**3-QUADRIC CLASSIFIER**

**A Quadratic classifier is a** [**statistical classifier**](https://en.wikipedia.org/wiki/Statistical_classification) **that uses a** [**quadratic**](https://en.wikipedia.org/wiki/Quadratic_function)[**decision surface**](https://en.wikipedia.org/wiki/Decision_boundary) **to separate measurements of two or more classes of objects or events. 8It is a more general version of the** [**linear classifier**](https://en.wikipedia.org/wiki/Linear_classifier)**.**

**A quadratic classifier is used in machine learning to separate measurements of two or more classes of objects or events by a quadric surface.**

**The purpose of classification or discriminant analysis is to analyse the observation-based set of measurements to classify the objects into one of several groups or classes. Based on the loss function the discriminant analysis is categorised as** [**Linear Discriminant Analysis**](https://analyticsindiamag.com/a-hands-on-guide-to-linear-discriminant-analysis-for-binary-classification/) **and Quadratic Discriminant Analysis.**

* **Linear Discriminant Analysis or Discriminant Function Analysis is a dimensionality reduction technique that is commonly used**

**for supervised classification problems.**

* **Quadratic discriminant analysis is quite similar to Linear discriminant analysis except we relaxed the assumption that the mean and covariance of all the classes were equal. Therefore, we required to calculate it separately.**

| **import numpy** |  |
| --- | --- |
|  |  |
|  | **class QDA:** |
|  | **def fit(self, X, t):** |
|  | **self.priors = dict()** |
|  | **self.means = dict()** |
|  | **self.covs = dict()** |
|  |  |
|  | **self.classes = np.unique(t)** |
|  |  |
|  | **for c in self.classes:** |
|  | **X\_c = X[t == c]** |
|  | **self.priors[c] = X\_c.shape[0] / X.shape[0]** |
|  | **self.means[c] = np.mean(X\_c, axis=0)** |
|  | **self.covs[c] = np.cov(X\_c, rowvar=False)** |
|  |  |
|  | **def predict(self, X):** |
|  | **preds = list()** |
|  | **for x in X:** |
|  | **posts = list()** |
|  | **for c in self.classes:** |
|  | **prior = np.log(self.priors[c])** |
|  | **inv\_cov = np.linalg.inv(self.covs[c])** |
|  | **inv\_cov\_det = np.linalg.det(inv\_cov)** |
|  | **diff = x-self.means[c]** |
|  | **likelihood = 0.5\*np.log(inv\_cov\_det) - 0.5\*diff.T @ inv\_cov @ diff** |
|  | **post = prior + likelihood** |
|  | **posts.append(post)** |
|  | **pred = self.classes[np.argmax(posts)]** |
|  | **preds.append(pred)** |
|  | **return np.array(preds)** |