Statistical Learning for Data Science

Lecture 13 Appendix

唐晓颖
电子与电气工程系
南方科技大学

April 22, 2023

使用scikit-learn对有标签的数据做LDA

当使用Python的Scikit-Learn库执行线性判别分析（LDA）时，首先需要导入LDA类和数据集。然后，需要将数据集分成训练集和测试集。接下来，可以使用fit方法拟合模型，使用predict方法预测模型，并使用score方法评估模型的准确性。

```python
# 导入 LDA 类和数据集
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.datasets import load_iris

# 加载数据集
iris = load_iris()
X = iris.data
y = iris.target

# 将数据集分成训练集和测试集
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

```python
# 创建 LDA 模型
lda = LinearDiscriminantAnalysis()

# 使用 fit 方法拟合模型
lda.fit(X_train, y_train)

# 使用 predict 方法预测模型
y_pred = lda.predict(X_test)

# 使用 score 方法评估模型的准确性
accuracy = lda.score(X_test, y_test)
print("Accuracy:", accuracy)
```

```
(m2mrf) bash-4.2$ /raid5/huangwk/anaconda3/envs/DL/bin/python /raid5/huangwk/code/test.py
Accuracy: 0.9777777777777777
```

通过python中的help( ) 函数打开帮助文档

```
>>> import sklearn.discriminant_analysis
>>> help(sklearn.discriminant_analysis)
Help on module sklearn.discriminant_analysis in sklearn:

NAME
    sklearn.discriminant_analysis - Linear Discriminant Analysis and Quadratic Discriminant Analysis

CLASSES
    sklearn.base.BaseEstimator(builtins.object)
        LinearDiscriminantAnalysis(sklearn.base.BaseEstimator, sklearn.linear_model.base.LinearClassifierMixin, sklearn.base.TransformerMixin)
        QuadraticDiscriminantAnalysis(sklearn.base.BaseEstimator, sklearn.base.ClassifierMixin)
    sklearn.base.ClassifierMixin(builtins.object)
        QuadraticDiscriminantAnalysis(sklearn.base.BaseEstimator, sklearn.base.ClassifierMixin)
    sklearn.base.TransformerMixin(builtins.object)
        LinearDiscriminantAnalysis(sklearn.base.BaseEstimator, sklearn.linear_model.base.LinearClassifierMixin, sklearn.base.TransformerMixin)
    sklearn.linear_model.base.LinearClassifierMixin(sklearn.base.ClassifierMixin)
        LinearDiscriminantAnalysis(sklearn.base.BaseEstimator, sklearn.linear_model.base.LinearClassifierMixin, sklearn.base.TransformerMixin)

    class LinearDiscriminantAnalysis(sklearn.base.BaseEstimator, sklearn.linear_model.base.LinearClassifierMixin, sklearn.base.TransformerMixin)
     |  LinearDiscriminantAnalysis(solver='svd', shrinkage=None, priors=None, n_components=None, store_covariance=False, tol=0.0001)
     |
     |  Linear Discriminant Analysis
     |
```

```
class LinearDiscriminantAnalysis(sklearn.base.BaseEstimator, sklearn.linear_model.base.LinearClassifierMixin, sklearn.base.TransformerMixin)
 |  LinearDiscriminantAnalysis(solver='svd', shrinkage=None, priors=None, n_components=None, store_covariance=False, tol=0.0001)
 |
 |  Linear Discriminant Analysis
 |
 |  A classifier with a linear decision boundary, generated by fitting class
 |  conditional densities to the data and using Bayes' rule.
 |
 |  The model fits a Gaussian density to each class, assuming that all classes
 |  share the same covariance matrix.
 |
 |  The fitted model can also be used to reduce the dimensionality of the input
 |  by projecting it to the most discriminative directions.
 |
 |  .. versionadded:: 0.17
 |     *LinearDiscriminantAnalysis*.
 |
 |  Read more in the :ref:`User Guide <lda_qda>`.
 |
 |  Parameters
 |  ----------
 |  solver : string, optional
 |      Solver to use, possible values:
 |        - 'svd': Singular value decomposition (default).
 |          Does not compute the covariance matrix, therefore this solver is
 |          recommended for data with a large number of features.
 |        - 'lsqr': Least squares solution, can be combined with shrinkage.
 |        - 'eigen': Eigenvalue decomposition, can be combined with shrinkage.
 |
 |  shrinkage : string or float, optional
 |      Shrinkage parameter, possible values:
```

```
Parameters
----------
solver : string, optional
    Solver to use, possible values:
      - 'svd': Singular value decomposition (default).
        Does not compute the covariance matrix, therefore this solver is
        recommended for data with a large number of features.
      - 'lsqr': Least squares solution, can be combined with shrinkage.
      - 'eigen': Eigenvalue decomposition, can be combined with shrinkage.

shrinkage : string or float, optional
    Shrinkage parameter, possible values:
      - None: no shrinkage (default).
      - 'auto': automatic shrinkage using the Ledoit-Wolf lemma.
      - float between 0 and 1: fixed shrinkage parameter.

    Note that shrinkage works only with 'lsqr' and 'eigen' solvers.

priors : array, optional, shape (n_classes,)
    Class priors.

n_components : int, optional (default=None)
    Number of components (<= min(n_classes - 1, n_features)) for
    dimensionality reduction. If None, will be set to
    min(n_classes - 1, n_features).

store_covariance : bool, optional
    Additionally compute class covariance matrix (default False), used
    only in 'svd' solver.

    .. versionadded:: 0.17

tol : float, optional, (default 1.0e-4)
    Threshold used for rank estimation in SVD solver.

    .. versionadded:: 0.17
```

```
Notes
-----
The default solver is 'svd'. It can perform both classification and
transform, and it does not rely on the calculation of the covariance
matrix. This can be an advantage in situations where the number of features
is large. However, the 'svd' solver cannot be used with shrinkage.

The 'lsqr' solver is an efficient algorithm that only works for
classification. It supports shrinkage.

The 'eigen' solver is based on the optimization of the between class
scatter to within class scatter ratio. It can be used for both
classification and transform, and it supports shrinkage. However, the
'eigen' solver needs to compute the covariance matrix, so it might not be
suitable for situations with a high number of features.
```

```
Examples
--------
>>> import numpy as np
>>> from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
>>> X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
>>> y = np.array([1, 1, 1, 2, 2, 2])
>>> clf = LinearDiscriminantAnalysis()
>>> clf.fit(X, y)  # doctest: +NORMALIZE_WHITESPACE
LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=None,
              solver='svd', store_covariance=False, tol=0.0001)
>>> print(clf.predict([[-0.8, -1]]))
[1]
```

scikit-learn官方文档

scikit learn Install User Guide API Examples Community More

Prev Up Next

scikit-learn 1.2.2
Other versions

Please **cite us** if you use the software.

sklearn.discriminant_analysis.**LinearDiscriminantAnalysis**
LinearDiscriminantAnalysis
Examples using sklearn.discriminant_analysis.Line

sklearn.discriminant_analysis.LinearDiscriminantAnalysis

class sklearn.discriminant_analysis.**LinearDiscriminantAnalysis**(*solver='svd', shrinkage=None, priors=None, n_components=None, store_covariance=False, tol=0.0001, covariance_estimator=None*) [source]

Linear Discriminant Analysis.

A classifier with a linear decision boundary, generated by fitting class conditional densities to the data and using Bayes' rule.

The model fits a Gaussian density to each class, assuming that all classes share the same covariance matrix.

The fitted model can also be used to reduce the dimensionality of the input by projecting it to the most discriminative directions, using the `transform` method.

New in version 0.17: LinearDiscriminantAnalysis.

Read more in the User Guide.

https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html#sklearn.discriminant_analysis.LinearDiscriminantAnalysis
https://scikit-learn.org/stable/modules/lda_qda.html#lda-qda

Parameters:

solver : {'svd', 'lsqr', 'eigen'}, default='svd'

Solver to use, possible values:

- 'svd': Singular value decomposition (default). Does not compute the covariance matrix, therefore this solver is recommended for data with a large number of features.
- 'lsqr': Least squares solution. Can be combined with shrinkage or custom covariance estimator.
- 'eigen': Eigenvalue decomposition. Can be combined with shrinkage or custom covariance estimator.

Changed in version 1.2: `solver="svd"` now has experimental Array API support. See the Array API User Guide for more details.

shrinkage : 'auto' or float, default=None

Shrinkage parameter, possible values:

- None: no shrinkage (default).
- 'auto': automatic shrinkage using the Ledoit-Wolf lemma.
- float between 0 and 1: fixed shrinkage parameter.

This should be left to None if `covariance_estimator` is used. Note that shrinkage works only with 'lsqr' and 'eigen' solvers.

priors : array-like of shape (n_classes,), default=None

The class prior probabilities. By default, the class proportions are inferred from the training data.

n_components : int, default=None

Number of components (<= min(n_classes - 1, n_features)) for dimensionality reduction. If None, will be set to min(n_classes - 1, n_features). This parameter only affects the `transform` method.

store_covariance : bool, default=False

If True, explicitly compute the weighted within-class covariance matrix when solver is 'svd'. The matrix is always computed and stored for the other solvers.

New in version 0.17.

tol : float, default=1.0e-4

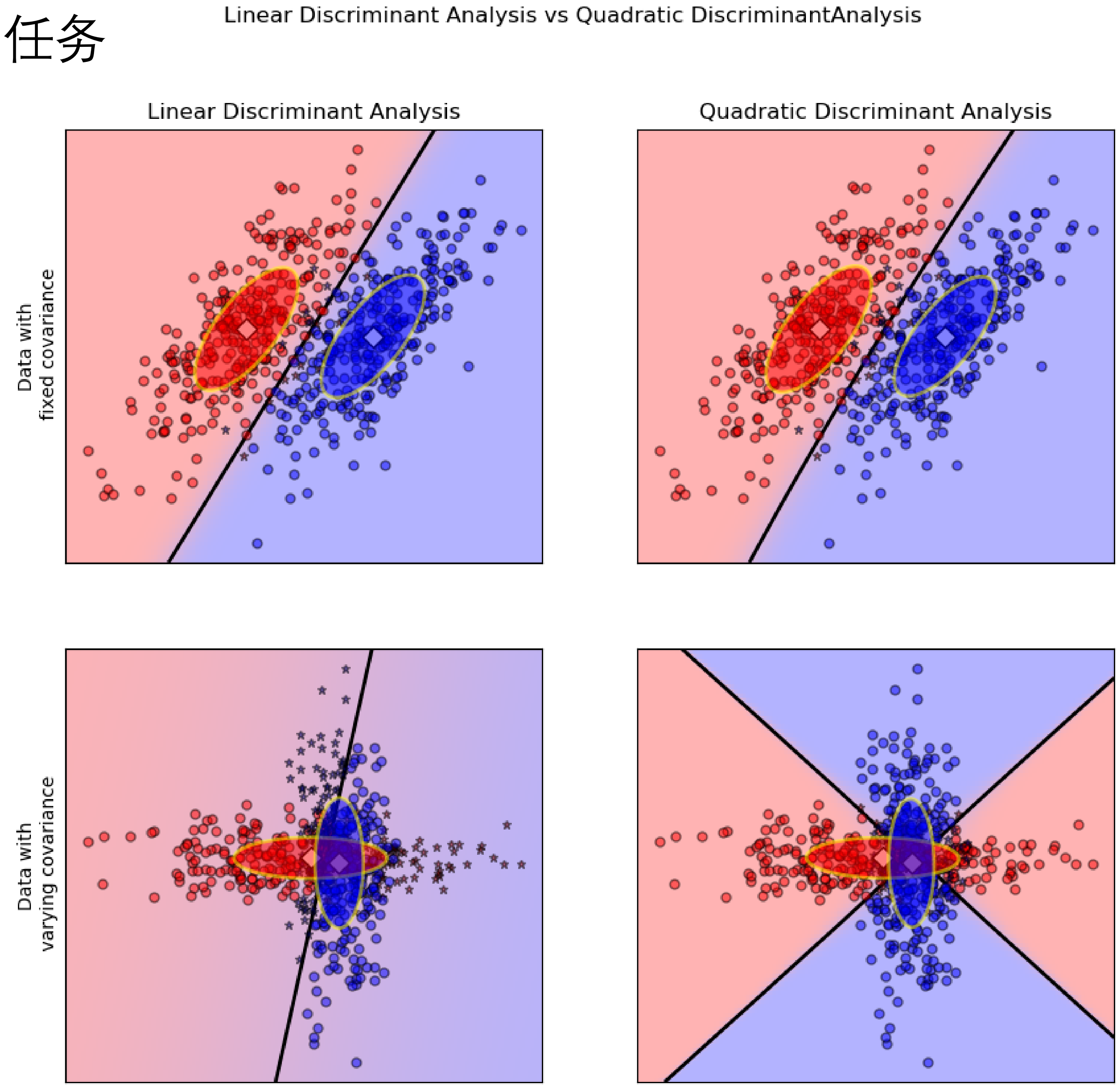
Examples

```
>>> import numpy as np
>>> from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
>>> X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
>>> y = np.array([1, 1, 1, 2, 2, 2])
>>> clf = LinearDiscriminantAnalysis()
>>> clf.fit(X, y)
LinearDiscriminantAnalysis()
>>> print(clf.predict([[-0.8, -1]]))
[1]
```

Methods

`decision_function`(X)	Apply decision function to an array of samples.
`fit`(X, y)	Fit the Linear Discriminant Analysis model.
`fit_transform`(X[, y])	Fit to data, then transform it.
`get_feature_names_out`([input_features])	Get output feature names for transformation.
`get_params`([deep])	Get parameters for this estimator.
`predict`(X)	Predict class labels for samples in X.
`predict_log_proba`(X)	Estimate log probability.
`predict_proba`(X)	Estimate probability.
`score`(X, y[, sample_weight])	Return the mean accuracy on the given test data and labels.
`set_output`(*[, transform])	Set output container.
`set_params`(**params)	Set the parameters of this estimator.
`transform`(X)	Project data to maximize class separation.

用散点图表示出分类任务



https://scikit-learn.org/stable/auto_examples/classification/plot_lda_qda.html

```python
from scipy import linalg
import  numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib import  colors

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
```

Scipy: 科学计算

Numpy：矩阵运算

Matplotlib：绘图工具

SKlearn：算法库

```python
mean1 = [0,0]
cov1 = [[10,0],[0,1]]
mean2 = [1,0]
cov2 = [[1,0],[0,10]]

def dataset_cov():
    #生成坐标数据
    n, dim = 300, 2
    np.random.seed(0)

    X1=np.random.multivariate_normal(mean1,cov1,n)
    X2=np.append(X1,np.random.multivariate_normal(mean2,cov2,n),0)
    #生成标签数据
    Y = np.hstack((np.zeros(n), np.ones(n)))
    return X2, Y
```

生成随机数据

Mean: 中心

Cov：协方差矩阵

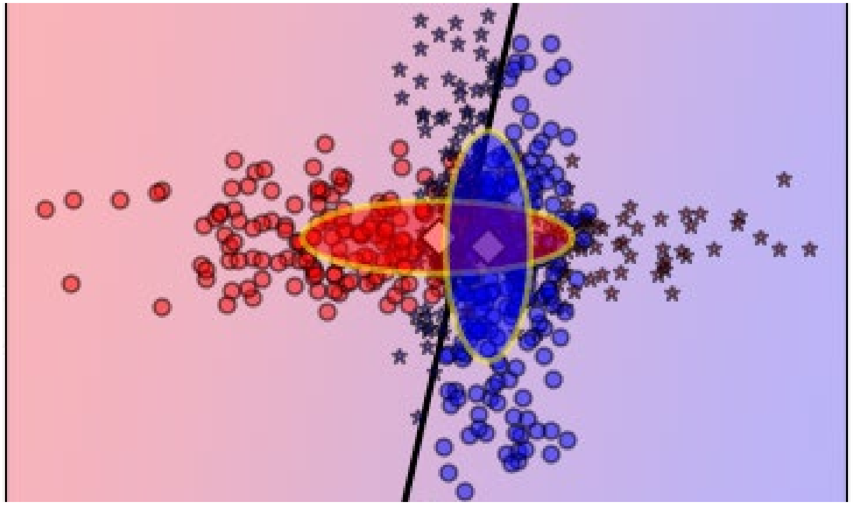
Point number：点数

不同分类的样本点有不同的颜色和形状

```python
tp = (Y==Y_pred)#找到预测成功的样本点
tp0,tp1 = tp[Y==0],tp[Y==1]#将预测成功的样本点根据y分类，得到的是索引
X0,X1= X[Y==0],X[Y==1]#将所有样本点按照y分类
X0_tp,X0_fp = X0[tp0],X0[~tp0]
#得到预测成功的y=0样本点的X值和预测失败的样本点的X值
X1_tp,X1_fp = X1[tp1],X1[~tp1]

alpha = 0.5
plt.plot(X0_tp[:,0],X0_tp[:,1],'o',alpha=alpha,color='red', markersize=5,
         markeredgecolor='k')
plt.plot(X0_fp[:,0],X0_fp[:,1],'*',alpha=alpha,color='#990000',  markersize=5,
         markeredgecolor = 'k')

plt.plot(X1_tp[:,0],X1_tp[:,1],'o',alpha=alpha,color='blue', markersize=5,
         markeredgecolor='k')
plt.plot(X1_fp[:,0],X1_fp[:,1],'*',alpha=alpha,color='#000099',  markersize=5,
         markeredgecolor = 'k')
```

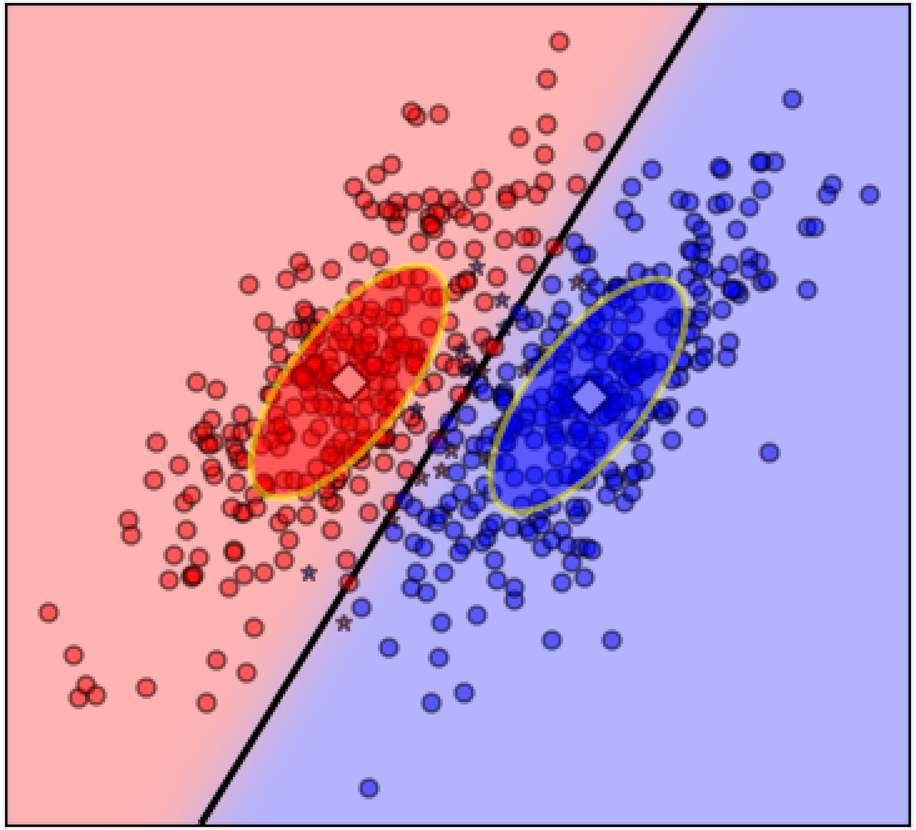


```
**Markers**

=============    ===============================
character        description
=============    ===============================
``'.'``          point marker
``','``          pixel marker
``'o'``          circle marker
``'v'``          triangle_down marker
``'^'``          triangle_up marker
``'<'``          triangle_left marker
``'>'``          triangle_right marker
``'1'``          tri_down marker
``'2'``          tri_up marker
``'3'``          tri_left marker
``'4'``          tri_right marker
``'s'``          square marker
``'p'``          pentagon marker
``'*'``          star marker
``'h'``          hexagon1 marker
``'H'``          hexagon2 marker
``'+'``          plus marker
``'x'``          x marker
``'D'``          diamond marker
``'d'``          thin_diamond marker
``'|'``          vline marker
``'_'``          hline marker
=============    ===============================
```

分区颜色表示、决策分界、高斯分布中心

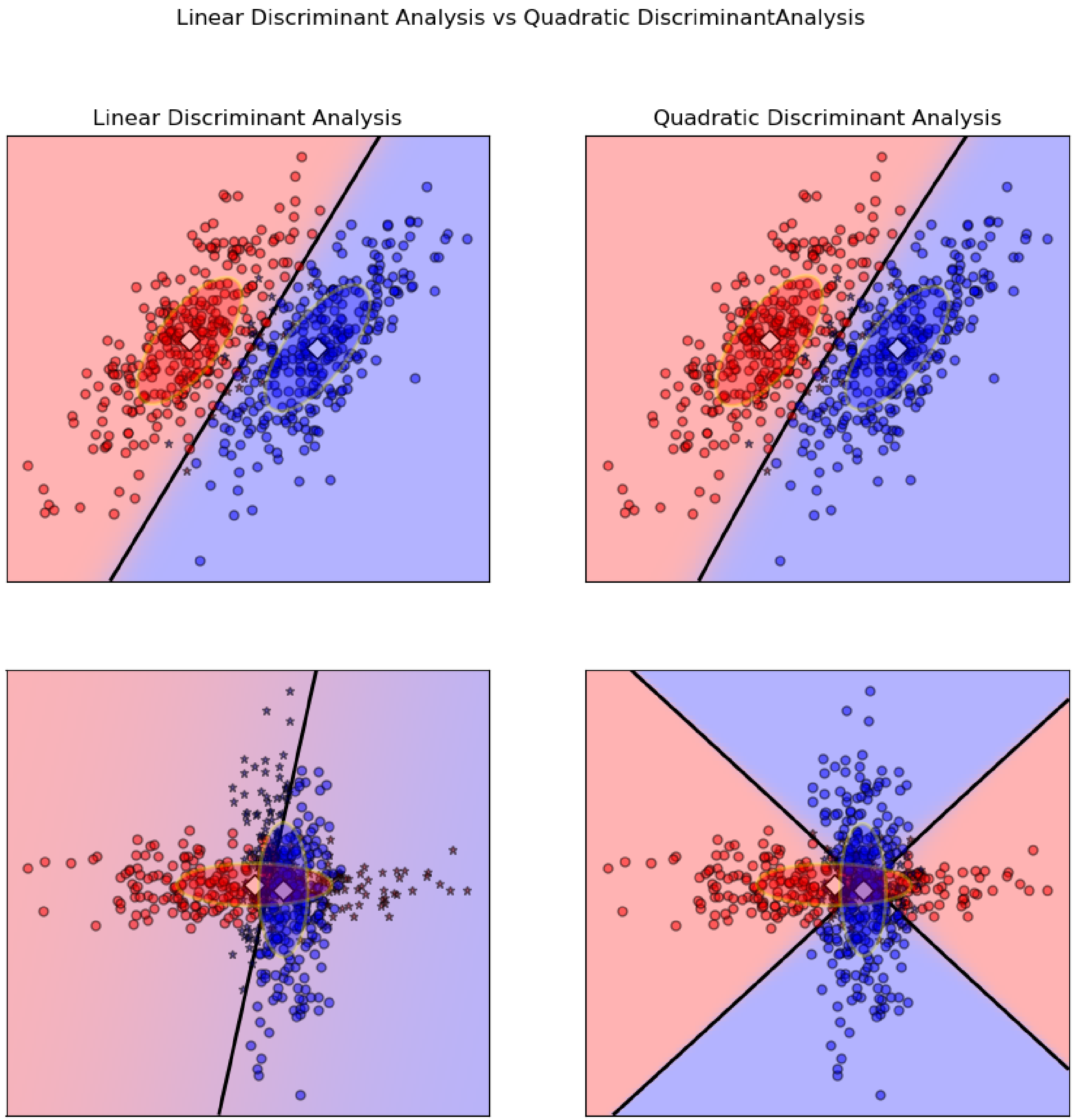
```python
#mesh building
nx,ny = 200,100
x_min,x_max = plt.xlim()
y_min,y_max = plt.ylim()
xx,yy = np.meshgrid(np.linspace(x_min,x_max,nx),
                    np.linspace(y_min,y_max,ny))
z = lda.predict_proba(np.c_[xx.ravel(),yy.ravel()])
z = z[:,1].reshape(xx.shape)
plt.pcolormesh(xx,yy,z,cmap='red_blue_classes',
               norm=colors.Normalize(0,1))
plt.contour(xx,yy,z,[0.5],linewidths = 2,colors='k')
plt.plot(lda.means_[0][0], lda.means_[0][1],
         'D', color='white', markersize=8, markeredgecolor='k')
plt.plot(lda.means_[1][0], lda.means_[1][1],
         'D', color='white', markersize=8, markeredgecolor='k')
```

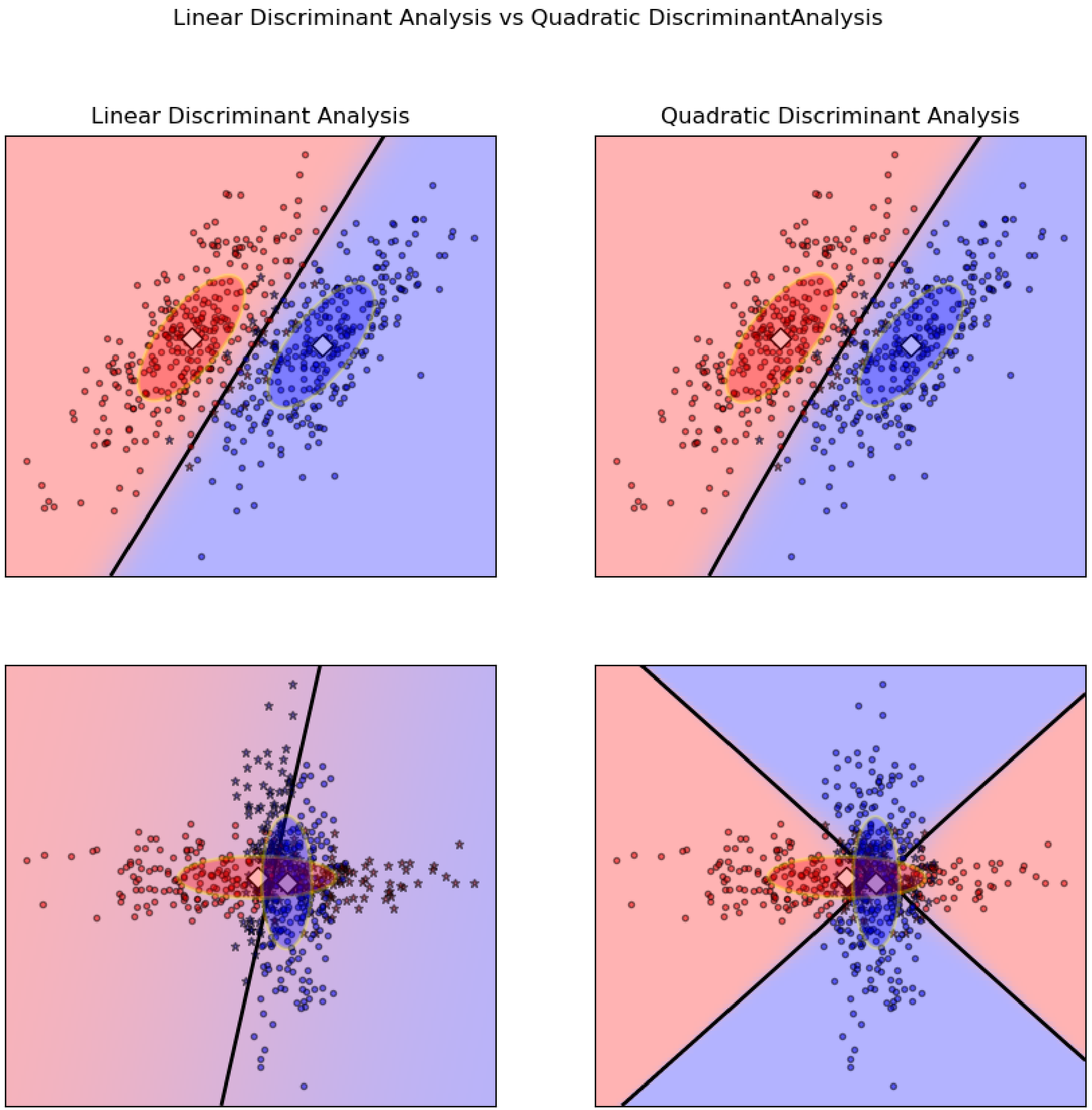


Pcolormesh

Cmap = ‘red_blue_classes’

Contour = 0.5


Linear Discriminant Analysis vs Quadratic DiscriminantAnalysis
Linear Discriminant Analysis
Quadratic Discriminant Analysis


Linear Discriminant Analysis vs Quadratic DiscriminantAnalysis
Linear Discriminant Analysis
Quadratic Discriminant Analysis

使用matplotlib的优势

- 矢量作图，可保存为矢量格式图片
- 绘图自由度高，可定制修改细节丰富
- 与计算端同平台，可与算法深度耦合