576465  0.18853002 -0.00103749]
```

In [19]:
```
# pair the feature names with the coefficients
zip(feature_cols, lm.coef_)
```

Out[19]:
```
[('TV', 0.045764645455397615),
 ('Radio', 0.18853001691820456),
 ('Newspaper', -0.0010374930424763272)]
```

In [20]:
```
# predict for a new observation
lm.predict([100, 25, 25])
```

Out[20]:
```
array([ 12.20266701])
```

In [21]:
```
# calculate the R-squared
lm.score(X, y)
```

Out[21]:
```
0.89721063817895208
```

Handling Categorical Predictors with Two Categories

In [22]:
```
import numpy as np

# set a seed for reproducibility
np.random.seed(12345)

# create a Series of booleans in which roughly half are True
nums = np.random.rand(len(data))
mask_large = nums > 0.5

# initially set Size to small, then change roughly half to be large
data['Size'] = 'small'
data.loc[mask_large, 'Size'] = 'large'
data.head()
```

Out[22]:

	TV	Radio	Newspaper	Sales	Size
1	230.1	37.8	69.2	22.1	large
2	44.5	39.3	45.1	10.4	small
3	17.2	45.9	69.3	9.3	small
4	151.5	41.3	58.5	18.5	small
5	180.8	10.8	58.4	12.9	large

In [23]:
```
# create a new Series called IsLarge
data['IsLarge'] = data.Size.map({'small':0, 'large':1})
data.head()
```

Out[23]:

	TV	Radio	Newspaper	Sales	Size	IsLarge
1	230.1	37.8	69.2	22.1	large	1
2	44.5	39.3	45.1	10.4	small	0
3	17.2	45.9	69.3	9.3	small	0
4	151.5	41.3	58.5	18.5	small	0
5	180.8	10.8	58.4	12.9	large	1

In [24]:
```
# create X and y
feature_cols = ['TV', 'Radio', 'Newspaper', 'IsLarge']
X = data[feature_cols]
y = data.Sales

# instantiate, fit
lm = LinearRegression()
lm.fit(X, y)

# print coefficients
zip(feature_cols, lm.coef_)
```

Out[24]:
```
[('TV', 0.045719820924362775),
 ('Radio', 0.18872814313427869),
 ('Newspaper', -0.0010976794483516655),
 ('IsLarge', 0.057423850854827763)]
```

Predictors with More than Two Categories

In [25]:
```
# set a seed for reproducibility
np.random.seed(123456)

# assign roughly one third of observations to each group
nums = np.random.rand(len(data))
mask_suburban = (nums > 0.33) & (nums < 0.66)
mask_urban = nums > 0.66
data['Area'] = 'rural'
data.loc[mask_suburban, 'Area'] = 'suburban'
data.loc[mask_urban, 'Area'] = 'urban'
data.head()
```

Out[25]:

	TV	Radio	Newspaper	Sales	Size	IsLarge	Area
1	230.1	37.8	69.2	22.1	large	1	rural
2	44.5	39.3	45.1	10.4	small	0	urban
3	17.2	45.9	69.3	9.3	small	0	rural
4	151.5	41.3	58.5	18.5	small	0	urban
5	180.8	10.8	58.4	12.9	large	1	suburban

In [26]:
```
# create three dummy variables using get_dummies, then exclude the first dummy column
area_dummies = pd.get_dummies(data.Area, prefix='Area').iloc[:, 1:]

# concatenate the dummy variable columns onto the original DataFrame (axis=0 means rows, axis=1
data = pd.concat([data, area_dummies], axis=1)
data.head()
```

Out[26]:

	TV	Radio	Newspaper	Sales	Size	IsLarge	Area	Area_suburban	Area_urban
1	230.1	37.8	69.2	22.1	large	1	rural	0.0	0.0
2	44.5	39.3	45.1	10.4	small	0	urban	0.0	1.0
3	17.2	45.9	69.3	9.3	small	0	rural	0.0	0.0
4	151.5	41.3	58.5	18.5	small	0	urban	0.0	1.0
5	180.8	10.8	58.4	12.9	large	1	suburban	1.0	0.0

Linear Regression using Gradient Descent

In [2]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import random

def generateSample(N, variance=100):
    X = np.matrix(range(N)).T + 1
    Y = np.matrix([random.random() * variance + i * 10 + 900 for i in range(len(X))]).T
    return X, Y

def fitModel_gradient(x, y):
    N = len(x)
    w = np.zeros((x.shape[1], 1))
    eta = 0.0001

    maxIteration = 100000
    for i in range(maxIteration):
        error = x * w - y
        gradient = x.T * error / N
        w = w - eta * gradient
    return w

def plotModel(x, y, w):
    plt.plot(x[:,1], y, "x")
    plt.plot(x[:,1], x * w, "r-")
    plt.show()
```

```
def test(N, variance, modelFunction):
    X, Y = generateSample(N, variance)
    X = np.hstack([np.matrix(np.ones(len(X))).T, X])
    w = modelFunction(X, Y)
    plotModel(X, Y, w)


test(50, 600, fitModel_gradient)
test(50, 1000, fitModel_gradient)
test(100, 200, fitModel_gradient)
```