7- Scikit Learn - Linear Modeling Let us begin by understanding what is Linear Regression in Sklearn.

7-1: Linear Regression studies the relationship between a dependent variable(Y) with a given set of independent variables(X).

7-2: Logistic Regression is a classification algorithm rather than regression algorithm. Based on a given set of independent variables, it is used to estimate discrete value(0 or 1, yes/no, true/false)

7-3: Ridge Regression or Tikhonov regularization technique that perform L2 regularization. It modifies the loss function by adding the penalty equivalent to the square of the magnitude of coefficients.

7-4: Bayesian Ridge Regression allows a natural mechanism to survive insufficient data or poorly distributed data by formulating linear regression using probability distributors rather than point estimates.

7-5: LASSO is the regularization technique that perform L1 regularization. It modifies the loss function by adding the penalty equivalent to the summation of the absolute value of coefficients.

7-6: Multi-task LASSO allows to fit multiple regression problems jointly enforcing the selected features to be same for all the regression problems, also called tasks. Sklearn provides a linear model named MultiTaskLasso, trained with a mixed L1, L2-norm for regularization, which estimates sparse coefficients for multiple regression problems jointly.

7-7: Elastic-Net is a regularized regression method that linearly combines both penalties. It is useful when there are multiple correlated features.

7-8: Multi-task Elastic-Net allows to fit multiple regression problems jointly enforcing the selected features to be same for all the regression problems, also called tasks.

8- Stochastic Gradient Descent

SGD is an optimization algorithm in Sklearn. This algorithm used to find the values of parameters/coefficients of functions that minimize a cost function. It has been successfully applied to large-scale datasets because the update to the coefficients is performed for each training instance, rather than at the end of instance.

Stochastic Gradient Descent classifier basically implements a plain SGD learning routine supporting various loss functions and penalties for classification. Scikit-learn provides **SGDClassifier** module to implement SGD classification.

8-1- Implementation Example

Like other Classifiers, SGD has to be fitted with following two arrays:

- An array X holding the training samples. It is of size[n_samples,n_features]
- An array Y holding the target values. It is of size[n_samples]

```
import numpy as np
from sklearn import linear_model
X=np.array([[-1,1],[-2,-3],[2,3],[1,2]])
Y=np.array([1,2,5,1])
SGDCLF=linear_model.SGDClassifier(max_iter=1000,tol=1e-3,penalty='elasticnet')
SGDCLF.fit(X,Y)
print(SGDCLF.predict([[2,1]]))
print(SGDCLF.coef_)
```

output:

```
[5]
[[-19.33273544  38.59729232]
 [-19.54811198 -29.32421683]
 [ 37.79761953  -9.39254474]]
```

Scikit-learn provides **SGDRegressor** module to implement SGD regression.

9- Support Vector Machines

Support Vector Machines(SVM) used for classification, regression, and , outlier's detection. SVMs are very efficient in high dimensional spaces and generally are used in classification problems.

The main goal of SVMs is to divide the datasets into number of classes in order to find a ***maximum marginal hyperplane(MMH)*** which can done in the following two steps:

- Support Vector Machines will first generate hyperplanes iteratively that separates the classes in the best way.
- After that it will choose the hyperplane that separate the classes correctly.

Scikit-learn provides three classes namely **SVC,NuSVC,LinearSVC** which can perform multi-class classification.

9-1: SVC

This module used by scikit-learn is **sklearn.svm.svc**. This class handles the multiclass support according to one-vs-one scheme.

9-1-1:Parameters

- **C**: it is the penalty parameter of the error term, default=0
- **kernel**: it specifies the type of kernel to be used in algorithm, we can choose among **'linear','poly','rbf','sigmoid','precomputed'**.
- **degree**: it represents the degree of the 'poly' kernel function and will be ignored by all other kernels.
- **gamma**: it is kernel coefficient for kernels 'rbf','poly' and 'sigmoid'.
- **coef0**: an independent term in kernel function which is only significant in 'poly' and 'sigmoid'.
- **tol**: this parameter represents the stopping criterion for iterations.
- **shrinking**: this parameter represents that whether we want to use shrinking heuristic or not.
- **probability**: this parameter enables or disables probability estimates.
- **cache_size**: this parameter will specify the size of the kernel cache. The value will be in MB(MegaBytes).

- **random_state**: this parameter represents the seed of the psedu random number generated which is used while shuffling the data.
- **class_weight**: this parameter will set the parameter C of class j to *class_weight[j]*C* for SVC.

9-1-2:Implementation Example

```
import numpy as np
from sklearn.svm import SVC
X=np.array([[-1,1],[-2,-3],[2,3],[1,2]])
y=np.array([1,2,2,1])
SVCCLF=SVC(kernel='linear',gamma='scale',shrinking=False)
SVCCLF.fit(X,y)
print(SVCCLF.predict([[-0.5,-0.8]]))
print(SVCCLF.coef_)
```

output:

```
[2]
[[ 0.58823529 -0.64705882]]
```

9-2: NuSVC

It is another class provided by scikit-learn which can perform multi-class classification. it is like SVC but NuSVC accepts slightly different sets of parameters:

- **nu** represents an upper bound on the fraction of training errors and a lower bound of the fraction of suport vectors. its value should be in the interval of (0,1].

9-2-1:Implementation Example

```
import numpy as np
from sklearn.svm import NuSVC
X=np.array([[-1,1],[-2,-3],[2,3],[1,2]])
y=np.array([1,2,2,1])
NuSVCCLF=NuSVC(kernel='linear',gamma='scale',shrinking=False)
NuSVCCLF.fit(X,y)
print(NuSVCCLF.predict([[-0.5,-0.8]]))
print(NuSVCCLF.coef_)
```

output:

```
[2]
[[ 2.         -1.33333333]]
```

9-3: LinearSVC

It is Linear Support Vector Classification. It is similar to SVC having kernel='linear'. The difference between them is that **LinearSVC** implemented in terms of liblinear while SVC is implemented in **libsvm**. That's the reason **LinearSVC** has more flexibility in the choice of penalties and loss functions. It also scales better to large number of samples.

9-3-1:Implementation Example

```
import numpy as np
from sklearn.svm import LinearSVC
X=np.array([[-1,1],[-2,-3],[2,3],[1,2]])
y=np.array([1,2,2,1])
LSVCClf=LinearSVC(dual=False,random_state=0,penalty='l1',tol=1e-5)
LSVCClf.fit(X,y)
print(LSVCClf.predict([[-0.5,-0.8]]))
print(LSVCClf.coef_)
```

output:

```
[2]
[[ 0.44804731 -0.38311391]]
```

9-4: Regression with SVM

SVM is used for both classification and regression problems. Scikit-learn's method of Support Vector Classification (SVC) can be extended to solve regression problems as well. That extended method is called Support Vector Regression (SVR).

Scikit-learn provides three classes namely **SVR,NuSVR and LinearSVR** as three different implementation of SVR.

9-4-1:SVR

It is epsilon-support vector regression whose implementation is based on **libsvm**. here, parameter **epsilon** represents the epsilon-SVR model, and specifies the epsilon-tube within within which no penalty is associated in the training loss function with points predicted within a distance epsilon from the actual value.

```
from sklearn.svm import SVR
X=[[1,2],[3,4]]
y=[1,3]
SVRREG=SVR(kernel='linear',gamma='auto')
SVRREG.fit(X,y)
print(SVRREG.predict([[1,2]]))
```

output:

[1.1]

9-4-2:NuSVR

It is like NuSVC, but NuSVR uses a parameter **nu** to control the number of support vectors. And moreover, unlike NuSVC where **nu** replaced C parameter, here it replaces **epsilon**.

```
from sklearn.svm import NuSVR
import numpy as np
n_samples,n_features=20,15
np.random.seed(0)
X=np.random.randn(n_samples,n_features)
y=np.random.randn(n_samples)
NuSVRReg=NuSVR(kernel='linear',gamma='auto',C=1.0,nu=0.1)
NuSVRReg.fit(X,y)
print(NuSVRReg.coef_)
```

output:

```
[[ 0.23180317  0.13549427 -0.38437291  0.21767392 -0.67694681 -0.01261807
  -0.2347462   0.16879733 -0.35119401 -0.21719007 -0.14536424  0.05250018
   0.18204661  0.08182123  0.26425696]]
```

9-5: LinearSVR

It is similar to SVR having kernel='linear'. The difference between them is that LinearSVR implemented in terms of **liblinear**. while SVC implemented in **libsvm**. That's the reason **LinearSVR** has more flexibility in the choice of penalties and loss functions. It also scales better to large number of samples. Also it does not support **'kernel'** because it is assumed to be linear.

```
from sklearn.svm import LinearSVR
from sklearn.datasets import make_regression
X,y=make_regression(n_features=4,random_state=0)
LSVRReg=LinearSVR(dual=False,random_state=0, loss='squared_epsilon_insensitive',tol=1e-5)
LSVRReg.fit(X,y)
print(LSVRReg.predict([[0,1,2,1]]))
print(LSVRReg.coef_)
```

output:

```
[256.00973816]
[20.47354746 34.08619401 67.23189022 87.47017787]
```