← Go Back to Machine Learning

Course Content

# New Package Introduction - Introduction to Supervised Learning and Classification

Some of the common functions used from the Scikit-learn library in the case studies related to the third lecture are listed below:

**LDA**

The aim of LDA is to maximize the between-class variance and minimize the within-class variance, through a linear discriminant function, under the assumption that data in every class are described by a Gaussian Probability Density Function with the same covariance.

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
sklearn.discriminant_analysis.LinearDiscriminantAnalysis(solver='svd', shrinkage=None, priors=None, n_components
```

Example:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
model = LinearDiscriminantAnalysis()
model.fit(xtrain,ytrain)
```

You can learn more about the LinearDiscriminantAnalysis() class that is available in sklearn documentation for a better understanding of the parameters and attributes.

**QDA**

Quadratic Discriminant Analysis (QDA) is a generative model. QDA assumes that each class follows a Gaussian distribution. The class-specific prior is simply the proportion of data points that belong to the class. The class-specific mean vector is the average of the input variables that belong to the class.

```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis(*, priors=None, reg_param=0.0, store_covariance=Fals
```

Example:

```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
model = QuadraticDiscriminantAnalysis()
model.fit(xtrain,ytrain)
```

You can learn more about the QuadraticDiscriminantAnalysis() class that is available in sklearn documentation for a better understanding of the parameters and attributes.

**KNN**

The k-nearest neighbors (KNN) algorithm is a simple, supervised machine learning algorithm that can be used to solve both classification and regression problems. It's easy to implement and understand, but has a major drawback of becoming significantly slows as the size of that data in use grows.

```
from sklearn.neighbors import KNeighborsClassifier
sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2
```

Example:

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
model.fit(xtrain,ytrain)
```

You can learn more about the KNeighborsClassifier() class that is available in sklearn documentation for a better understanding of the parameters and attributes.

### Classification Evaluation Metrics

A **Classification report** is used to measure the quality of predictions from a classification algorithm. How many predictions are True and how many are False. More specifically, True Positives, False Positives, True negatives and False Negatives are used to predict the metrics of a classification report.

A **confusion matrix** is a table that is used to define the performance of a classification algorithm. A confusion matrix visualizes and summarizes the performance of a classification algorithm.

The **precision-recall curve** is constructed by calculating and plotting the precision against the recall for a single classifier at a variety of thresholds. For example, if we use logistic regression, the threshold would be the predicted probability of an observation belonging to the positive class.

### Importing the functions

```
from sklearn.metrics import classification_report, confusion_matrix, precision_recall_curve
```

Example

```
from sklearn.metrics import classification_report, confusion_matrix, precision_recall_curve
print(classification_report(actual, predictions))
print(confusion_matrix(actual, predictions))
precision_recall_curve(actual, predictions)
```

You can refer to this official documentation where you can explore different classification evaluation metrics, their individual parameters, and their attributes.

### Model Tuning

**GridSearchCV** is a technique to search through the best parameter values from the given set of the grid of parameters. It is basically a cross-validation method. the model and the parameters are required to be fed in. Best parameter values are extracted and then the predictions are made.

```
# importing the GridSearchCV
from sklearn.model_selection import GridSearchCV
sklearn.model_selection.GridSearchCV(estimator, param_grid, *, scoring=None, n_jobs=None, refit=True, cv=None, v
```

You can learn more about the GridSearchCV() class that is available in sklearn documentation for a better understanding of the parameters and attributes.

Happy Learning!

 Help