Тема 2

Linear Discriminant Analysis (LDA)

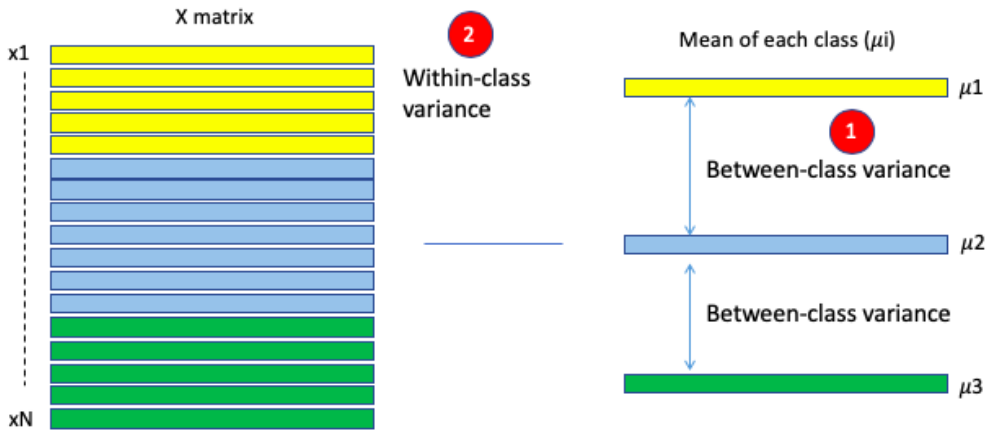
(Линеен дискриминантен анализ)

1. Какво е LDA?

LDA се стреми да запази добра дискриминационна сила на зависимата променлива, като същевременно проектира матрица на оригиналното множество от данни върху пространство с по-малко измерения. LDA е вид метод от контролираното обучение. Класовете от зависимата променлива се използват, за да се раздели пространството на предикторите в региони. Всички региони имат линейни граници. Именно от тук идва и името на този метод. Моделът предсказва, че всички наблюдения в даден регион принадлежат към един и същи клас на зависимата променлива.

Това се постига в три стъпки:

- Изчислява се междукласовата дисперсия (between-class variance) или разделението между различните класове. Това всъщност е разстоянието между средните на отделните класове, които се съдържат в зависимата променлива (Фиг. 1. 1).
- Изчислява се дисперсията в рамките на класа (within-class variance). Това е разстоянието между средното и извадката от всеки клас (Фиг. 1. 2).
- В третата стъпка се максимизира междукласовата дисперсия от първа стъпка и се минимизира дисперсията в рамките на класа от втора стъпка в пространство с по-малко измерения. Тази операция се нарича критерий на Фишер (F-критерий).



Фигура 1. LDA

2. Пример 1. Логистична регресия и LDA

В този пример ще приложим LDA върху елементарен модел с логистична регресия, както направихме в предходната лекция с PCA. Ще си генерираме множество от данни, състоящо се от 1000 наблюдения и 20 характеристики.

Започваме като заредим необходимите библиотеки. Ще използваме numpy и scikit-learn.

```
# evaluate lda with logistic regression algorithm for classification
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
```

След това е необходимо да дефинираме самото множество от данни. Това става чрез функцията make_classification, която се записва в обектите X и y. С n_samples определяме броя наблюдения (в случая 1000). С n_features определяме броя на променливите, а с n_informative и n_redundant – съответно броя информативни и излишни характеристики.

```
X, y = make_classification(n_samples=1000, n_features=20, n_informative=10, n_redundant=10, random_state=7)
```

Дефинираме pipeline, в който включваме метода LDA и модела – логистична регресия. С аргумента n_components обозначаваме броя на желаните измерения в изходните данни (резултати, output) след приложените трансформации.

```
steps = [('lda', LinearDiscriminantAnalysis(n_components=1)), ('m', LogisticRegression())]
```

model = Pipeline(steps=steps)

Следва да оценим модела. Това става като първо приложим крос-валидация чрез KFold. За нас от значение е само точността на модела (accuracy). В n_scores ще се запишат всички изчислени скорове (точности), като самото изчисление се осъществява с помощта на функцията cross_val_scores.

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)

В случая ще вземем предвид средното и стандартното отклонение от всички изчислени точности чрез функциите mean и std.

print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))

Резултатът, който се отпечатва в конзолата е: 0.825 (0.034).

3. Пример 2. Директно прилагане на LDA върху данни

В този пример ще приложим върху вече познатото ни множеството от данни метода LDA и ще проверим дали това ще доведе до промяна в точността на модела. Ще сравним резултатите от двата примера и по този начин ще определим кой има по-висока предсказваща сила.

Започваме със зареждането на библиотеките. Отново ще използваме numpy и scikit-learn.

from numpy import mean

from numpy import std

from sklearn.datasets import make_classification

from sklearn.model_selection import cross_val_score

from sklearn.model_selection import RepeatedStratifiedKFold

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

Както в предходния пример, така и тук ще дефинираме множество от случайни данни, като наблюденията ще бъдат 1000, а променливите – 10, всяка от които е информативна.

X, y = make_classification(n_samples=1000, n_features=10, n_informative=10, n_redundant=0, random_state=1)

Следващата стъпка е да дефинираме модел. В случая моделът, който ще използваме е LDA, който се дефинира чрез функцията LinearDiscriminantAnalysis() и се съхранява в обекта model.

model = LinearDiscriminantAnalysis()

Определяме метод, по който ще оценим модела, а именно чрез крос-валидацията RepeatedStratifiedKFold. Отново ще имаме 10 подмножества, и операцията ще се изпълни 3 пъти.

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

Следващата стъпка е да оценим самия модел. Това ще стане като изчислим точността (accuracy) чрез функцията cross_val_score, която приема за аргументи модела, множеството от данни, типа оценка (точност), крос-валидацията. Използваме n_jobs=-1, за да използваме всички процесори при изчисленията.

scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)

Извеждаме резултата в конзолата, като вземем предвид средната точност и стандартното отклонение.

print('Mean Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))

Точността е 0.893, а ст. отклонение 0.033.

Сравнявайки резултати от двата модела, виждаме, че моделът от пример две има по-добра производителност, като при него не се използва логистична регресия.

4. Пример 3. Прогнозиране с LDA модел

Зареждаме необходимите модули от scikit-learn.

from sklearn.datasets import make_classification

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

Дефинираме множеството от данни със същите параметри, както в пример 2.

X, y = make_classification(n_samples=1000, n_features=10, n_informative=10, n_redundant=0, random_state=1)

Определяме какъв да бъде вида на модела – в случая LDA.

```
model = LinearDiscriminantAnalysis()
```

Този път пропускаме крос-валидацията, но ще тренираме модела, като изпълним функцията fit() върху обекта model. В тази си форма, функцията съдържа единствено множеството от данни.

```
model.fit(X, y)
```

Дефинираме обект row, в който се съдържа едно наблюдение с по една стойност за всяка характеристика. Целта е да предскажем към кой клас спада това наблюдение.

```
row = [0.12777556,-3.64400522,-2.23268854,-1.82114386,1.75466361,0.1243966,1.03397657,2.35822076,1.01001752,0.56768485]
```

Извършваме прогнозирането като в yhat прилагаме функцията predict() върху модела по отношение на наблюдението row, което можем да приемем за аргумент на функцията.

```
yhat = model.predict([row])
```

С този ред отпечатваме в конзолата към кой клас спада наблюдението row. В конзолата се изписва Predicted Class: 1, което означава, че row спада към клас 1. Когато работим с реално множество от данни, всеки един клас ще има смислено значение.

```
print('Predicted Class: %d' % yhat)
```

5. Пример 4. Сравнение между PCA и LDA, чрез прилагането им върху реално множество от данни

В този пример ще разгледаме как би повлияло приложението на LDA и PCA върху множеството от данни, съдържащо в себе си информация за проби от различни червени вина.

Зареждаме необходимите библиотеки. Това са matplotlib, scikit-learn, pandas и numpy.

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt
```

from sklearn.decomposition import PCA

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

Зареждаме множеството от данни чрез функцията read_csv. Множеството се запазва в обекта wine.

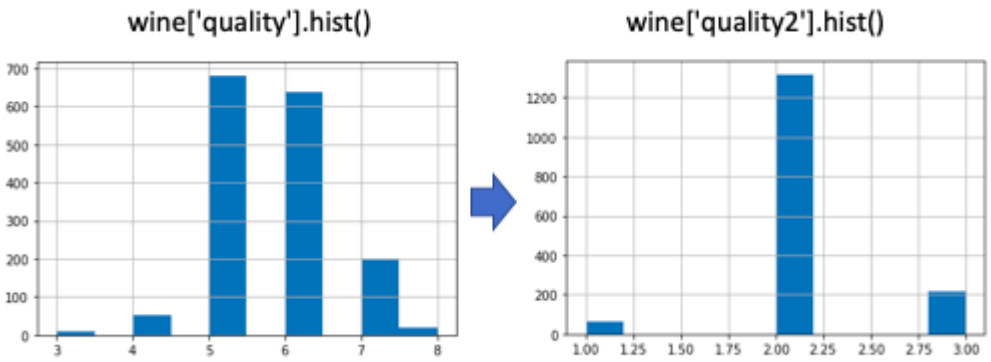
wine = pd.read_csv(‘winequality-red.csv’)

Прилагайки функцията head върху множеството от данни, в конзолата ще се отпечатат първите няколко реда от данните.

wine.head()

Необходимо е да трансформираме зависимата променлива quality, тъй като съдържа в себе си много стойности. Ще ги групираме в три класа. Проби с оценка до 4 се класифицират като 1, проби с оценка между 5 и 6 – като 2, проби с оценки над 7 – като 3.

wine[‘quality2’] = np.where(wine[‘quality’]<=4,1, np.where(wine[‘quality’]<=6,2,3)).



Фигура 2. Преди и след трансформирането на целевата променлива.

Дефинираме с X независимите променливи, като не взимаме предвид последните две целеви променливи (оригиналната и трансформираната), а в y записваме трансформираната целева променлива.

X = wine.drop(columns=[‘quality’,’quality2'])

y = wine[‘quality2’]

В target_names записваме възможните стойности, наблюдавани в y, т.е. в Ще се съдържат стойности 1, 2 и 3.

```
target_names = np.unique(y)

target_names
```

Прилагаме PCA и LDA с по два компонента върху множеството с независимите променливи X (при PCA) и върху X и y (при LDA).

```
pca = PCA(n_components=2)

X_r = pca.fit(X).transform(X)

lda = LinearDiscriminantAnalysis(n_components=2)

X_r2 = lda.fit(X, y).transform(X)
```

С четирите реда код отдолу принтираме в конзолата дисперсията на отделните компоненти на PCA и LDA.

```
print('explained variance ratio (first two components of PCA): %s'

 % str(pca.explained_variance_ratio_))

print('explained variance ratio (first two components of LDA): %s'

 % str(lda.explained_variance_ratio_))
```

В конзолата се отпечатва следното:

```
explained variance ratio (first two components of PCA): [0.99448915 0.00522862]

explained variance ratio (first two components of LDA): [0.83262622 0.16737378]
```

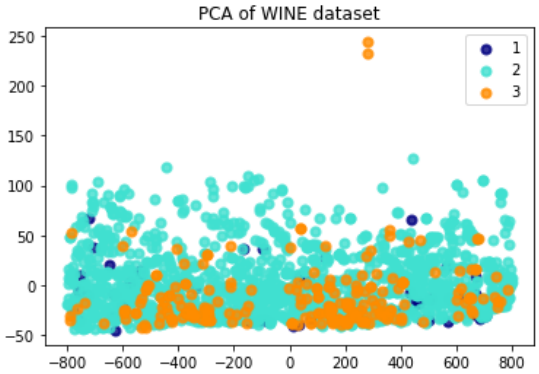
Тези резултати показват, че дисперсията на първия компонент при PCA е по-висока, отколкото на LDA. Сравнявайки резултатите, виждаме, че и при двата метода първите компоненти са най-значими.

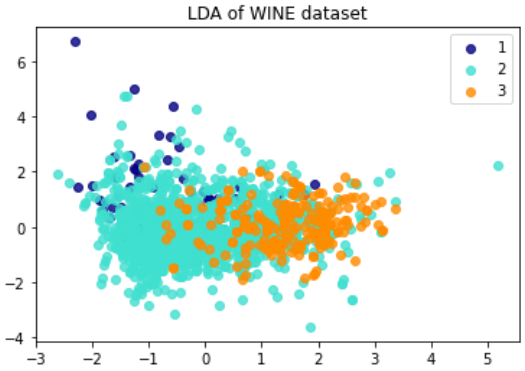
За да сравним PCA и LDA, трябва да визуализираме трансформираните множества, като всъщност наблюденията ще бъдат разпределени според класа, към който принадлежат. Всеки от трите класа е визуализиран с различен цвят, за можем по-лесно да ги различим.

На Фигура 3 ясно се вижда, че наблюденията са по-равномерно разпределени, в случая при прилагане на PCA, докато след приложението на LDA се наблюдава концентрирано струпване на наблюденията. Оптималният случай е наблюденията от

трите класа да са напълно разграничени и отделени в три отделни клъстера. Това не се наблюдава в този случай.

```
plt.figure()
colors = [‘navy’, ‘turquoise’, ‘darkorange’]
lw = 2
for color, i, target_name in zip(colors, target_names, target_names):
 plt.scatter(X_r[y == i, 0], X_r[y == i, 1], color=color, alpha=.8, lw=lw,
 label=target_name)
plt.legend(loc=’best’, shadow=False, scatterpoints=1)
plt.title(‘PCA of WINE dataset’)
plt.figure()
for color, i, target_name in zip(colors, target_names, target_names):
 plt.scatter(X_r2[y == i, 0], X_r2[y == i, 1], alpha=.8, color=color,
 label=target_name)
plt.legend(loc=’best’, shadow=False, scatterpoints=1)
plt.title(‘LDA of WINE dataset’)
plt.show()
```





Фигура 3. Резултати от приложението на PCA и LDA върху множеството от данни.

6. Пример 5. LDA класификационен модел

В този пример ще разгледаме множеството Iris и ще приложим LDA върху него.

Зареждаме необходимите библиотеки. Ще използваме scikit-learn, matplotlib, numpy и pandas.

```
from sklearn.model_selection import train_test_split

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

from sklearn.model_selection import RepeatedStratifiedKFold

from sklearn.model_selection import cross_val_score

from sklearn import datasets

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np
```

Зареждаме множеството Iris от scikit-learn библиотеката. Във Variable explorer се зарежда обекта Iris, който съдържа 8 елемента – описание на множеството, стойностите на независимите променливи, имената на тези променливи, стойностите на целевата променлива и имената им. Дефинираме дейтафрейм df, в който ще съхраним данните – независимите променливи и целевата променлива чрез функцията c_, която всъщност копира избраните от нас елементи от Iris и ги поставя в df.

```
iris = datasets.load_iris()

df = pd.DataFrame(data = np.c_[iris['data'], iris['target']],
```

```
columns = iris['feature_names'] + ['target'])
```

След това трансформираме целевата променлива в категорийна, като вместо стойности 0, 1 и 2, вече ще имаме setosa, versicolor и virginica. Това се прави след колоната target, която съдържа класовете под формата на числови стойности.

```
df['species'] = pd.Categorical.from_codes(iris.target, iris.target_names)
```

След това заменяме дългите има на характеристиките с по-къси.

```
df.columns = ['s_length', 's_width', 'p_length', 'p_width', 'target', 'species']
```

Със следващите два реда принтираме първите няколко наблюдения от множеството в конзолата (head()), както и с колко наблюдения разполагаме (len()).

```
print(df.head())
```

```
len(df.index)
```

С X дефинираме кои ще бъдат независимите променливи, а с y целевата променлива.

```
X = df[['s_length', 's_width', 'p_length', 'p_width']]
```

```
y = df['species']
```

Дефинираме модела, който искаме да използваме – LDA – и го запазваме в обекта model. След това фитваме (тренираме) модела.

```
model = LinearDiscriminantAnalysis()
```

```
model.fit(X, y)
```

Прилагаме и крос-валидацията RepeatedStratifiedKFold, като ще разделим множеството на 10 части и ще повторим операциите 3 пъти.

```
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
```

Следващата стъпка е да изберем показател за оценяване на модела. Избираме accuracy. След изпълнението на следващите два реда код, в конзолата се принтира точността на модела, а тя е 98%.

```
scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
```

```
print(np.mean(scores))
```

Дефинираме ново наблюдение new, със стойности за всяка характеристика и след това чрез функцията predict() изчисляваме към кой клас ще спадне това наблюдение. В конзолата се отпечатва, че наблюдението принадлежи на клас setosa.

```
new = [5, 3, 1, .4]

model.predict([new])
```

Последната стъпка е да визуализираме линейните дискриминанти на модела, за да видим колко добре LDA е разделил разграничил трите класа един от друг.

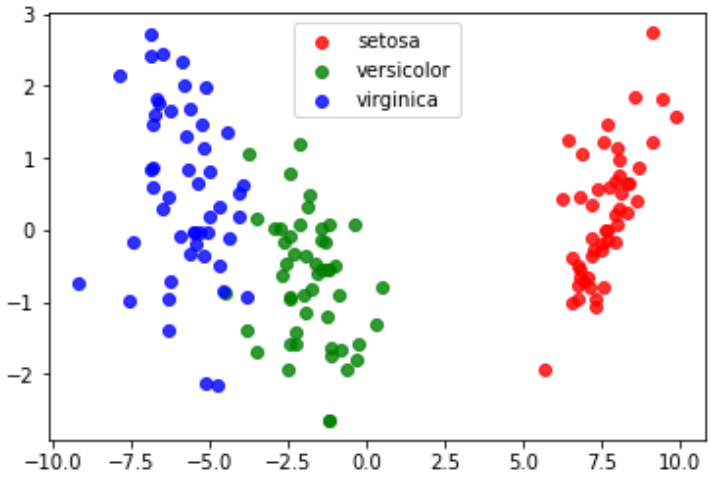
Дефинираме данните, чрез които ще бъде изградена графиката.

```
X = iris.data

y = iris.target

model = LinearDiscriminantAnalysis()

X_r2 = model.fit(X, y).transform(X)

target_names = iris.target_names
```

Дефинираме самата графика.

```
plt.figure()

colors = ['red', 'green', 'blue']

lw = 2

for color, i, target_name in zip(colors, [0, 1, 2], target_names):

    plt.scatter(X_r2[y == i, 0], X_r2[y == i, 1], alpha=.8, color=color,

                label=target_name)

plt.legend(loc='best', shadow=False, scatterpoints=1)

plt.show()
```

На графиката по-долу виждаме, че най-ясно разграничен е класът setosa. Versicolor и virginica също сравнително добре отделени един от друг. (Направете съпоставка с резултатите от предходния пример.)



Фигура 4. Разграничаване на класовете чрез LDA.

Препоръчителна литература:

1. Dimensionality Reduction in Data Science, Garzon, M., **Chapter 2, Chapter 4 (p. 90 - 95)**
2. CRISP-DM Process Model Microsoft Word - D2-3-1 crisp-dm_sig.doc (keithmccormick.com) **(ЗАДЪЛЖИТЕЛНО)**
3. Linear Discriminant Analysis | What is Linear Discriminant Analysis (analyticsvidhya.com)