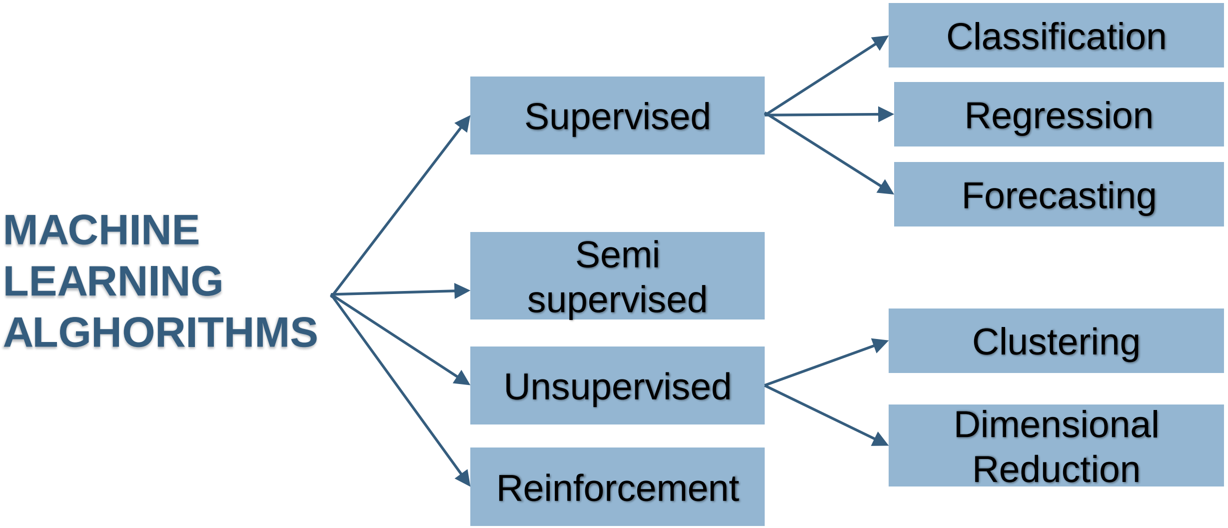
Machine Learning Alghorithms

Professor: Dr. Hamidreza Bolhasani
Lecturer: Rozhan Moosavi

TYPES OF MACHINE LEARNING ALGHORITHMS



Regression

Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting.

Regression types:

- Linear Regression
- Polynomial Regression
- Logistic Regression

Linear Regression(math mode)

$Y = 3X + 4$

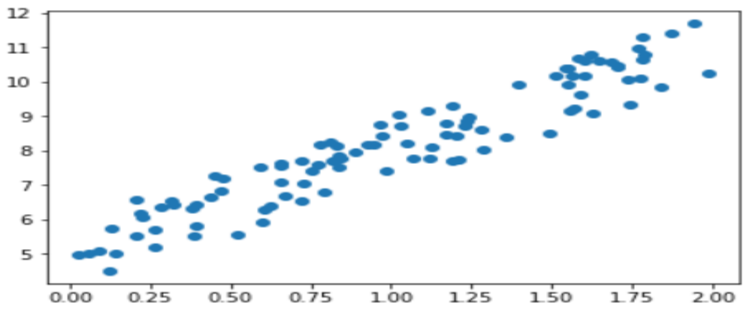
$Y = W_1 * X + W_0$

$Y = [W_1 , W_0][^{X}{}_{1}]$

$Y = W_TX \text{ OR } WX_T$

$W = (X_T\ X)^{-1}(X_T\ Y)$

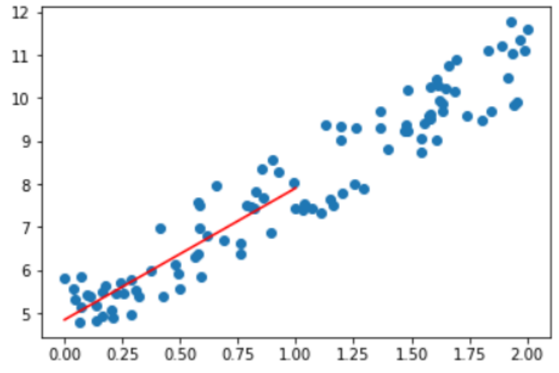
```
x = 2 * np.random.rand(100 , 1)
y = 4+ 3 * x + (2* np.random.rand(100,1))
plt.plot(x, y, 'o')
plt.show()
```



Linear Regression(math mode)

```
[4] x_sample = np.array([[0] , [1]])
    x_sample_b = np.c_[np.ones((2,1)) , x_sample]
```

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x,y)
y_pred = model.predict(x_sample)
plt.plot(x , y, 'o')
plt.plot(x_sample , y_pred, '-' , color = 'red')
plt.show()
```



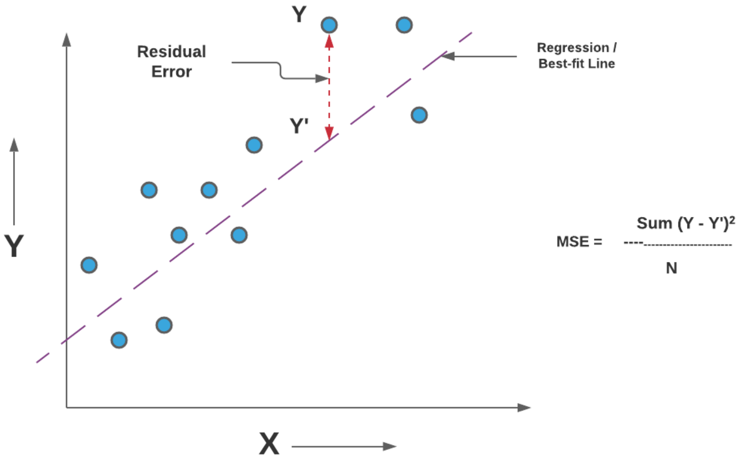
Mean Squared Error (MSE)

Loss function:

$$\text{MSE}(\mathbf{X}, h_{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^{T}\mathbf{x}^{(i)} - y^{(i)}\right)^{2}$$

Gradien of Loss function:

$$\frac{\partial}{\partial\theta_j}\text{MSE}(\theta) = \frac{2}{m}\sum_{i=1}^{m}\left(\theta^{T}\mathbf{x}^{(i)} - y^{(i)}\right)x_j^{(i)}$$



Linear Regression(Gradient Descent)

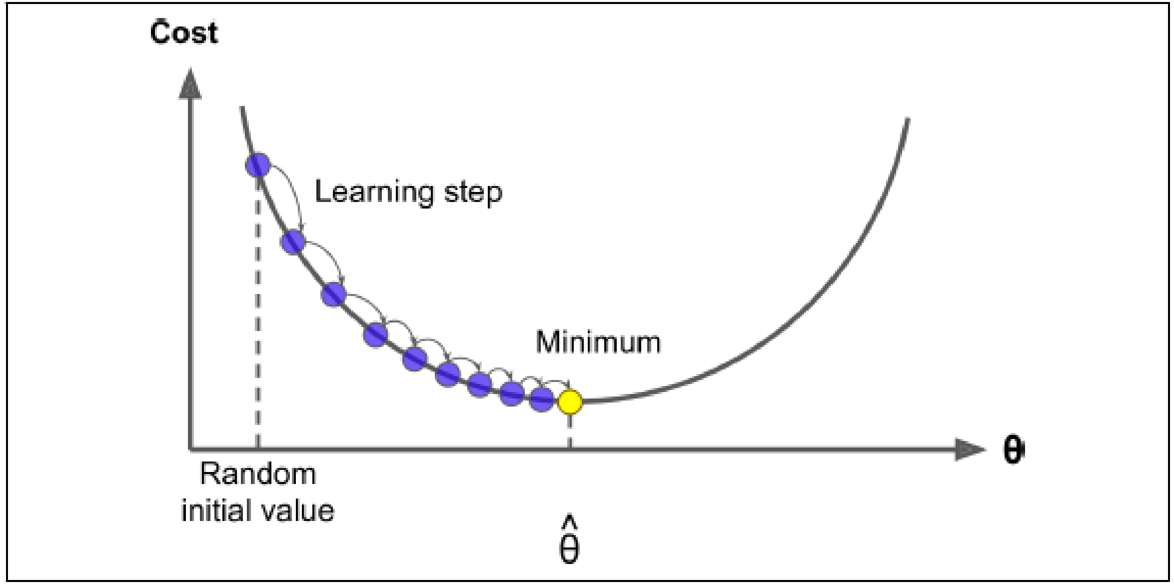
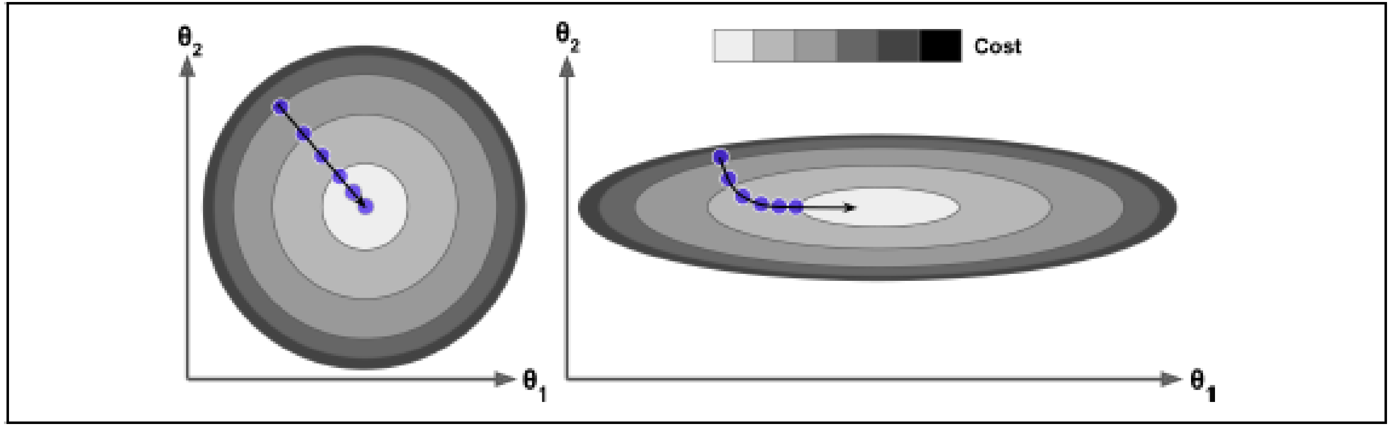


Figure 4-3. Gradient Descent

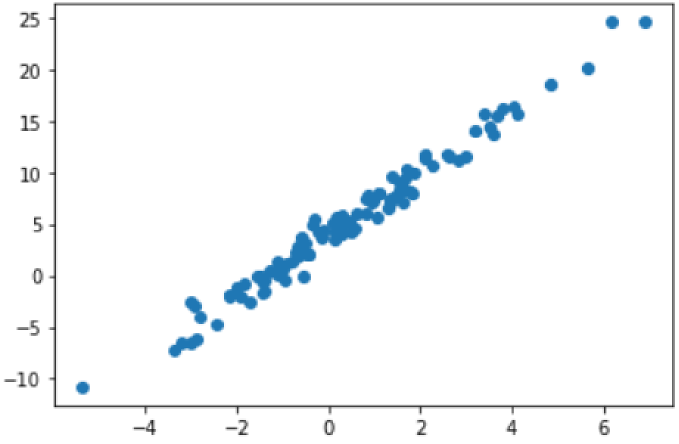
Linear Regression(Gradient Descent)



Gradient Descent Example

[17]

```
x = 2* np.random.randn(100,1)
y = np.random.randn(100,1) + 4 + 3*x
plt.plot(x , y , 'o')
plt.show()
x_b = np.c_[np.ones((100,1)), x]
```



Gradient Descent Example

```
[18] lr = 0.1 #learning_rate
     epochs = 100
     w = np.random.randn(2, 1)
     m = 100 # size of data

     Ws = []
     for epoch in range(epochs):
       gradiants = (2 / m ) * x_b.T.dot(x_b.dot(w) - y)
       w = w - lr * gradiants

       loss = (1 / m) * np.sum( (x_b.dot(w) - y)**2 )
       print("on epochs {} loss is {}".format(epoch , loss))

       Ws.append(w)
```

```
on epochs 0 loss is 16.504865678562798
on epochs 1 loss is 11.108342144423366
on epochs 2 loss is 7.5912715055225535
on epochs 3 loss is 5.29510364015185
on epochs 4 loss is 3.7960180581006724
on epochs 5 loss is 2.817318888658924
on epochs 6 loss is 2.1783613296412385
on epochs 7 loss is 1.7612088736330094
on epochs 8 loss is 1.4888650080698176
on epochs 9 loss is 1.3110614735547566
on epochs 10 loss is 1.1949799278196525
on epochs 11 loss is 1.1191944531551454
on epochs 12 loss is 1.0697168369349837
```

```
on epochs 71 loss is 0.9766348133861 44
on epochs 72 loss is 0.976663481338231
on epochs 73 loss is 0.9766348133798 09
on epochs 74 loss is 0.97666348133781 7
on epochs 75 loss is 0.9766634813377106
on epochs 76 loss is 0.9766634813376408
on epochs 77 loss is 0.9766634813375953
on epochs 78 loss is 0.9766634813375656
on epochs 79 loss is 0.9766634813375462
on epochs 80 loss is 0.9766634813375337
on epochs 81 loss is 0.9766634813375252
on epochs 82 loss is 0.97666348133752
on epochs 83 loss is 0.9766634813375162
on epochs 84 loss is 0.9766634813375137
on epochs 85 loss is 0.9766634813375126
on epochs 86 loss is 0.9766634813375114
on epochs 87 loss is 0.9766634813375107
on epochs 88 loss is 0.9766634813375105
on epochs 89 loss is 0.9766634813375104
on epochs 90 loss is 0.97666348133751
on epochs 91 loss is 0.9766634813375098
on epochs 92 loss is 0.9766634813375096
on epochs 93 loss is 0.9766634813375098
on epochs 94 loss is 0.9766634813375098
on epochs 95 loss is 0.9766634813375096
on epochs 96 loss is 0.9766634813375098
on epochs 97 loss is 0.9766634813375096
on epochs 98 loss is 0.9766634813375095
on epochs 99 loss is 0.9766634813375095
```

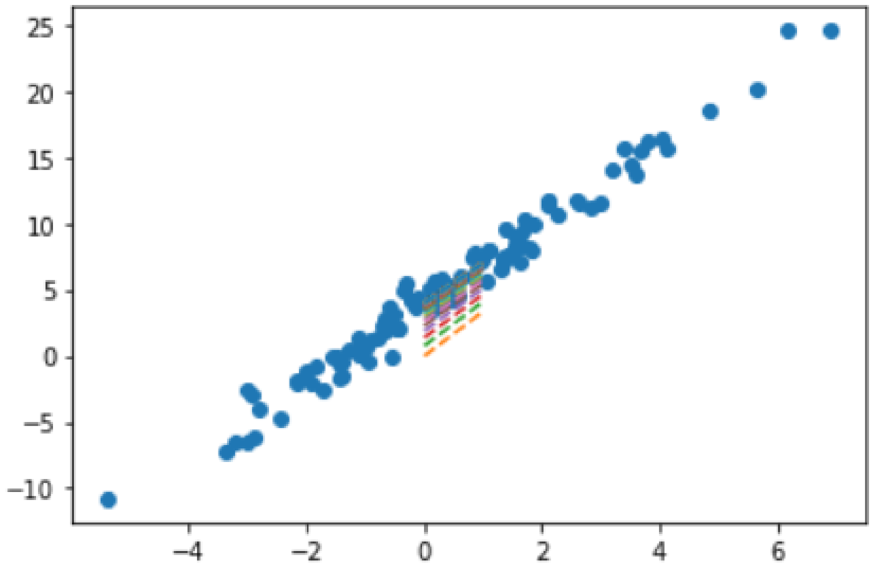
Gradient Descent Example

```
[21] print(Ws)

[array([[-0.02186189],
        [ 3.3604752 ]]), array([[0.75051536],
       [3.3257514 ]]), array([[1.37116953],
       [3.26374287]]), array([[1.87260791],
       [3.21314242]]), array([[2.27776943],
       [3.17224996]]), array([[2.60513995],
       [3.13920874]]), array([[2.86965535],
       [3.11251144]]), array([[3.08338381],
       [3.09094001]]), array([[3.25607642],
       [3.0735103 ]]), array([[3.39561206],
       [3.05942709]]), array([[3.50835687],
       [3.04804785]]), array([[3.59945468],
       [3.03885343]]), array([[3.67306172],
       [3.03142434]]), array([[3.73253621],
       [3.02542163]]), array([[3.7805916 ],
       [3.02057144]]), array([[3.81942036],
       [3.01665249]]), array([[3.850794  ],
       [3.01348597]]), array([[3.87614391],
       [3.01092743]]), array([[3.89662663],
       [3.00886013]]), array([[3.91317667],
       [3.00718975]]), array([[3.92654911],
       [3.00584008]]), array([[3.93735404],
       [3.00474955]]), array([[3.94608442],
       [3.0038684 ]]), array([[3.95313857],
       [3.00315643]]), array([[3.95883832],
       [3.00258116]]), array([[3.96344372],
       [3.00211634]]), array([[3.96716489],
       [3.00174076]]), array([[3.97017159],
       [3.0014373 ]]), array([[3.97260101],
```

```
[21] print(Ws)

       [3.00021239]]), array([[3.98240718],
       [3.00020237]]), array([[3.98248736],
       [3.00019428]]), array([[3.98255215],
       [3.00018774]]), array([[3.9826045 ],
       [3.00018246]]), array([[3.98264679],
       [3.00017819]]), array([[3.98268097],
       [3.00017474]]), array([[3.98270859],
       [3.00017195]]), array([[3.9827309],
       [3.0001697]]), array([[3.98274893],
       [3.00016788]]), array([[3.98276349],
       [3.00016641]]), array([[3.98277526],
       [3.00016522]]), array([[3.98278477],
       [3.00016426]]), array([[3.98279246],
       [3.00016349]]), array([[3.98279867],
       [3.00016286]]), array([[3.98280368],
       [3.00016235]]), array([[3.98280774],
       [3.00016194]]), array([[3.98281101],
       [3.00016161]]), array([[3.98281366],
       [3.00016135]]), array([[3.9828158 ],
       [3.00016113]]), array([[3.98281753],
       [3.00016096]]), array([[3.98281892],
       [3.00016082]]), array([[3.98282005],
       [3.0001607 ]]), array([[3.98282096],
       [3.00016061]]), array([[3.9828217 ],
       [3.00016053]]), array([[3.98282229],
       [3.00016047]]), array([[3.98282277],
       [3.00016043]]), array([[3.98282316],
       [3.00016039]]), array([[3.98282348],
       [3.00016036]]), array([[3.98282373],
```

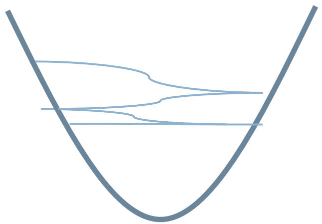
Gradient Descent Example

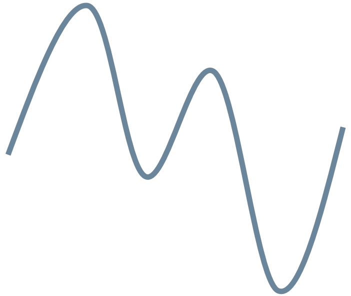
```
[19] plt.plot(x , y , 'o')
     for w in Ws:
       plt.plot(x_sample , x_sample_b.dot(w) , '--')
```



Gradient Descent Problems

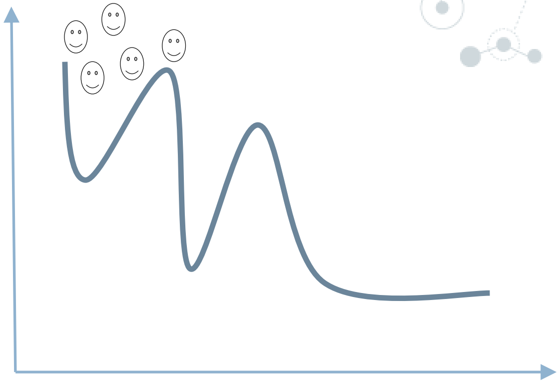
- Learning Rate too big
- Learning Rate too small
- Loss Function not convex

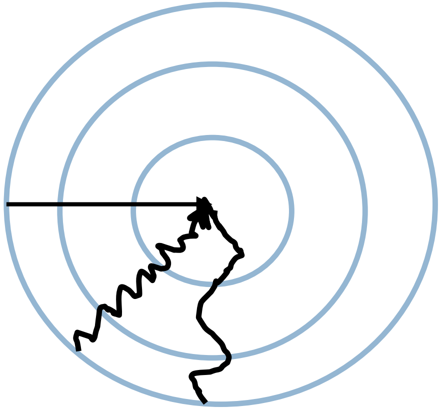




Gradient Descent Types

- Batch Gradient Descent
- Stochastic Gradient Descent
- Mini Batch Gradient Descent



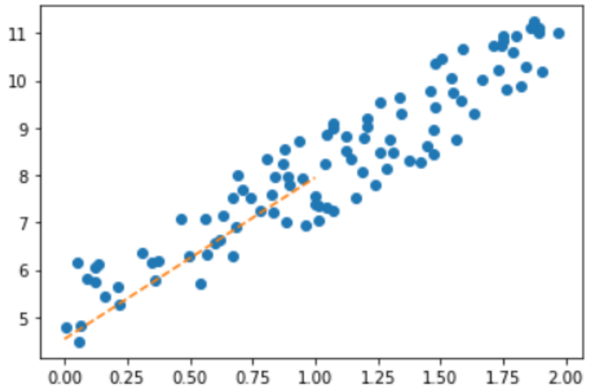


Stochastic Gradient Descent Implementation

```
[4] from sklearn.linear_model import SGDRegressor
    model = SGDRegressor()
    model.fit(x,y)
    y_prediction = model.predict(x_sample)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validati
  y = column_or_1d(y, warn=True)
```

```
plt.plot(x,y,'o')
plt.plot(x_sample, y_prediction, '--')
plt.show()
```



Gradient Descent Performance Measurements

Mean Absolute Error

```
[11] from sklearn.metrics import mean_absolute_error
     mean_absolute_error(y, y_prediction)

     0.5204412628339145
```

Mean Squared Error

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y, y_prediction)

0.389093613069074
```

RMSE

```
[21] import math
     math.sqrt(mean_squared_error(y, y_prediction))

     0.6237736873811478
```

R2 Score (accuracy)

```
[15] from sklearn.metrics import r2_score
     r2_score(y, y_prediction)

     0.8636412963739661
```

SVM(support vector machine)

- Supervised Algorithm

Linear classification with svm

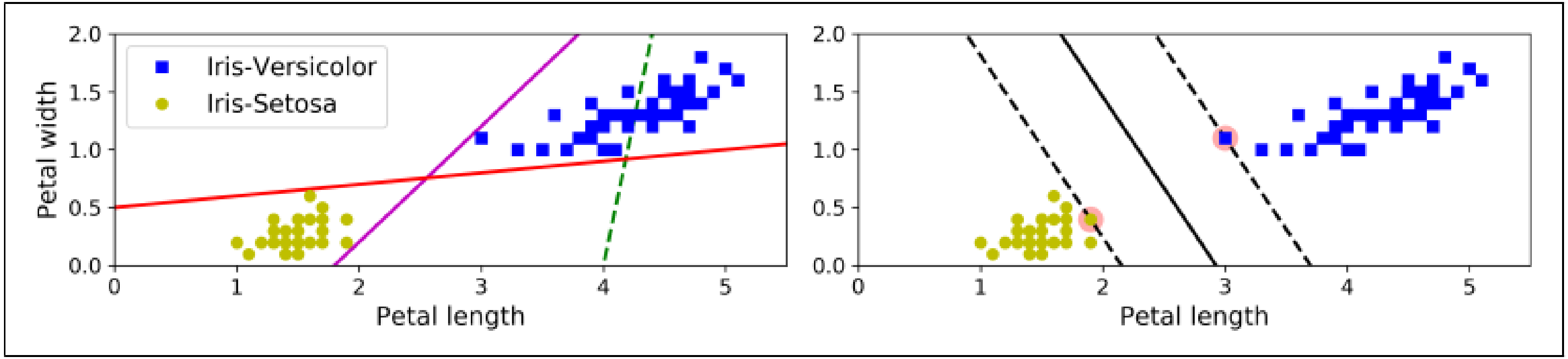
non Linear classification with svm

- Polynomial features
- Polynomial kernel
- Gaussian kernel
- RBF kernel
- Parameter tuning with grid search

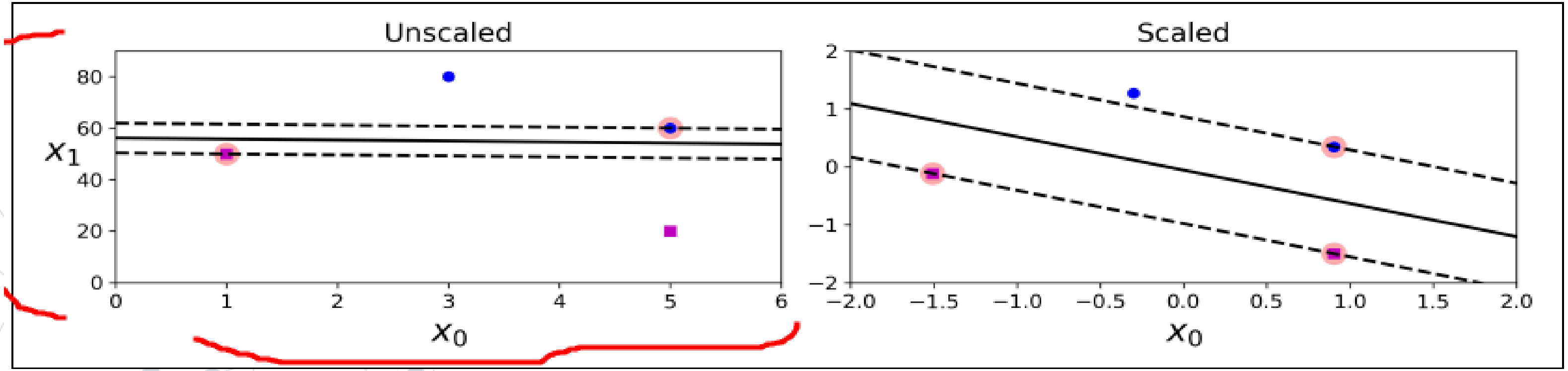
Regression with svm

Linear classification with SVM

- Best SVM



- Sensitive to the feature scales



SVM – C hyperparameter

- Hard margin
- Soft margin

- Large margin
- Fewer margin

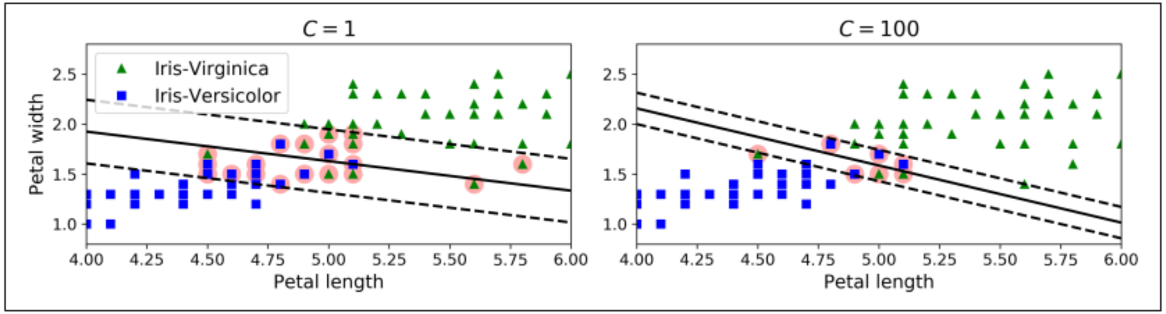


Figure 5-4. Large margin (left) versus fewer margin violations (right)

Linear SVM Example

```
[3] import numpy as np
    from sklearn import datasets
    from sklearn.pipeline import Pipeline
    from sklearn.preprocessing import StandardScaler
    from sklearn.svm import LinearSVC
```

```
iris = datasets.load_iris()
iris
```

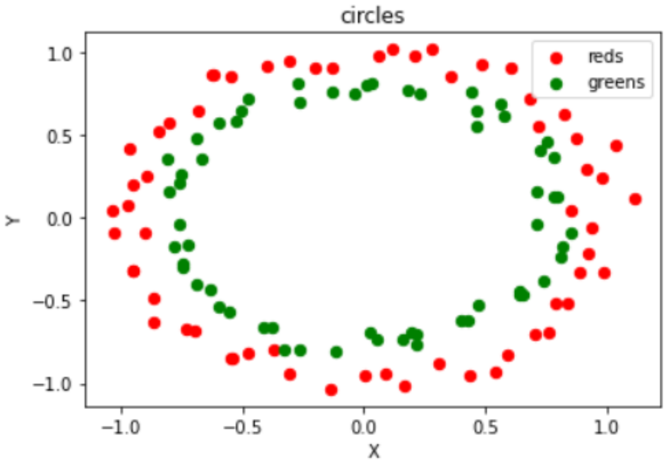
```
[20] x = iris['data'][: , (2,3)] # 'petal length (cm)','petal width (cm)'
     y = (iris['target'] == 2).astype(np.float64) #'versicolor'
     svm = Pipeline([
         ('Scaler' , StandardScaler()),
         ('linear_svc' , LinearSVC(C=20)),
     ])
     svm.fit(x,y)
     y_pred = svm.predict(x)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:1208: ConvergenceWarnin
  ConvergenceWarning,
```

```
from sklearn.metrics import f1_score
f1_score(y, y_pred)
```

```
0.94
```

Non Linear SVM Example

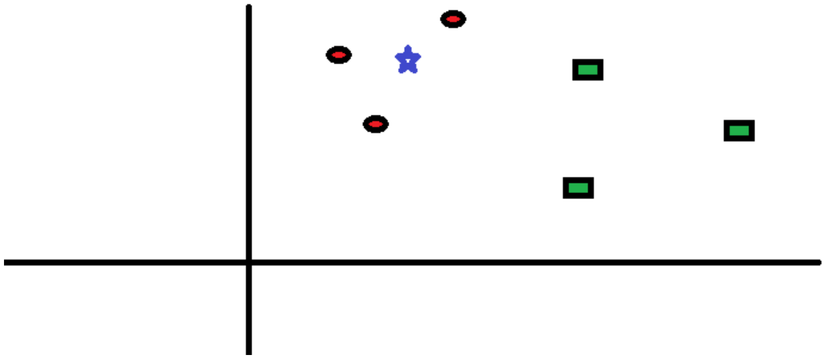


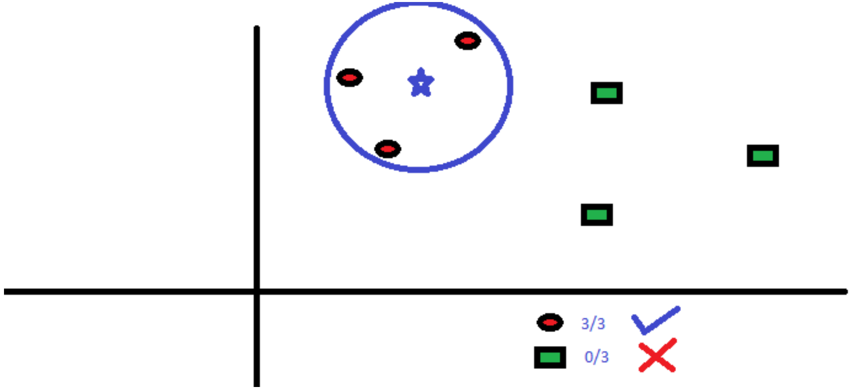

```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import f1_score

polynomial_svm = Pipeline([
    ("poly_features", PolynomialFeatures(degree=3)),
    ("scaler", StandardScaler()),
    ("svm_clf", LinearSVC(C=10))
])
polynomial_svm.fit(data, labels)
y_pred = polynomial_svm.predict(data)
f1_score(labels ,y_pred)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.
  ConvergenceWarning,
0.9917355371900827
```

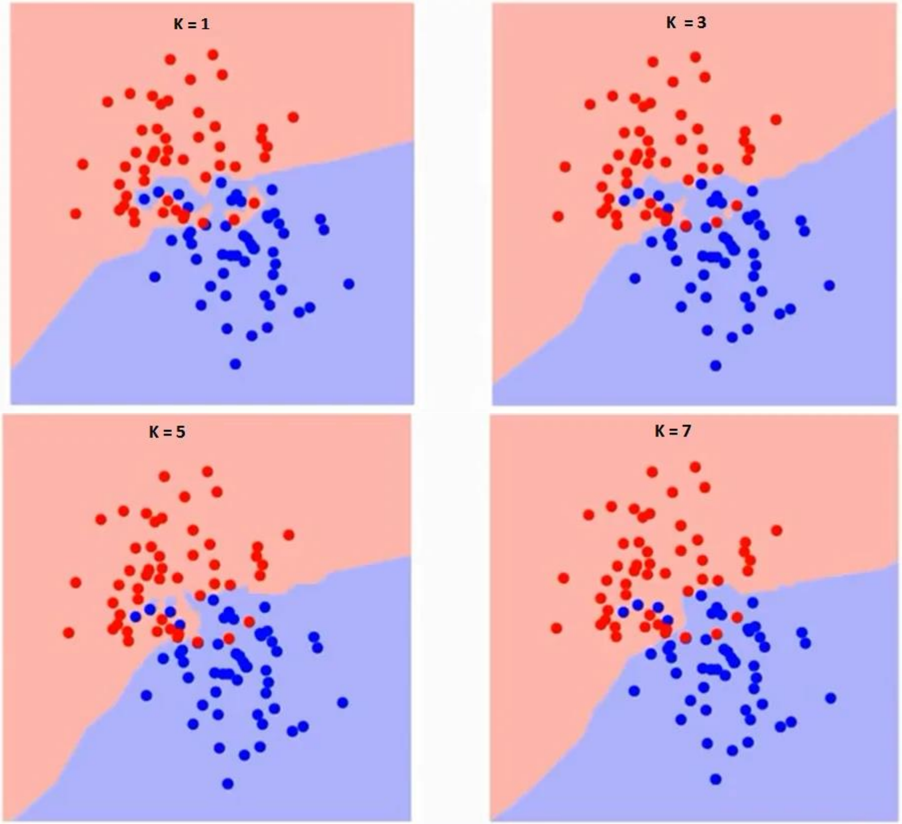
K-Nearest Neighbors

- Supervised algorithm
- Classification & Regression





K-Nearest Neighbors



K-Nearest Neighbors

Advantages

- The algorithm is simple and easy to implement.
- There’s no need to build a model, tune several parameters, or make additional assumptions.
- The algorithm is versatile. It can be used for classification, regression, and search (as we will see in the next section).

Disadvantages

- The algorithm gets significantly slower as the number of examples and/or predictors/independent variables increase.

KNN Example

[15]
```python
x = iris['data'][:, (2,3)]
y = iris['target']
```

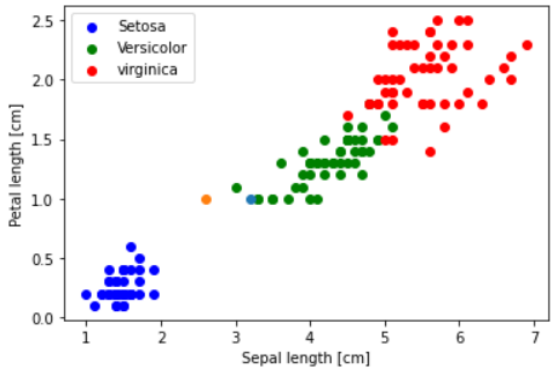
[26]
```python
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier (n_neighbors=3)
knn.fit(x,y)
x_test=[[3.2, 2.6]]
y_pred = knn.predict(x_test)
y_pred
```

```
array([1])
```

[27]
```python
print(knn.kneighbors(x_test)[1])
```

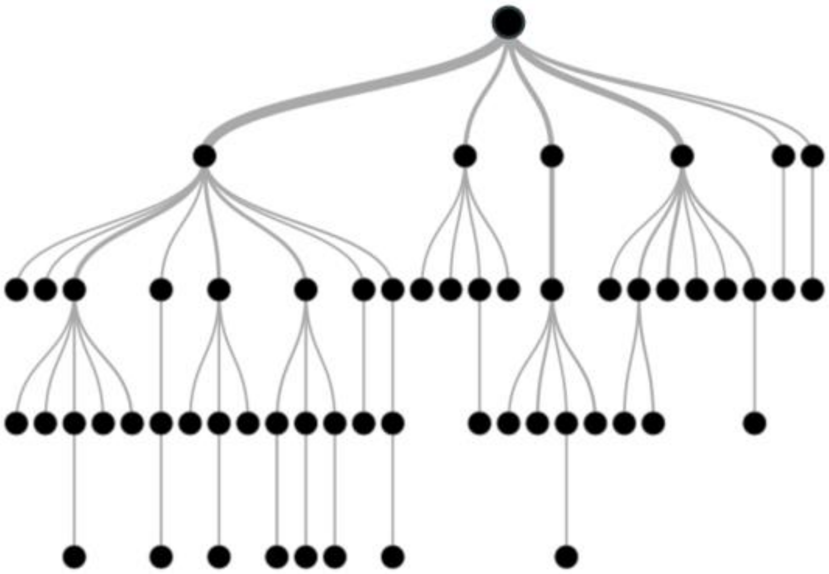
```
[[64 59 61]]
```

```python
plt.scatter(x[:50, 0], x[:50, 1],
            color='blue', marker='o', label='Setosa')
plt.scatter(x[50:100, 0], x[50:100, 1],
            color='green', marker='o', label='Versicolor')
plt.scatter(x[100:150, 0], x[100:150, 1],
            color='red', marker='o', label='virginica')
plt.plot(x_test, y_pred, 'o')
plt.xlabel('Sepal length [cm]')
plt.ylabel('Petal length [cm]')
plt.legend(loc='upper left')
plt.show()
```



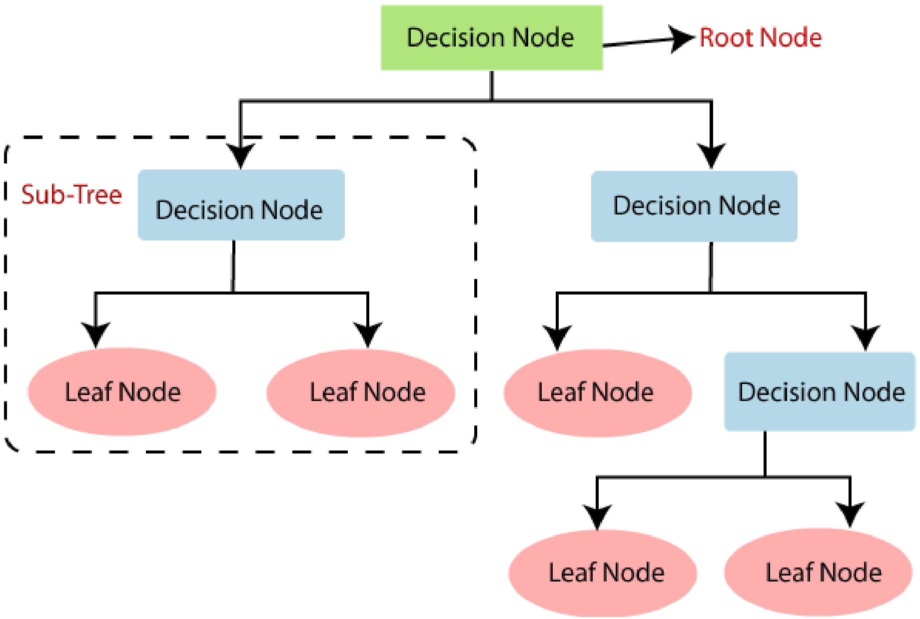
Decision Tree

- Supervised algorithm
- Classification & Regression



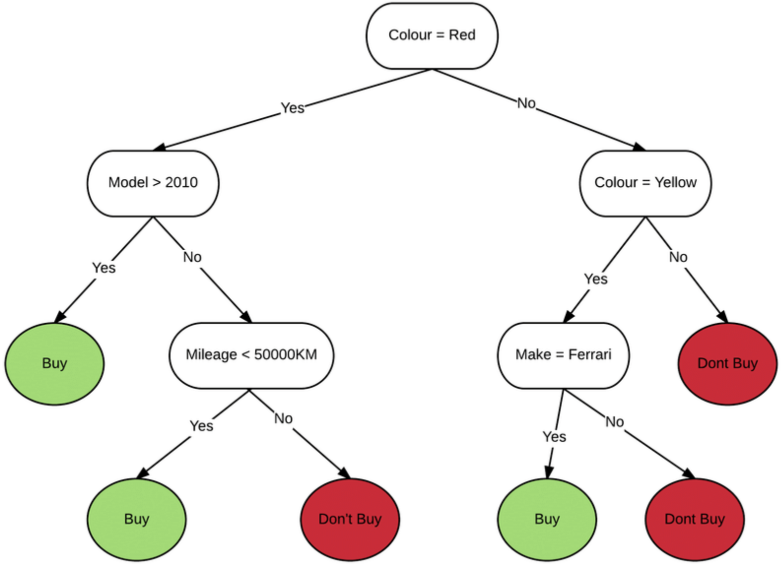
Decision Tree

- Root Node
- Splitting
- Decision Node
- Leaf / Terminal Node
- Pruning
- Sub-tree / Branch
- Parent & Child Node



Decision Tree

- Root node with full dataset
- Attribute Selection Measure
- Splitting
- Making the best decision node
- Continuing until reaching leaf



Decision Tree - Attribute Selection Measure

- Information Gain

  Information Gain=Entropy(S)−[(WeightedAvg)∗Entropy(eachfeature)

  Entropy(S)=−P(yes)log2P(yes)−P(no)log2P(n)

- Gini Index

  Gini Index=1−ΣjPj2

Types of Decision Trees

- **Classification Tree**

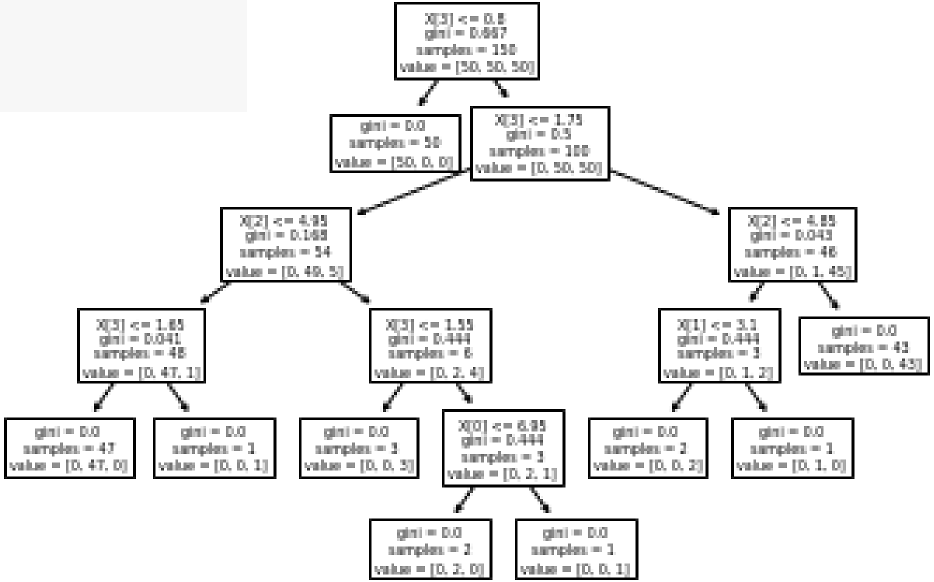
  classifying things into categories

- **Regression Tree**

  predicting numeric values

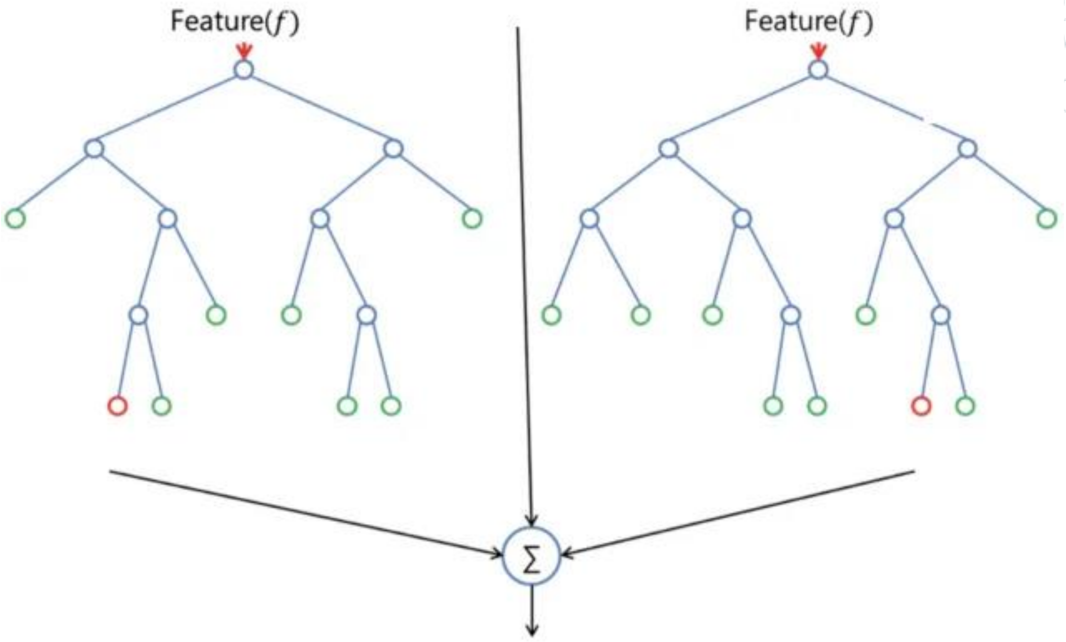
Decision Tree Example

```python
from sklearn import tree
X, y = iris.data, iris.target
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, y)
tree.plot_tree(clf)
```



Random Forest

- Supervised algorithm
- Classification & Regression



Random Forest vs. Decision Tree

Random Forest

- Advantages
  - Usable for both classification & regression
  - Easy to work with
  - Few hyperparameters(n-jobs, n-estimator)
- Disadvantages
  - Not being fast
  - High chance of overfitting

Random Forest Example

```
[41] X = df.drop(["case", "site", "Pop", "sex", 'Unnamed: 0'], axis=1)
     y = df["sex"]
```

```
[42] from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

     from sklearn.ensemble import RandomForestClassifier
     rf_model = RandomForestClassifier(n_estimators=50, max_features="auto")
     rf_model.fit(X_train, y_train)
```

```
RandomForestClassifier(n_estimators=50)
```

```
[43] y_predict = rf_model.predict(X_test)
     y_predict
```

```
array(['m', 'm', 'f', 'f', 'm', 'f', 'f', 'm', 'f', 'f', 'm', 'm', 'm',
       'm', 'm', 'm', 'm', 'f', 'm', 'm', 'm', 'm', 'm', 'm', 'f', 'm',
       'm', 'f', 'm', 'f', 'm'], dtype=object)
```

```
[45] from sklearn.metrics import accuracy_score
     accuracy_score(y_test, y_predict)
```

```
0.4838709677419355
```

THE END