machine-learning-assignment-3

January 10, 2024

Q1. What are different regression techniques? State the usage of different regression techniques. *Ans=* Regression techniques are fundamental tools in machine learning, used to predict continuous numerical values based on given input features.

Here are some common regression techniques and their usage:

Linear Regression: Linear regression is a simple and widely used technique that assumes a linear relationship between the input features and the target variable.It is used for predicting continuous numerical values when the relationship between the input features and the target variable is approximately linear.

Polynomial Regression: Polynomial regression extends linear regression by allowing for non-linear relationships between the input features and the target variable. It is used when the relationship is known to be non-linear or when a simple linear model is not performing well.

Ridge Regression: Ridge regression is a regularization technique that helps prevent overfitting in linear regression models. It adds a penalty term to the loss function, penalizing large coefficients and preventing the model from becoming too complex.

Lasso Regression: Lasso regression is another regularization technique that is similar to ridge regression, but it uses an L1 norm penalty instead of an L2 norm penalty. This can lead to sparsity in the model, meaning that some coefficients become zero, effectively eliminating those features from the model.

Logistic Regression: Logistic regression is a classification technique that is sometimes used for regression tasks, particularly when the target variable is binary (e.g., yes or no). It is used to predict the probability of an event occurring based on the input features.

Support Vector Regression: Support Vector Regression (SVR) is a non-linear regression technique that uses support vectors to define a hyperplane that minimizes the prediction error. It is used when the relationship between the input features and the target variable is non-linear or when the data is noisy.

Decision Tree Regression: Decision tree regression is a tree-based regression technique that builds a decision tree to predict the target variable. It is a non-parametric technique, meaning that it does not make assumptions about the underlying relationship between the input features and the target variable.

Random Forest Regression: Random forest regression is an ensemble learning technique that combines multiple decision trees to make predictions. It is often used to improve the accuracy and reduce the overfitting of individual decision trees.

K-Nearest Neighbors Regression: K-nearest neighbors regression is a non-parametric regression technique that predicts the target variable for a new data point by averaging the target values of its k nearest neighbors. It is a simple and effective technique, but it can be computationally expensive for large datasets.

Neural Networks for Regression: Neural networks, particularly deep neural networks, can be used for regression tasks.They can model complex non-linear relationships between the input features and the target variable. These regression techniques have diverse applications across various domains, such as: Predicting housing prices based on features like location, size, and amenities Forecasting stock prices based on historical data and market trends Estimating customer churn based on customer behavior and demographics Predicting patient outcomes based on medical history and treatment data Optimizing machine performance by predicting performance metrics based on configuration parameters.

Q2.Take the pizza dataset. Apply Linear regression to predict the price of the pizza. Compare results using evaluation measures.

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split as tts
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
#laoding datasets
df=pd.read_csv("pizza.csv")
df.info()
#allocation
X=df.drop(["rupee"],axis=1)
y=df["rupee"]
#splitting data into training and testing
X_train,X_test,y_train,y_test=tts(X,y,test_size=0.25,random_state=40)
#creating Linear Regression model
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
#training the model
lr.fit(X_train,y_train)
#prediting data
y_pred=lr.predict(X_test)
#Evaluation the model
mae=mean_absolute_error(y_test,y_pred)
mse=mean_squared_error(y_test,y_pred)
r2=r2_score(y_test,y_pred)
#printing evaluation output
print("mean_absolute_error:",mae)
print("mean_squared_error:",mse)
print("R_squared:",r2)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
```

```
---  ------    --------------  -----
 0   id        5 non-null      int64
 1   diameter  5 non-null      int64
 2   rupee     5 non-null      int64
dtypes: int64(3)
memory usage: 248.0 bytes
mean_absolute_error: 20.000000000000014
mean_squared_error: 500.0000000000003
R_squared: 0.4444444444444441
```

Q3. Take SocialNetworkAd data, perform Lostic Regression on it.

[2]:
```
import pandas as pd
from sklearn.model_selection import train_test_split as tts
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
df=pd.read_csv("Social_Network_Ads.csv")
#df.info()
#df.head()
X=df.drop(["EstimatedSalary"],axis=1)
y=df["EstimatedSalary"]
#splittting data into training and testing
X_train,X_test,y_train,y_test=tts(X,y,test_size=0.25,random_state=140)
#model
le=LogisticRegression()
#training the data with model
le.fit(X_train,y_train)
#prediction of model
y_pred=le.predict(X_test)
#valuation
from sklearn import metrics as mat
r2=mat.r2_score(y_test,y_pred)
mae=mat.mean_absolute_error(y_test,y_pred)
mse=mat.mean_absolute_error(y_test,y_pred)
#printing output
print("R square of Logistic Regression :",r2)
print("mean_absolute_error of Logistic Regression:",mae)
print("mean_absolute_error of Logistic Regression:",mse)
```

```
R square of Logistic Regression : -4.719290690613498e-05
mean_absolute_error of Logistic Regression: 28780.0
mean_absolute_error of Logistic Regression: 28780.0
```

Q4. Do Ridge,Lasso and Elastic Net regression on Hitters data.

[3]:
```
import pandas as pd
from sklearn.model_selection import train_test_split as tts
from sklearn.linear_model import LinearRegression
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import Ridge,Lasso,ElasticNet
```

[5]:
```
df=pd.read_csv("Hitters.csv")
#df.head()
#preprocessing=EDA
#df.isnull().sum()
A=df[df["League"]=="A"]
A.isnull().sum()
N=df[df["League"]=="N"]
N.isnull().sum()
```

[5]:
```
AtBat         0
Hits          0
HmRun         0
Runs          0
RBI           0
Walks         0
Years         0
CAtBat        0
CHits         0
CHmRun        0
CRuns         0
CRBI          0
CWalks        0
League        0
Division      0
PutOuts       0
Assists       0
Errors        0
Salary       23
NewLeague     0
dtype: int64
```

[6]:
```
df1=df.dropna()
df1.isnull().sum()
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
#changing the text to numner
df1["League"]=le.fit_transform(df1["League"])
df1["NewLeague"]=le.fit_transform(df1["NewLeague"])
df1["Division"]=le.fit_transform(df1["Division"])
#assume target as "Salary" features and target
X=df1.drop("Salary",axis=1)
y=df1[["Salary"]]
```

```
#splitting for training and testing
x_train,x_test,y_train,y_test=tts(X,y,test_size=0.25,random_state=140)
#Theorem and models
rid=Ridge()
lass=Lasso()
enet=ElasticNet()
#Training data
rid.fit(x_train,y_train)
lass.fit(x_train,y_train)
enet.fit(x_train,y_train)
#prediction of testing sets
y_rid=rid.predict(x_test)
y_lass=lass.predict(x_test)
y_net=enet.predict(x_test)
#evaluation
mae_ridge=mean_absolute_error(y_test,y_rid)
mse_lasso=mean_squared_error(y_test,y_lass)
r2_elastic=r2_score(y_test,y_net)
#printing output of RIDGE
print("Mean absolute error of Ridge:",mae_ridge)
print("Mean absolute error of Lasso:",mse_lasso)
print("Mean absolute error of ElasticNet:",r2_elastic)
#
print("Mean Square error of Ridge:",mae_ridge)
print("Mean Square error of Lasso:",mse_lasso)
print("Mean Square error of ElasticNet:",r2_elastic)
print("R2_score of Ridge:",mae_ridge)
print("R2_score of Lasso:",mae_ridge)
print("R2_score of ElasticNet:",mae_ridge)
```

```
Mean absolute error of Ridge: 266.2991208958147
Mean absolute error of Lasso: 154791.5913688277
Mean absolute error of ElasticNet: 0.16502635633286522
Mean Square error of Ridge: 266.2991208958147
Mean Square error of Lasso: 154791.5913688277
Mean Square error of ElasticNet: 0.16502635633286522
R2_score of Ridge: 266.2991208958147
R2_score of Lasso: 266.2991208958147
R2_score of ElasticNet: 266.2991208958147
```

```
<ipython-input-6-fbccd6c96973>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df1["League"]=le.fit_transform(df1["League"])
<ipython-input-6-fbccd6c96973>:7: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df1["NewLeague"]=le.fit_transform(df1["NewLeague"])
<ipython-input-6-fbccd6c96973>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df1["Division"]=le.fit_transform(df1["Division"])
/usr/local/lib/python3.10/dist-
packages/sklearn/linear_model/_coordinate_descent.py:631: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 7.612e+06, tolerance: 4.067e+03
  model = cd_fast.enet_coordinate_descent(
/usr/local/lib/python3.10/dist-
packages/sklearn/linear_model/_coordinate_descent.py:631: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 7.788e+06, tolerance: 4.067e+03
  model = cd_fast.enet_coordinate_descent(
```

5. What is SVM? State the difference between Linear and Non-linear SVC. *Ans=* Support Vector Machines Support vector machines (SVM) are another approach to predictive modeling that is based on error-based learning. The instance nearest the decision boundary, based on perpendicular distance, is known as the margin. The dashed lines on either side of the decision boundary show the extent of the margin, and we refer to these as the margin extents. – Training a support vector machine involves searching for the decision boundary, or separating hyperplane, that leads to the maximum margin because this will best separate the levels of the target feature. The instances in a training dataset that fall along the margin extents, and therefore the margins, are known as the support vectors. We define the separating hyperplane as: w0+w.d=0. – Instances above a separating hyperplane is defined as: w0+w.d>0 Instances below a separating hyperplane is defined as: w0+w.d<0 When training a support vector machine, we wish to find a hyperplane that distinguishes between the two target levels, −1 and +1. – Linear SVM Classification The two classes can clearly be separated easily with a straight line (they are linearly separable) SVM classifier as fitting the widest possible street between the classes, is called large margin classification. Soft Margin Classification. There are two main issues with hard margin classification. First, it only works if the data is linearly separable, and second it is quite sensitive to outliers. The objective is to find a good balance between keeping the street as large as possible and limiting the margin violations is called soft margin classification. To control this balance using the C hyperparameter: a smaller C value leads to a wider street but more margin violations. Nonlinear SVM Classification Handling nonlinear datasets is to add more features, such as polynomial features in some cases this can result in a linearly separable dataset. To implement this idea using Scikit-Learn, you can create a Pipeline containing a PolynomialFeatures transformer. Eg. pipe = Pipeline([('scaler',

StandardScaler()), ('svc', SVC())]) Polynomial Kernel While using PolynomialFeatures, with a high polynomial degree it creates a huge number of features, making the model too slow Mathematical technique called the kernel trick is used for that. Eg. poly = PolynomialFeatures(degree=2) Adding Similarity Features Another technique to tackle nonlinear problems is to add features computed using a similarity function that measures how much each instance resembles a particular landmark. Using Gaussian RBF kernel similarity measures are added. Eg. svm_rbf = SVC(kernel='rbf', C=1.0, gamma='scale')

Q6. Implement all SVM methods in iris data.

[8]:
```
import numpy as np
from sklearn.model_selection import train_test_split as tts
from sklearn import datasets
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.metrics import accuracy_score
from sklearn.svm import LinearSVC, SVC

iris = datasets.load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = tts(X, y, test_size=0.20, random_state=20)

svm_clf = Pipeline([
    ("scaler", StandardScaler()),
    ("linear_svc", LinearSVC(C=1, loss="hinge"))
])

svm_clf.fit(X_train, y_train)
y_pred_linear = svm_clf.predict(X_test)
accuracy_linear = accuracy_score(y_test, y_pred_linear)

poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

svm_poly = SVC(kernel="poly", degree=2, C=2.0, gamma="scale")
svm_poly.fit(X_train_poly, y_train)
y_pred_poly = svm_poly.predict(X_test_poly)
accuracy_poly = accuracy_score(y_test, y_pred_poly)

svm_rbf = SVC(kernel="rbf", gamma="scale", C=2.0)
svm_rbf.fit(X_train, y_train)
y_pred_rbf = svm_rbf.predict(X_test)
accuracy_rbf = accuracy_score(y_test, y_pred_rbf)

svm_clf.fit(X, y)
```

```
a = svm_clf.predict([[5.4, 3.9, 1.7, 0.4]])
a1 = svm_clf.predict([[7.2, 3.6, 6.1, 2.5]])

print("Accuracy of Linear SVM:", accuracy_linear)
print("Accuracy of Polynomial SVM:", accuracy_poly)
print("Accuracy of RBF SVM:", accuracy_rbf)
print("Support Vector Machine (Sample 1):", a)
print("Support Vector Machine (Sample 2):", a1)
```

```
Accuracy of Linear SVM: 0.9333333333333333
Accuracy of Polynomial SVM: 0.9666666666666667
Accuracy of RBF SVM: 1.0
Support Vector Machine (Sample 1): [0]
Support Vector Machine (Sample 2): [2]
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  warnings.warn(
```