MATHEMATICS FOR DATA SCIENCE

Instructor: Bharath B N

Title : Insurance risk prediction

Group No: 17

Group Members : Girish K Gupta, Arvind Raj S, Sivarama Krishna

Member contribution : All members have equally contributed in this work.

Accessing the given data set

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
data=[];
with open("/content/drive/MyDrive/MDS_Project2/data_set.data","r") as data_file:
    for line in data_file:
        fields=line.split('\n');
        del fields[-1]
        data.append(fields[0].split(','));
```

```
# Creating data frame for the given dataset
import pandas as pd
data_=pd.DataFrame(data,columns=['symboling','normalized-losses','make','fuel-type','aspir
                                 'engine-location','wheel-base','length','width','height',
                                'engine-size','fuel-system','bore','stroke','compression-r
```

```
data_.head()
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	dr wh
0	3	?	alfa-romero	gas	std	two	convertible	
1	3	?	alfa-romero	gas	std	two	convertible	
2	1	?	alfa-romero	gas	std	two	hatchback	
3	2	164	audi	gas	std	four	sedan	
4	2	164	audi	gas	std	four	sedan	

Converting categorical/text data into numeric values

```
list_dummy=['make','fuel-type','aspiration','num-of-cylinders','num-of-doors','body-style'
for j in list_dummy:
  l=data_[j].unique()
  c=0;dictionary={};
  for i in l:
    dictionary[i]=c;
    c+=1;
  for i in range(len(data_[j])):
    data_[j][i]=dictionary[data_[j][i]];
```

```
data_.head(10)
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels
0	3	?	0	0	0	0	0	0
1	3	?	0	0	0	0	0	0
2	1	?	0	0	0	0	1	0
3	2	164	1	0	0	1	2	1
4	2	164	1	0	0	1	2	2
5	2	?	1	0	0	0	2	1
6	1	158	1	0	0	1	2	1
7	1	?	1	0	0	1	3	1
8	1	158	1	0	1	1	2	1
9	0	?	1	0	1	0	1	2

Separating our target/label (i.e. symboling column)

```
data_=data_.replace('?',0)
y=data_['symboling'].astype(float)
data_=data_.drop('symboling',axis=1)
data_.columns
```

```
Index(['normalized-losses', 'make', 'fuel-type', 'aspiration', 'num-of-doors',
       'body-style', 'drive-wheels', 'engine-location', 'wheel-base', 'length',
       'width', 'height', 'curb-weight', 'engine-type', 'num-of-cylinders',
       'engine-size', 'fuel-system', 'bore', 'stroke', 'compression-ratio',
       'horsepower', 'peak-rpm', 'city-mpg', 'highway-mpg', 'price'],
      dtype='object')
```

Replacing NaN values with 0. Note: We can also try to replace these NaN with the mean values

```
import numpy as np
data=data_.replace('?',0)
X=data.astype(np.float)
```

Normalizing - Standardize features by removing the mean and scaling to unit variance.

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(data)
```

Calling the model, here we call logistic regression

```
from sklearn.linear_model import LogisticRegression
classifier=LogisticRegression(solver='liblinear',class_weight="balanced");
```

Splitting the data into train and test sets.

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y)
classifier.fit(X_train,y_train)
```

```
     LogisticRegression(C=1.0, class_weight='balanced', dual=False,
                        fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                        max_iter=100, multi_class='auto', n_jobs=None, penalty='l2',
                        random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                        warm_start=False)
```

Predicting symboling for test sample and finding the accuracy comparing with training data

```
y_hat=classifier.predict(X_test)
c=0;
for idx,i in enumerate(y_test):
    if(y_hat[idx]==i):
        c+=1;
print("Accuracy:",c/len(y_test))
```

```
     Accuracy: 0.47058823529411764
```

Repeating the same with L1 regulariser

```
from sklearn.linear_model import LogisticRegression
classifier=LogisticRegression(solver='liblinear',class_weight="balanced",penalty="l1");
```

```
from sklearn.model_selection import train_test_split
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y)
classifier.fit(X_train,y_train)
```

```
LogisticRegression(C=1.0, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='auto', n_jobs=None, penalty='l1',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
y_hat=classifier.predict(X_test)
c=0;
for idx,i in enumerate(y_test):
    if(y_hat[idx]==i):
        c+=1;
print("Accuracy:",c/len(y_test))
```

```
Accuracy: 0.6274509803921569
```

Testing using given test data point.

```
test=pd.read_csv('/content/drive/MyDrive/MDS_Project2/data_point.txt',header=None)
```

```
test.columns=['normalized-losses','make','fuel-type','aspiration','num-of-doors','body-sty
                                 'engine-location','wheel-base','length','width','height',
                                'engine-size','fuel-system','bore','stroke','compression-r
```

```
test
```

	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engin locati
0	101	honda	gas	std	two	hatchback	fwd	fro

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
data=[];
with open("/content/drive/MyDrive/MDS_Project2/data_set.data","r") as data_file:
    for line in data_file:
        fields=line.split('\n');
        del fields[-1]
        data.append(fields[0].split(','));
```

```
import pandas as pd
data_=pd.DataFrame(data,columns=['symboling','normalized-losses','make','fuel-type','aspir
                                 'engine-location','wheel-base','length','width','height',
                                'engine-size','fuel-system','bore','stroke','compression-r
```

Merging given data to existing data file

```
data1=pd.concat([data_,test])
```

```
data1
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style
0	3	?	alfa-romero	gas	std	two	convertible
1	3	?	alfa-romero	gas	std	two	convertible
2	1	?	alfa-romero	gas	std	two	hatchback
3	2	164	audi	gas	std	four	sedan
4	2	164	audi	gas	std	four	sedan
...	...	...	...	...	...	...	...
200	-1	95	volvo	gas	turbo	four	sedan
201	-1	95	volvo	gas	std	four	sedan
202	-1	95	volvo	diesel	turbo	four	sedan
203	-1	95	volvo	gas	turbo	four	sedan
0	NaN	101	honda	gas	std	two	hatchback

```
list_dummy=['make','fuel-type','aspiration','num-of-cylinders','num-of-doors','body-style'
for j in list_dummy:
  l=data_[j].unique()
  c=0;dictionary={};
  for i in l:
    dictionary[i]=c;
    c+=1;
  for i in range(len(data_[j])):
    data_[j][i]=dictionary[data_[j][i]];
```

```
data_.head()
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels
0	3	?	0	0	0	0	0	0
1	3	?	0	0	0	0	0	0
2	1	?	0	0	0	0	1	0
3	2	164	1	0	0	1	2	1
4	2	164	1	0	0	1	2	2

```
data_.head(10)
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels
0	3	?	0	0	0	0	0	0
1	3	?	0	0	0	0	0	0
2	1	?	0	0	0	0	1	0
3	2	164	1	0	0	1	2	1
4	2	164	1	0	0	1	2	2
5	2	?	1	0	0	0	2	1
6	1	158	1	0	0	1	2	1
7	1	?	1	0	0	1	3	1
8	1	158	1	0	1	1	2	1
9	0	?	1	0	1	0	1	2

```
data_=data_.replace('?',0)
y=data_['symboling'].astype(float)
data_=data_.drop('symboling',axis=1)
data_.columns
```

```
    Index(['normalized-losses', 'make', 'fuel-type', 'aspiration', 'num-of-doors',
           'body-style', 'drive-wheels', 'engine-location', 'wheel-base', 'length',
           'width', 'height', 'curb-weight', 'engine-type', 'num-of-cylinders',
           'engine-size', 'fuel-system', 'bore', 'stroke', 'compression-ratio',
           'horsepower', 'peak-rpm', 'city-mpg', 'highway-mpg', 'price'],
          dtype='object')
```

```
import numpy as np
data=data_.replace('?',0)
X=data.astype(np.float)
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(data)

from sklearn.linear_model import LogisticRegression
classifier=LogisticRegression(solver='liblinear',class_weight="balanced");
```

```
X[0:-1,:].shape
```

```
(203, 25)
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X[0:-1,:],y[0:-1])
classifier.fit(X_train,y_train)
```

```
LogisticRegression(C=1.0, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
y_hat=classifier.predict(X_test)
c=0;
for idx,i in enumerate(y_test):
    if(y_hat[idx]==i):
        c+=1;
print("Accuracy:",c/len(y_test))
```

```
Accuracy: 0.5294117647058824
```

```
X[-1,:].shape
```

```
(25,)
```

```
totest=np.expand_dims(X[-1,:],axis=-1).T

y_test=classifier.predict(totest)
print(y_test)
```

```
[-2.]
```

```
from sklearn.linear_model import LogisticRegression
classifier=LogisticRegression(solver='liblinear',class_weight="balanced",penalty="l1");
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X[0:-1,:],y[0:-1])
classifier.fit(X_train,y_train)
```

```
LogisticRegression(C=1.0, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='auto', n_jobs=None, penalty='l1',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
y_hat=classifier.predict(X_test)
c=0;
for idx,i in enumerate(y_test):
    if(y_hat[idx]==i):
        c+=1;
print("Accuracy:",c/len(y_test))
```

```
     Accuracy: 0.5294117647058824
```

```
totest=np.expand_dims(X[-1,:],axis=-1).T

y_test=classifier.predict(totest)
print(y_test)
```

```
     [-2.]
```

✓ 0s completed at 11:13 PM ● ×