```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from scipy.stats import multivariate_normal
```

```python
def fit(x_train, y_train):
    m = y_train.shape[0] # Number of training example
    #Reshapeing the training set
    x_train = x_train.reshape(m, -1)
    input_feature = x_train.shape[1] # Number of input feature. In our c
    class_label = len(np.unique(y_train.reshape(-1))) # Number of class.

    # Start everything with zero first.
    # Mean for each class. Each row contains an individual class. And ea
    mu = np.zeros((class_label, input_feature))
    # Each row will conatain the covariance matrix of each class.
    # The covariance matrix is a square symettric matrix.
    # It indicates how each of the input feature varies with each other.
    sigma = np.zeros((class_label, input_feature, input_feature))
    # Prior probability of each class.
    # Its the measure of knowing the likelihood of any class before seei
    phi = np.zeros(class_label)

    for label in range(class_label):
        # Seperate all the training data for a single class
        indices = (y_train == label)

        phi[label] = float(np.sum(indices)) / m
        mu[label] = np.mean(x_train[indices, :], axis=0)
        # Instead of writting the equation we used numpy covariance func
        sigma[label] = np.cov(x_train[indices, :], rowvar=0)

    return phi, mu, sigma
```

```
def predict(x_tests, phi, mu, sigma):
    # flatten the training data
    x_tests = x_tests.reshape(x_tests.shape[0], -1)
    class_label = mu.shape[0] # Number of label we have in our case it's
    scores = np.zeros((x_tests.shape[0], class_label))  # Initially we s
    for label in range(class_label): # We will calculate the probability
        # normal_distribution_prob.logpdf Will give us the log value of
        normal_distribution_prob = multivariate_normal(mean=mu[label], c
        # x_test can have multiple test data we will calculate the proba
        for i, x_test in enumerate(x_tests):
            scores[i, label] = np.log(phi[label]) + normal_distribution_
    predictions = np.argmax(scores, axis=1)
    return predictions
```

```
data = load_iris()
x_train, x_test, y_train, y_test = train_test_split(data.data, data.targ
phi, mu, sigma = fit(x_train, y_train)
y_predict = predict(x_test, phi, mu, sigma)
score = f1_score(y_test, y_predict, average="weighted")
print("f1 score of our model: ", score)

# Compare this model with scikitlearn LinearDiscriminatorAnalysis
lda = LinearDiscriminantAnalysis()
lda.fit(x_train, y_train)
y_predict_sk = lda.predict(x_test)
print("f1 score of scikit-learn model is: ", f1_score(y_test, y_predict_
```

```
f1 score of our model:  0.9740594802514307
f1 score of scikit-learn model is:  0.9740594802514307
```