ИППИ РАН

Дискриминантный анализ Фишера с kernel trick

Каплоухая Нина

(Все материалы проекта доступны по ссылке)

Содержание I

1. FDA
   - Принцип работы FDA
2. KFDA
   - Переход в расширенное пространство
   - Kernel trick
   - Основной код для реализации KFDA
   - KFDA на Toy Dataset
   - Параметры ядра
   - KFDA на Iris Dataset
3. Анализ работы KFDA
   - ROC кривые
   - Accuracy. Сравнительная таблица
4. Применение
5. Распознавание лиц
   - Fisherfaces
   - Сравнение проекций FDA/PCA
   - Сравнение проекций KFDA/KPCA
   - Анализ точности методов

Содержание II

6 Список литературы

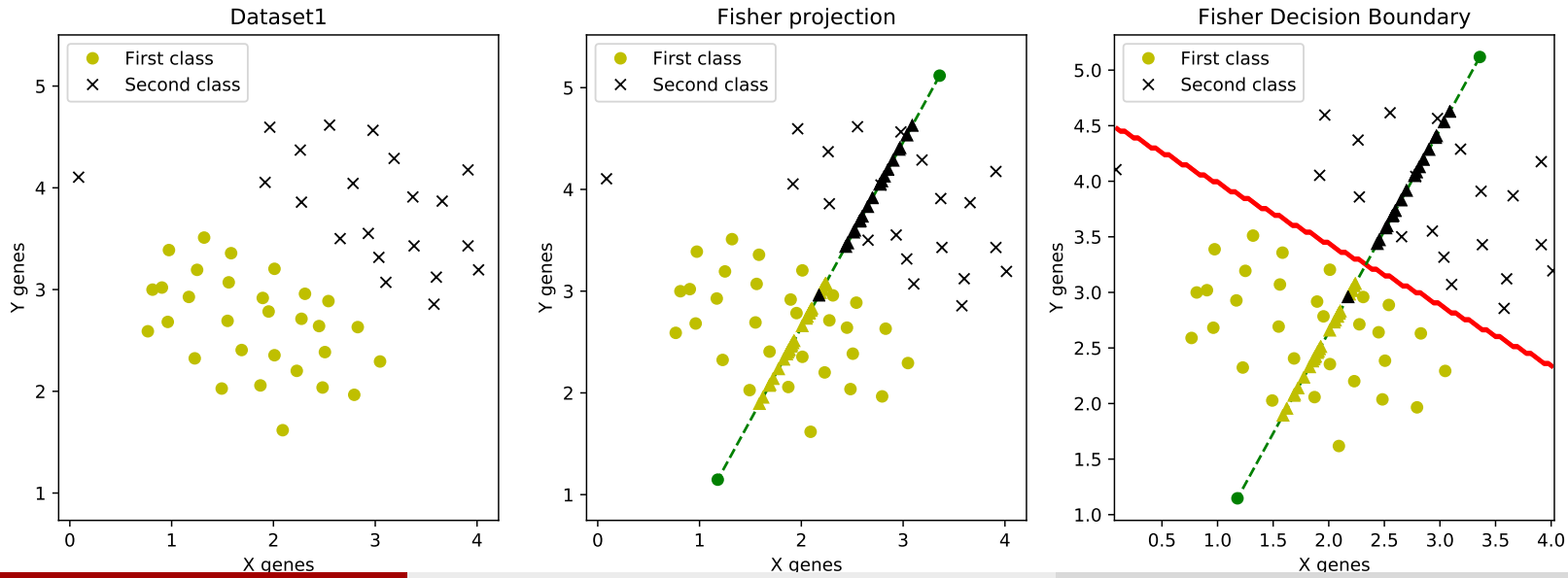
7 Теоретическая справка
- FDA. Вывод алгоритма
- Алгоритм FDA
- Пример работы алгоритма
- KFDA. Вывод алгоритмя
- Виды ядер
- Переход от FDA к KFDA
- KFDA для многоклассовой задачи
- Другие методы DA. LDA и QDA. Общие принципы

Принцип работы *FDA*

Линейный дискриминантный анализ и линейный дискриминант Фишера(LDA и FDA)

Метод статистики и машинного обучения, применяемый для нахождения линейных комбинаций признаков, наилучшим образом разделяющих два или более класса объектов или событий. Полученная комбинация может быть использована в качестве линейного классификатора или для сокращения размерности пространства признаков перед последующей классификацией.

Пример работы *FDA* (подробный пример см. здесь):



Принцип работы *FDA*.
Как найти наилучший вектор для проекции данных?

Задача: Найти вектор ω, в проекции на который максимально отношение матрицы межклассового распределения S_b' к матрице внутриклассового S_w':

$$I = \frac{(\mu_1' - \mu_2')^2}{s_1^2 + s_2^2} = \frac{S_b'}{S_w'}$$

$$\omega = \underset{\omega}{\operatorname{argmax}} I = \underset{\omega}{\operatorname{argmax}} I(\omega) = \underset{\omega}{\operatorname{argmax}} \frac{\omega S_b \omega^T}{\omega S_w \omega^T}$$

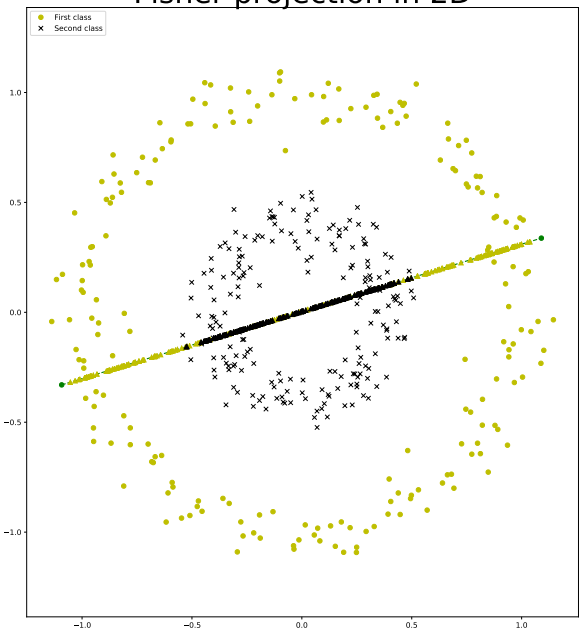
Продифференцировав по ω (подробнее вывод см. здесь), приходим к ур-ию:

$$S_w^{-1} S_b \omega = I\omega$$

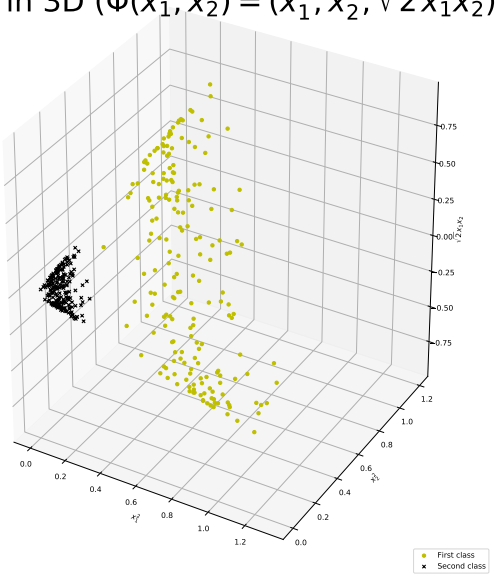
Откуда, искомый вектор ω - собственный вектор матрицы $S_w^{-1} S_b$,

Переход в пространство более высокой размерности. Пример:

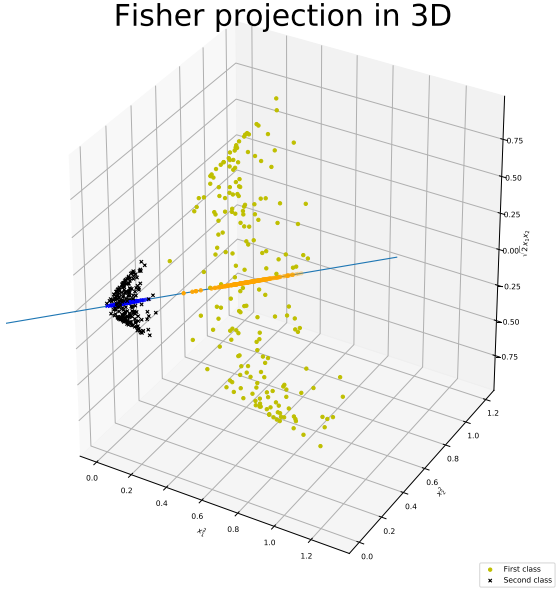
Fisher projection in 2D



Data in 3D ($\Phi(x_1, x_2) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$)



Fisher projection in 3D

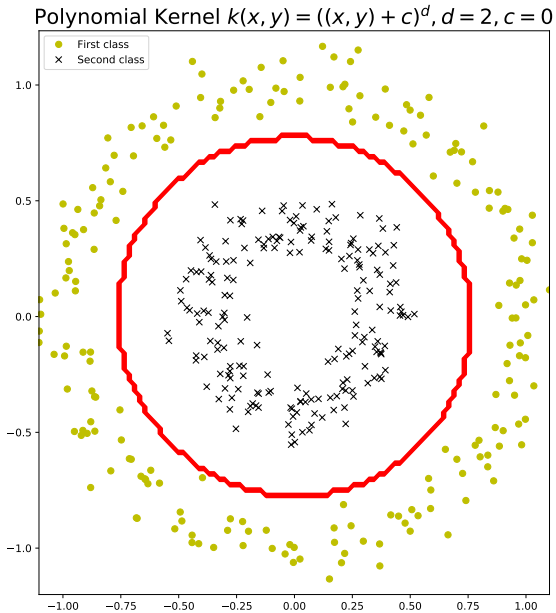


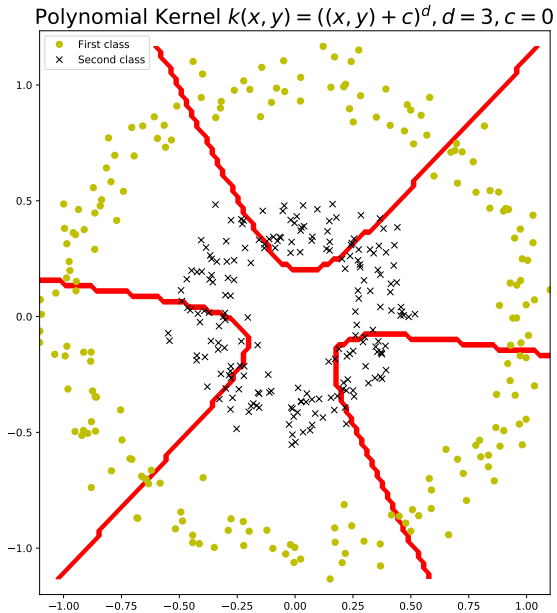
Kernel Trick.[1] KFDA

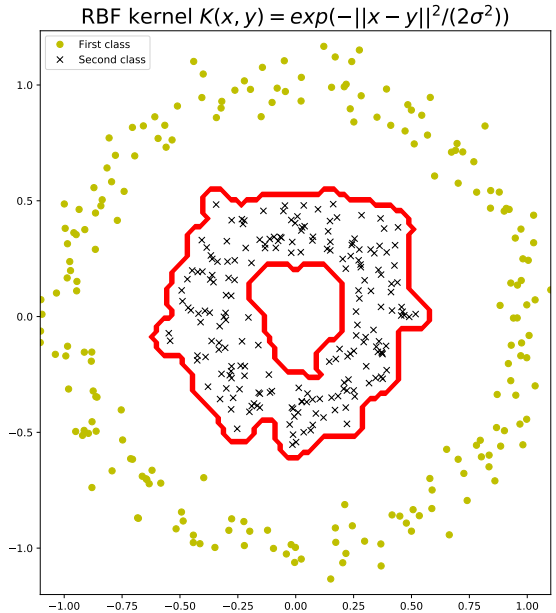
Kernel trick

$$k(x_i, x_j) = (\Phi(x_i), \Phi(x_j)), \text{ где } \Phi : R^n \longrightarrow R^m.$$

Пример работы классификатора *KFDA* на *CircleData* с различными ядрами:







[1] Подробнее про ядра см. здесь.

Основной код для реализации KFDA[2]

```
from sklearn.base import BaseEstimator, ClassifierMixin
class KFDACl(BaseEstimator, ClassifierMixin): #Класс, для реализации KFDA
    def __init__(self, kernel = 'RBF', gamma = 'auto', c= None, d = None, Alg = 'SVM'):
        """
       parameters:
       kernel - string, тип ядра: linear, Polynomial, RBF
       gamma - float, параметр RBF kernel.
       c, d - int,  параметры Polynomial kernel
        (свободный член и степень соответсвенно)
       Alg - string, способ классификации после понижения размерности: KNN, SVM
        """
        self.kernel = kernel
        self.gamma = gamma
        self.c = c
        self.d = d
        self.Alg = Alg
```

[2]Подробный вывод алгоритма *KFDA* см. здесь.

KFDACl.fit I

```
def fit(self, X, y):
        """
        input:
           X - np.ndarray, матрица признаков объектов  nxk
           y - np.array, вектор меток классов nx1
        """
        if (self.gamma =='auto'):
            self.gamma = 1/(X.shape[1])
        self.labels_ = np.unique(y) # находим значение меток классов
        self.Cl_ = len(self.labels_) #Кол-во классов
        self.cl_ = [np.nonzero(y == self.labels_[i])[0] for i in range(0,self.Cl_)]
        Cl =self.Cl_
        K = self.Kernelmtrx(X, X)
        #Разделяем образцы по классам
        Xcl = [X[self.cl_[i],:] for i in range(0,Cl)]
        #Кол-во образцов в каждом из классов
        ni = [Xcl[i].shape[0] for i in range (0,Cl)]
        n= sum(ni)
        #Вычисляем kernel matrix
        Ki = [K[:,np.nonzero(y == self.labels_[i])[0]] for i in range(0,Cl)]
        #Вычисляем Mi
        Mi = [np.mean(Ki[i], axis= 1) for i in range (0,Cl)]
        M0 = np.array([np.mean(K, axis=1)])
        M = np.zeros((n,n))
        for i in range (0,Cl):
            M = M + np.dot((M0-Mi[i]).T, M0-Mi[i])
        #Вычисляем Ni
        I = [np.eye(ni[i]) for i in range (0,Cl)]
        O = [1/float(ni[i]) for i in range (0,Cl)]
        T = [(I[i] - O[i]) for i in range (0, Cl)]
```

KFDACl.fit II

```
Ni = [np.dot(Ki[i], np.dot(T[i], Ki[i].T)) for i in range (0, Cl)]
#Добавляем ти (для регуляризации)
eps=np.eye((n))
eps = eps * 0.001
N= sum(Ni) + eps
e, v = np.linalg.eigh(M) #находим собственные значения и вектора M
eps = np.ones(len(e))*abs(min(e))
e = e + 2*eps
sqrtE = [math.sqrt(x) for x in e]
sqrtM =  np.dot(np.dot(v,np.diag(sqrtE)), np.matrix.transpose(v))
S = np.dot(sqrtM, np.dot(np.linalg.inv(N), sqrtM))
eigenValues, eigenVectors = scipy.linalg.eigh(S)
self.alpha_ = np.zeros((eigenVectors.shape[0], Cl-1)) #alpha, вектор/матрица для проекции данных
idx = eigenValues.argsort()[::-1]
eigenValues = eigenValues[idx]
eigenVectors = eigenVectors[:,idx]
#alpha матрица для проекции, сост. из Cl-1 собств. векторов, соотв. Cl-1 наиб. собств. значениям S:
for i in range (0, Cl):
    self.alpha_[:,i-1] =  np.dot(np.linalg.inv(sqrtM),eigenVectors[:, Cl-i-1])
Z=np.dot(K,self.alpha_)
# Значение образцов в проекции в Cl-1 пространство:
self.X_red_ = Z.reshape(Z.shape[0],(Cl-1))
self.Xtr_= X
```

KFDACl.fit (продолжение), KFDACl.predict

```
#Продолжение fit():
    # После понижения размерности, для спроектированных данных строим предсказательную модель:
    if (self.Alg == 'KNN'):

            PrModel = KNeighborsClassifier(n_neighbors=5)
            PrModel.fit(self.X_red_, y)
            self.PrModel_=PrModel
            print (PrModel.score(self.X_red_,y))

    if (self.Alg == 'SVM'):
        if (self.Cl_ == 2):
            PrModel= SVC(kernel = 'linear', probability =True)
        else:
            PrModel= SVC(probability =True)
        PrModel.fit(self.X_red_, y)
        self.PrModel_=PrModel
    return self

def predict(self,Xtst):
"""
input:
    Xtst - np.ndarray, матрица признаков объектов, для которых необх предсказать класс, mxk
output:
   ypred - вектор предсказанный меток классов для объектов из Xpr, mx1
"""
    Cl = self.Cl_
    Kpr = self.Kernelmtrx(self.Xtr_, Xtst)
    Xtst_red = np.dot(Kpr, self.alpha_) #Проектируем данные в уже найденное (Cl-1) мерное подпр-во
    Xtst_red = Xtst_red.reshape(Xtst_red.shape[0],(Cl-1))
    ypred = np.zeros((Xtst_red.shape[0],1))
    #Классифицируем спроектир. данные в (Cl-1)-мерном пр-ве, пользуясь уже построенной предс. моделью:
    y_pred = self.PrModel_.predict(Xtst_red)
    return y_pred
```

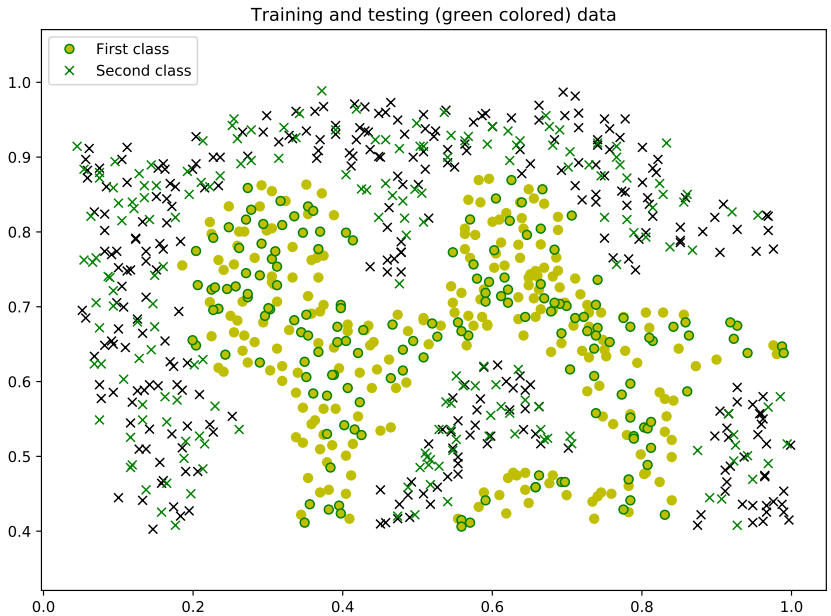
KFDACl.Kernelmtrx

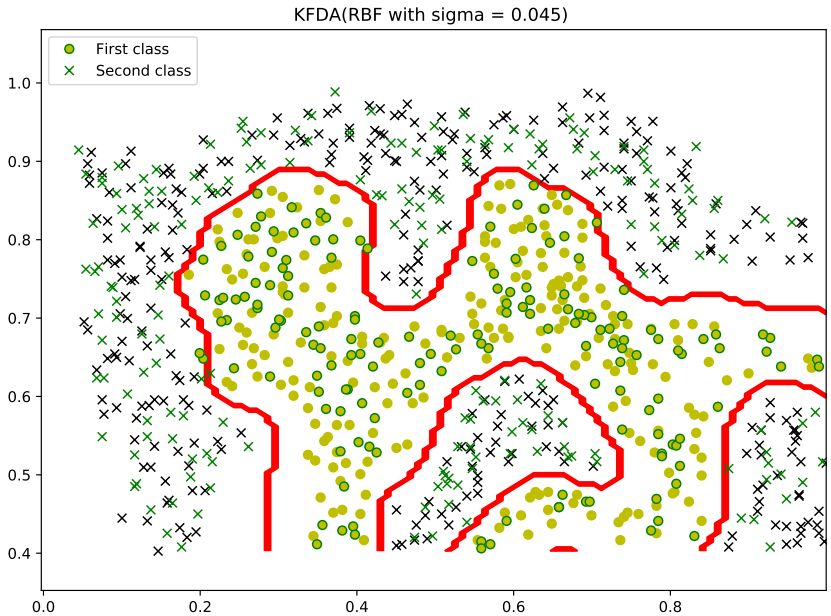
```
def Kernelmtrx(self, Xtr, Xtst):
    """
    input:
        Xtr - матрица  1ой группы объектов nxk
        Xtst - матрица 2-ой группы объектов mxk
    output:
        K - kernel матрица для признаковых векторов двух данных групп объектов, mxn
    """
    K = np.zeros((Xtst.shape[0], Xtr.shape[0]))
    if (self.kernel == 'linear'):   #В случае linear kernel KFDA будет равносильно FDA
        return np.dot(Xtst, Xtr.T)
    if (self.kernel == 'RBF'):   #RBF kernel
        D = scipy.spatial.distance.cdist(Xtst, Xtr, 'sqeuclidean')
        K=exp(-0.5*D/(self.gamma*self.gamma))
        return K
    if (self.kernel =='Polynomial'): #K(x,y) = ((x, y.T) + b)^d
            if (self.c == None):
                print ("Введите свободный параметр c Polynomial Kernel")
                self.c = int(input(),10)
            if (self.d == None):
                print ("Введите степень d Polynomial Kernel")
                self.d = int(input(),10)
            K = (np.dot(Xtst, Xtr.T)+ self.c)**(self.d)
            return K
```

KFDACl.Transform

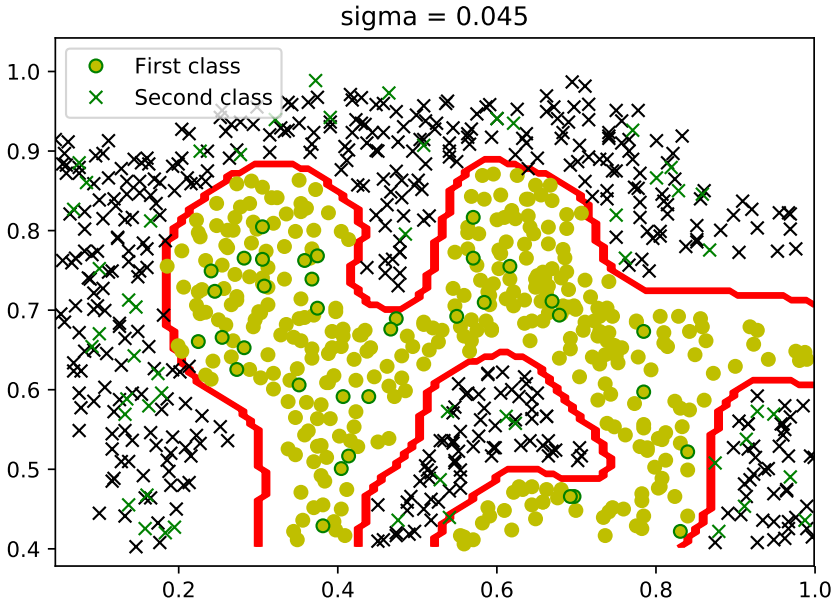
```
def transform(self, Xtst , Proj):
"""
input: Xtst - np.ndarray, матрица признаков объектов, для которых необх. предсказать класс, mxk
       Proj - int, размерность подпространства, в которое проецируем: (Cl-1)<=Proj <=k
 output: Xtst_red - np.darray, матрица данных Xtst, спроецированных  в подпространство из Proj "наилучших"
          найденных KFDA(fit), mxProj
"""
    Cl = self.Cl_
    Kpr = self.Kernelmtrx(self.Xtr_, Xtst)
    w = self.alpha_[:,:Proj]
    Xtst_red = np.dot(Kpr, w)
    Xtst_red = Xtst_red.reshape(Xtst_red.shape[0],Proj)
    return (Xtst_red)


def score(self, X, y, k=5, t = 1, gamma = 0):
    """
input: Xtst - np.ndarray, матрица тестовых объектов, mxk
y - истинные значения меток тестовых объектов
output: accuracy_score - float, ср. точность, с которой классификатор предсказывает метки классов

    """
    return accuracy_score(y, self.predict(X))
```

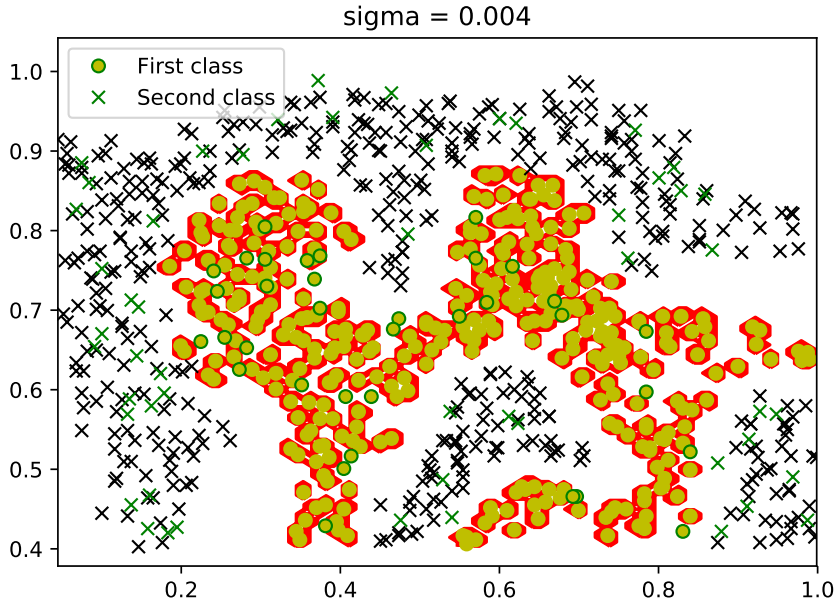
KFDA. Пример работы алгоритма на Toy Dataset:

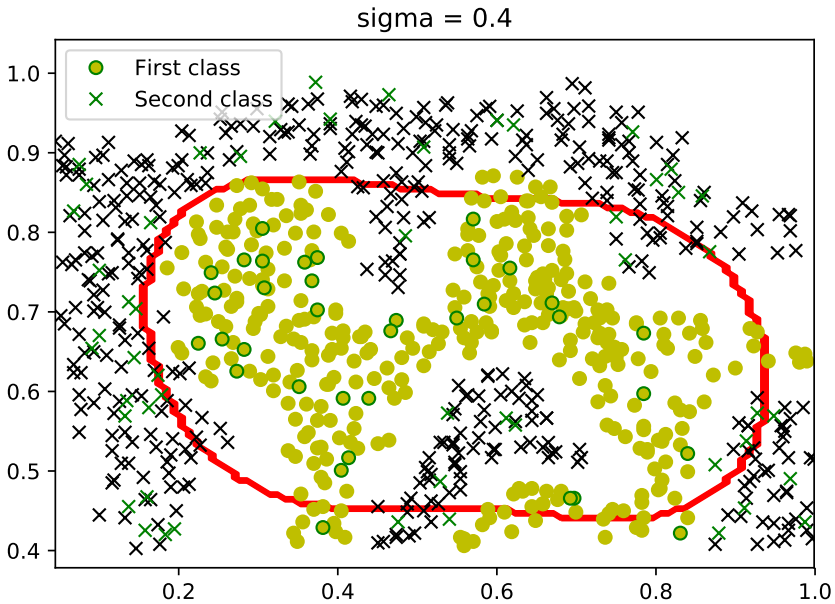


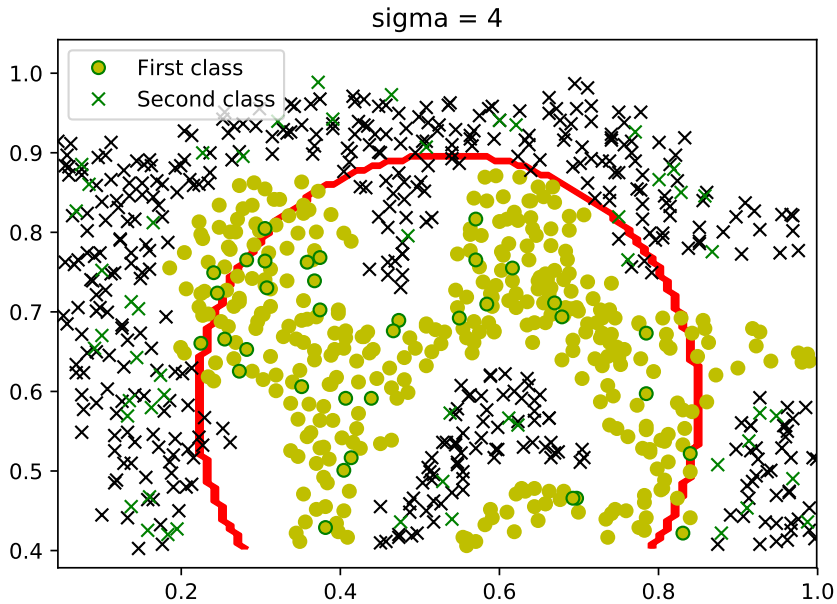


Зависимость *KFDA* с *RBF* kernel от σ (для Toy Dataset)

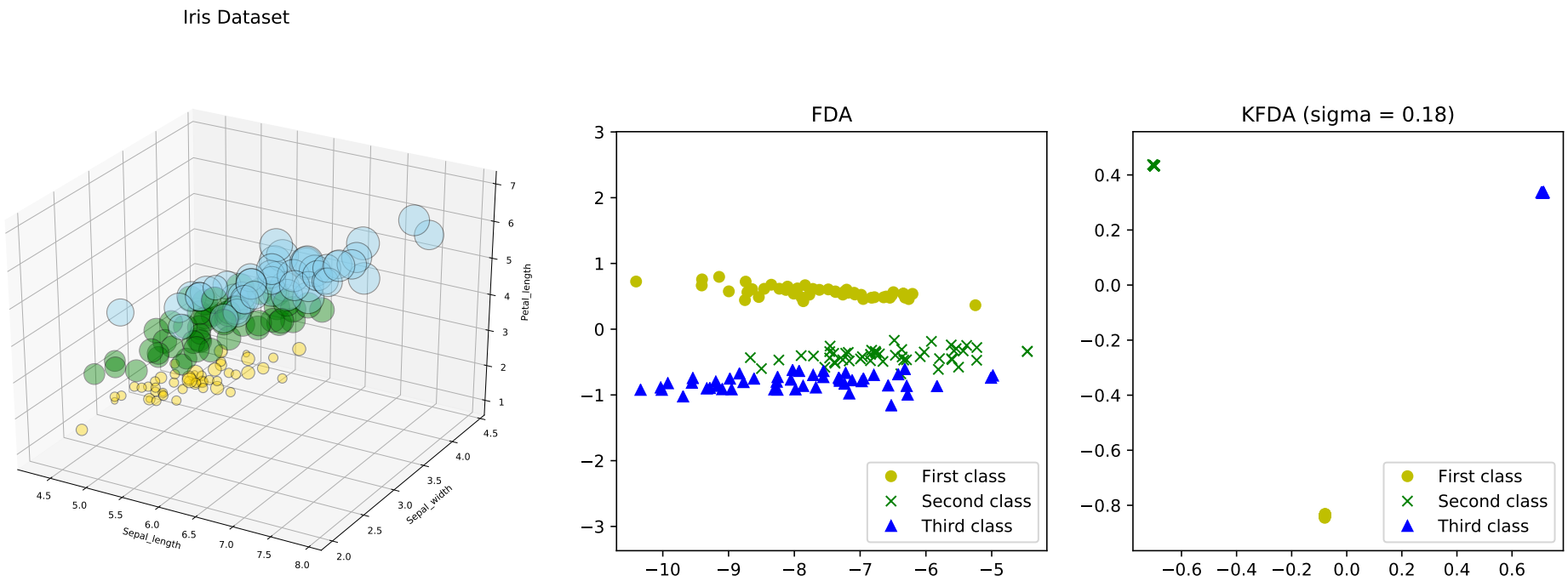






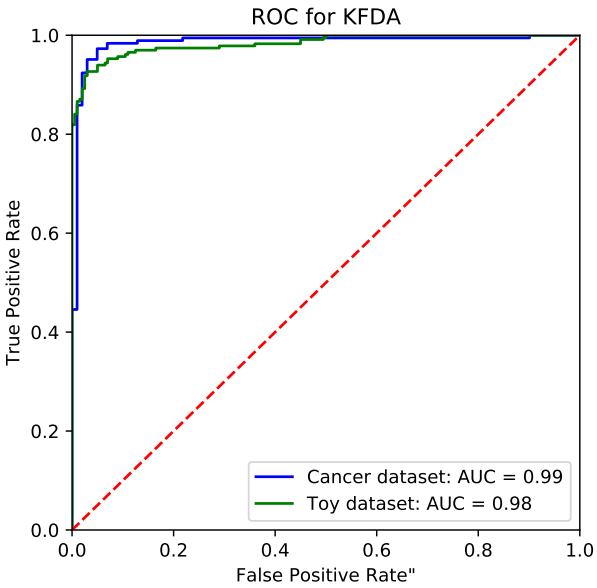


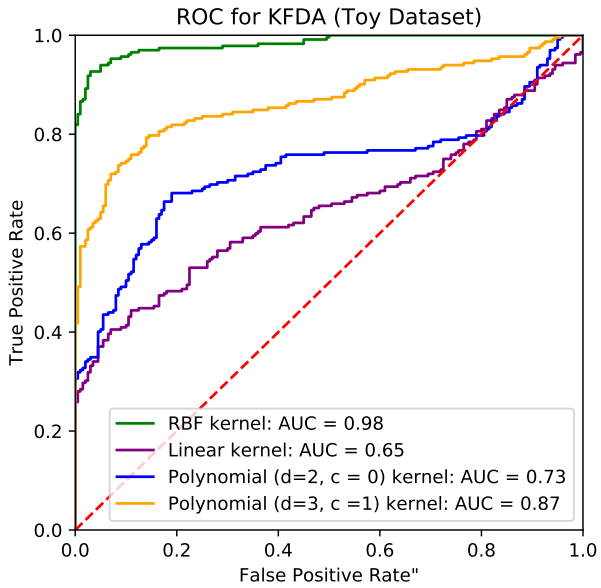
KFDA для многоклассовый задачи. Пример работы:

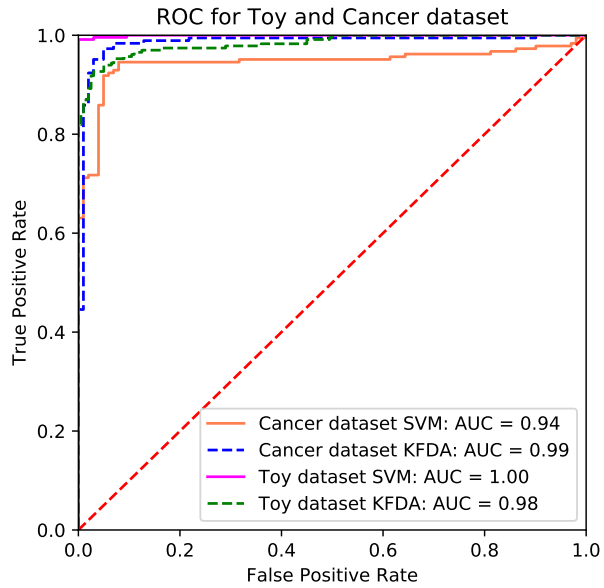


FDA и KFDA на Iris dataset

ROC кривые для *KFDA* на бинарных данных.

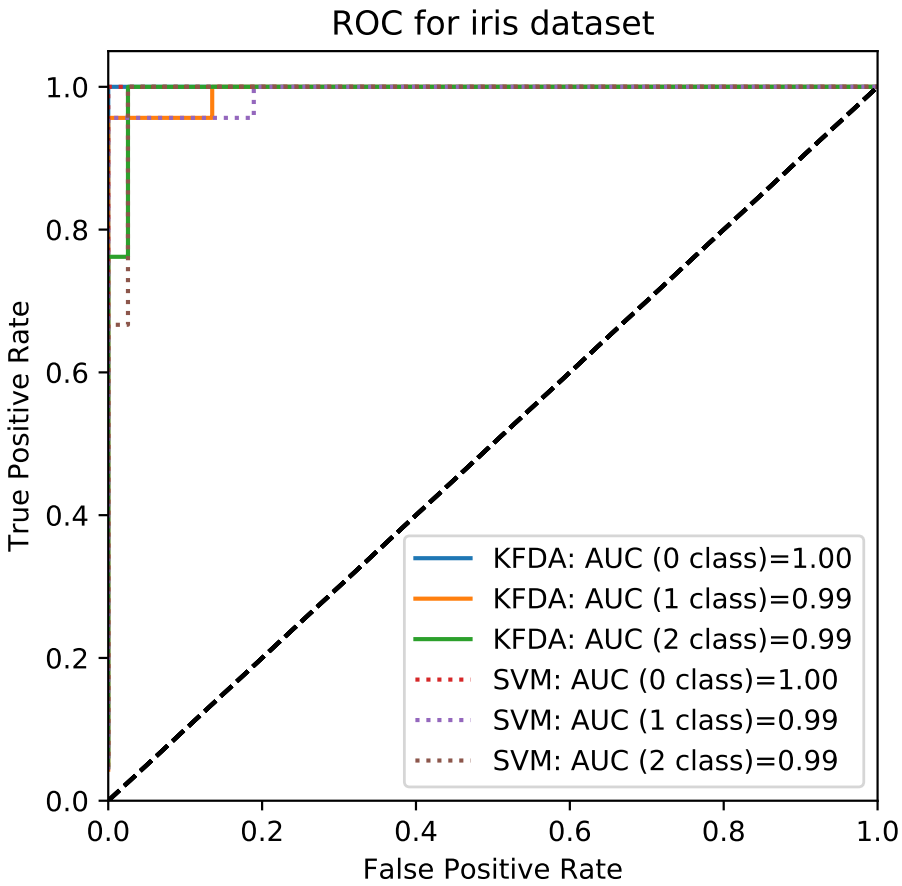


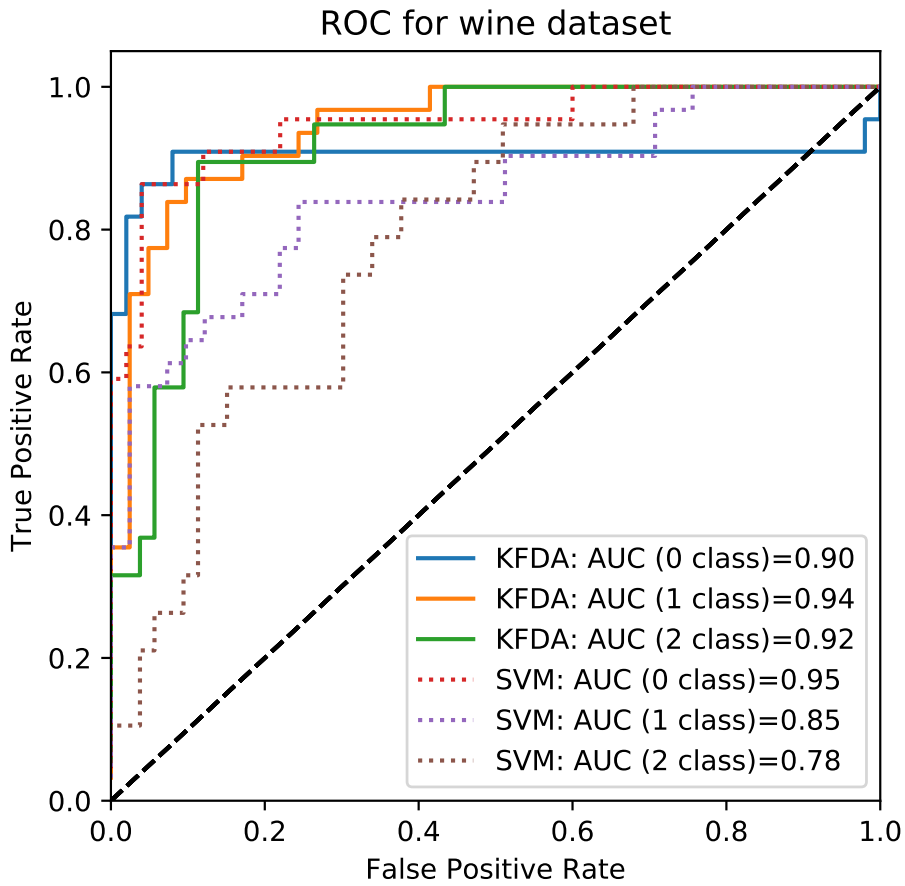




а)Сравнение работы KFDA (RBF) на Cancer и Toy Dataset; б) Сравнение KFDA с разными ядрами на Toy Dataset; в) Сравнение работы KFDA (RBF) и SVM (RBF).

ROC кривые для *KFDA* и *SVM*:





Сравнительная таблица точности классификаторов:

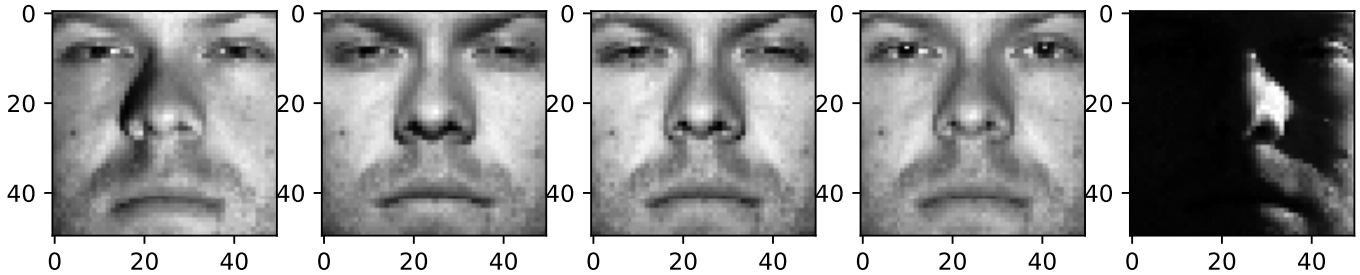
DATASETS	KFDA (RBF)	SVM (RBF)	FDA	KNN (k=5)	LDA	QDA
Toy	91% (+/- 8.5%)	85% (+/- 15.1%)	45% (+/- 14.3%)	84% (+/- 14.8%)	45% (+/- 15.6%)	63% (+/- 21.2%)
Breast Cancer	95% (+/- 0.8%)	91% (+/- 1.3%)	94% (+/- 2.0%)	93% (+/- 2.1%)	96% (+/- 0.9%)	96% (+/- 1.3%)
Iris	97% (+/- 2.1%)	97% (+/- 3.7%)	93% (+/- 3.0%)	97% (+/- 2.5%)	98% (+/- 2.7%)	98% (+/- 2.7%)
Wine	82% (+/- 5.5%)	69% (+/- 11.4%)	96% (+/- 4.1%)	69% (+/- 4.4%)	98% (+/- 1.1%)	96% (+/- 1.3%)

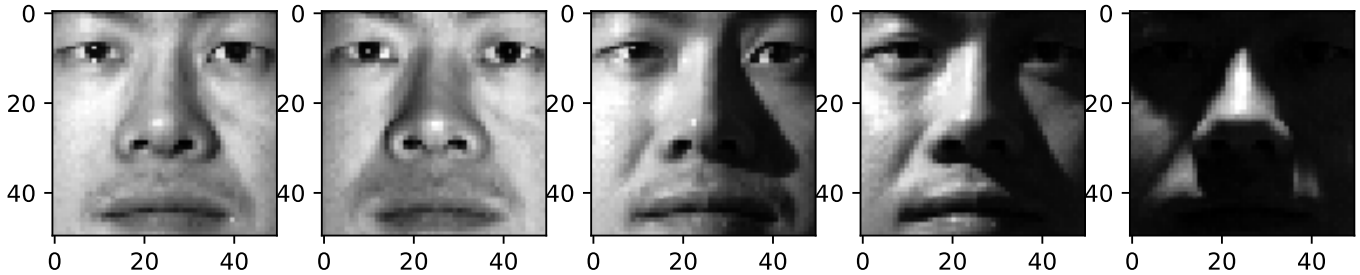
Применение

- **Распознавание лиц.** FDA сокращает количество признаков. Создает тэмплэйты,состоящие из новых размерностей (линейных комбинаций значений пикселей, называемых *фишеровскими лицами*).
- **Маркетинг.** Разделение покупателей/товаров на группы на основании опросов или других форм сбора данных.
- **Медицина.** Предсказание эффективности лечения, диагностика заболеваний:
  - Обработка и создание комплексов лабораторных тестов. Нахождение оптимальных комбинаций лабораторных тестов для диагностики заболевания.
  - Дифференциация между болезнями, схожими по проявлениям. Например, офтольмологическая проблема: отличие глаукомы от глазной гипертензии.
  - Предсказание доброкачественности/злокачественности опухоли по ее хар-кам, состоянию пациента.
  - Ранняя диагностика опухолей головного мозга. Выявление пораженных клеток по записям слуховых потенциалов ствола головного мозга.

Распознавание лиц

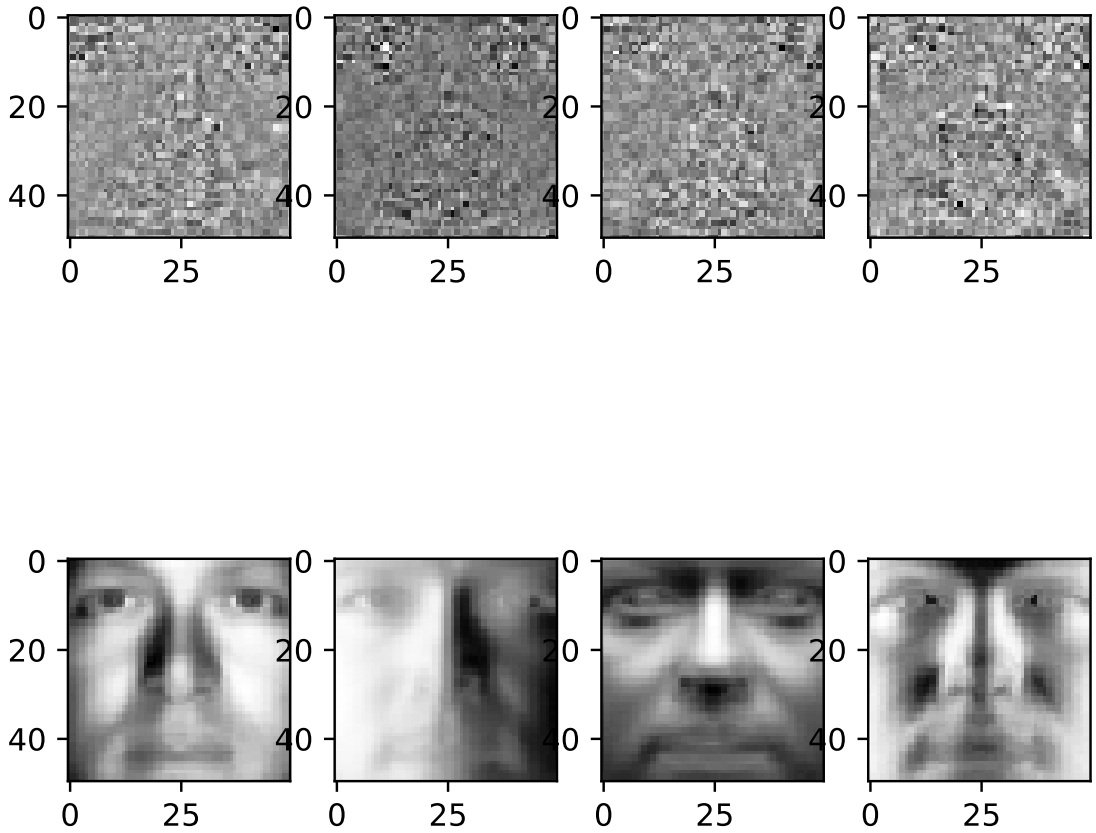
На примере классификации изображений из базы данных Faces (MIT), состоящей из 640 изображений (разрешением 50×50) 10 людей.
Пример данных:





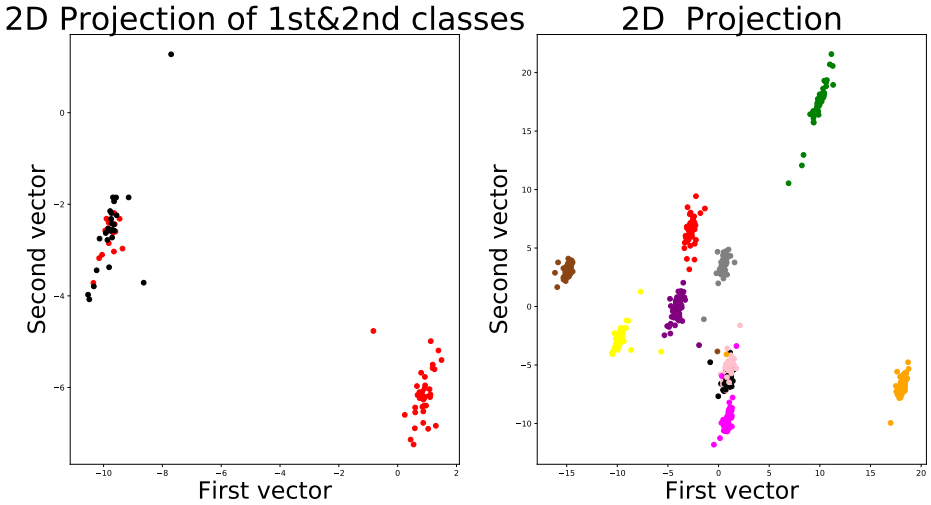
Распознавание лиц

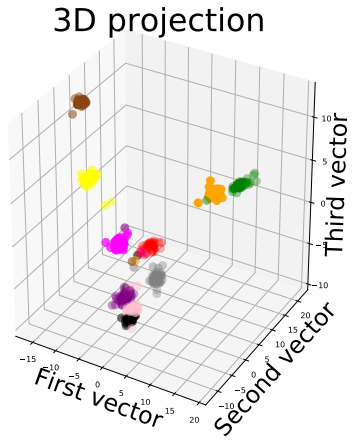
С помощью *FDA* найдем фишеровские лица, а с помощью *PCA* собственные лица:

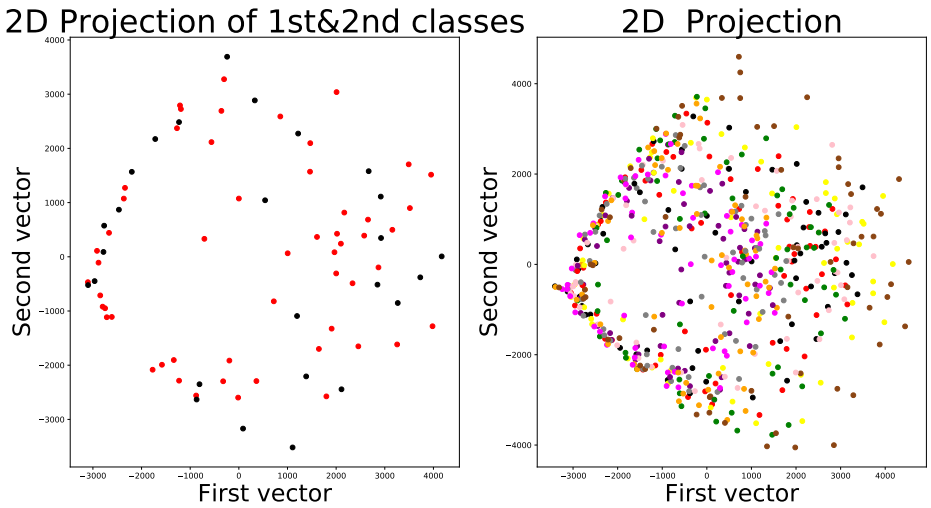


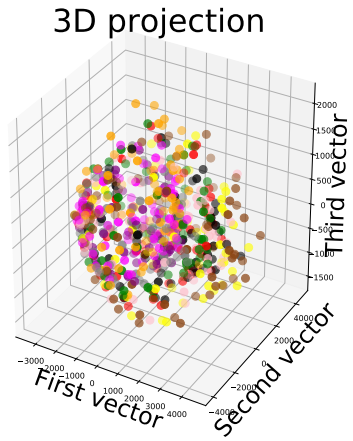
Fisherfaces (сверху), Eigenfaces (снизу)

Сравнение проекций FDA/PCA



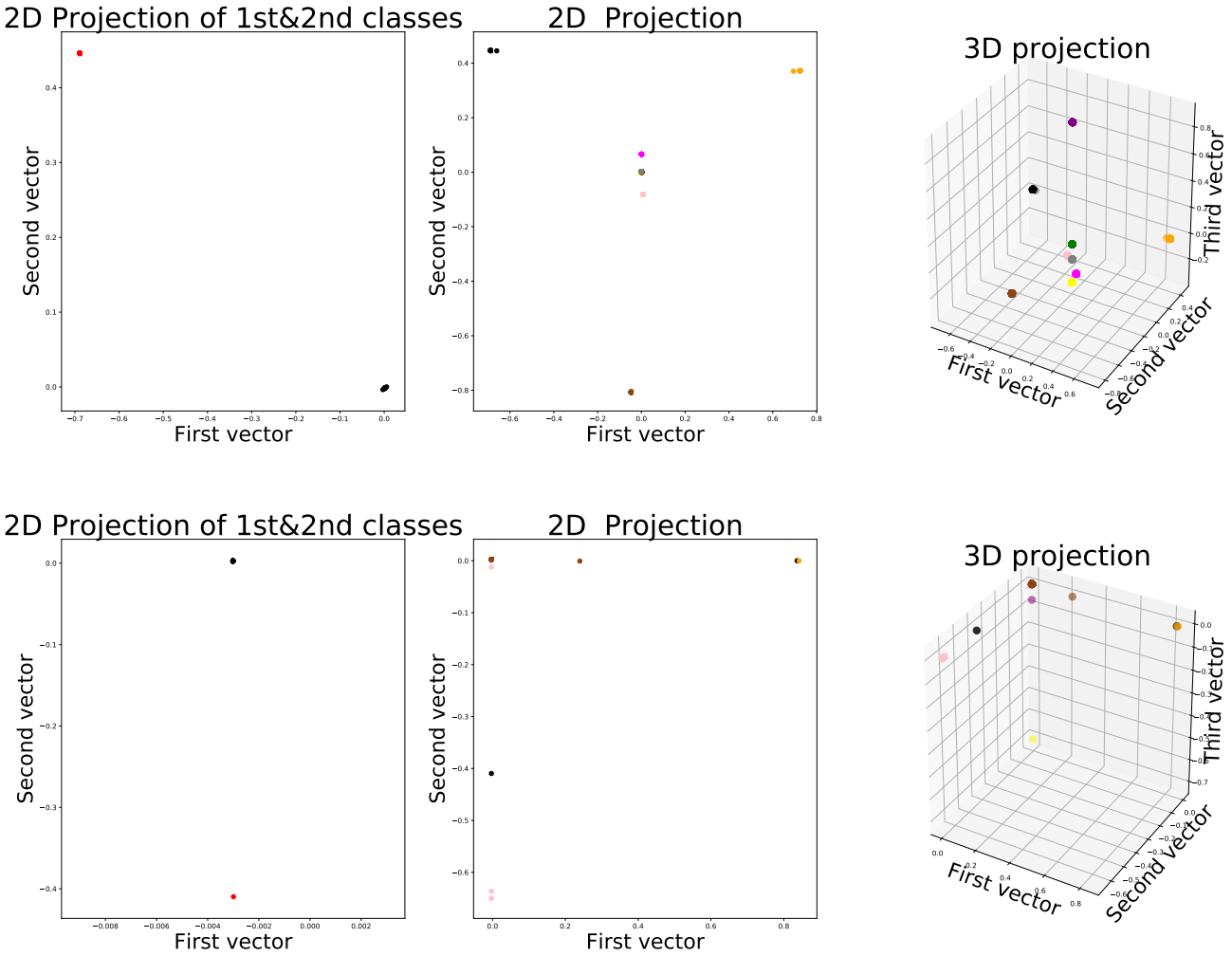






Проекция данных в подпространство, найденное с помошью *FDA* (сверху), *PCA* (снизу).

Сравнение проекций KFDA/KPCA



Проекция данных в подпространство, найденное с помошью *KFDA* (сверху), *KPCA* (снизу).

Анализ точности методов

Method	Accuracy (for Face (MIT) dataset)
PCA + KNN	55.3 %
FDA + KNN	77.5%
KPCA + KNN	84.6 %
KFDA + KNN	98.7%

а) Сравнительная таблица точности *KNN* после уменьшение размерности пространства одним из методов

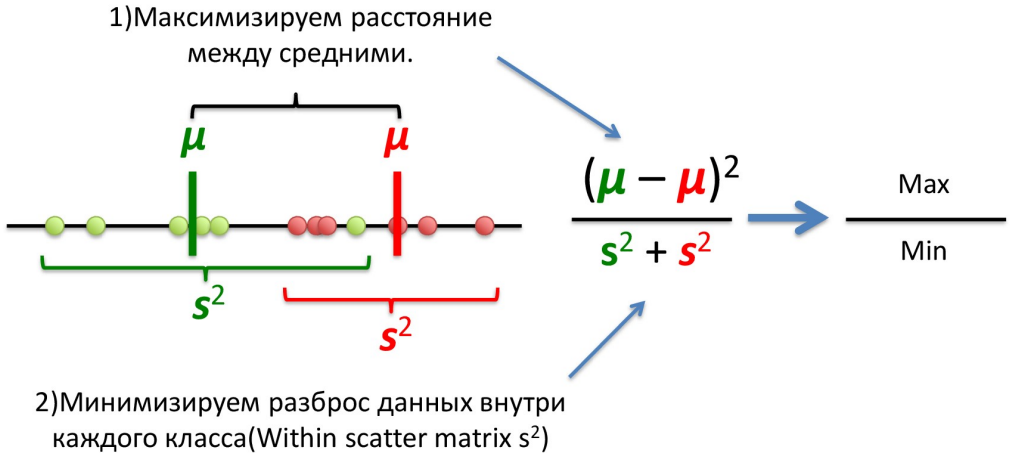
DATASETS	KFDA (RBF)	SVM (RBF)	FDA	KNN (k=5)	LDA	QDA
Toy	91% (+/- 8.5%)	85% (+/- 15.1%)	45% (+/- 14.3%)	84% (+/- 14.8%)	45% (+/- 15.6%)	63% (+/- 21.2%)
Breast Cancer	95% (+/- 0.8%)	91% (+/- 1.3%)	94% (+/- 2.0%)	93% (+/- 2.1%)	96% (+/- 0.9%)	96% (+/- 1.3%)
Iris	97% (+/- 2.1%)	97% (+/- 3.7%)	93% (+/- 3.0%)	97% (+/- 2.5%)	98% (+/- 2.7%)	98% (+/- 2.7%)
Wine	82% (+/- 5.5%)	69% (+/- 11.4%)	96% (+/- 4.1%)	69% (+/- 4.4%)	98% (+/- 1.1%)	96% (+/- 1.3%)

б) Сравнительная таблица точности для *KFDA* и других классификаторов

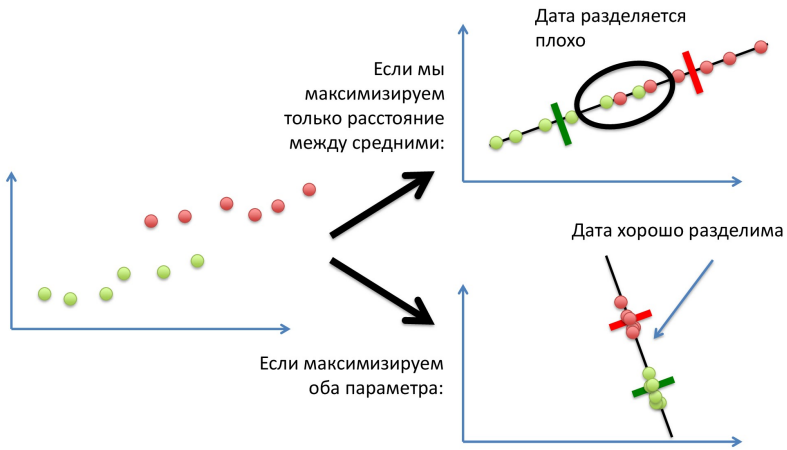
Список литературы

- Основная статья: Fisher Discriminant analysis with Kernels by Sebastian Mika.
- The Elements of Statistical Learning by Trevor Hastie, Robert Tibshirani (Раздел 4.3).
- Miller-Mika-Ratsch-Tsuda-Scholkopf "An Introduction to Kernel Based Learning Algorithms"
- Ядра и их применение в машинном обучении. Евгений Соколов.
- Statistical Pattern Recognition Toolbox for Matlab. User's guide.
- Face recognition. Fisherfaces.
- Wikipedia: KFDA и FDA.

Теоретическая справка.
Как найти наилучший вектор для проекции данных?



Оптимизировать только одну величину недостаточно:



Принцип работы FDA

Задача: найти вектор ω, в проекции на который максимально отношение

$$I = \frac{(\mu_1' - \mu_2')^2}{s_1^2 + s_2^2} = \frac{S_b'}{S_w'}$$

- Обозначим еще не спроектированные образцы двух классов как:

$$X_1 = \begin{bmatrix} x_1^1 \\ x_2^1 \\ \dots \\ x_{l_1}^1 \end{bmatrix} = \begin{bmatrix} x_{1_1}^1 & x_{1_2}^1 & \dots & x_{1_n}^1 \\ x_{2_1}^1 & x_{2_2}^1 & \dots & x_{2_n}^1 \\ \dots & & & \\ x_{l_1 1}^1 & x_{l_1 2}^1 & \dots & x_{l_1 n}^1 \end{bmatrix}, X_2 = \begin{bmatrix} x_1^2 \\ x_2^2 \\ \dots \\ x_{l_2}^2 \end{bmatrix}$$

- $$\mu_1 = \begin{bmatrix} \frac{1}{l_1}\sum_{i=1}^{l_1} x_{i1}^1, & \dots, \frac{1}{l_1}\sum_{i=1}^{l_1} x_{in}^1 \end{bmatrix}$$
$$\mu_2 = \dots$$

- Значение scatter matrix S_w для исходных данных:

$$S_w = S_1 + S_2 = \sum_{x \in X_1} (x - \mu_1)^T (x - \mu_1) + \sum_{x \in X_2} (x - \mu_2)^T (x - \mu_2) =$$

$$\begin{bmatrix} x^1_{1_1} - \mu_{11} & x^1_{1_2} - \mu_{12} & \dots & x^1_{1_n}\mu_{1n} \\ \dots & & & \\ x^1_{1_1} - \mu_{11} & x^1_{1_2} - \mu_{12} & \dots & x^1_{1_n}\mu_{1n} \end{bmatrix}^T \begin{bmatrix} x^1_{1_1} - \mu_{11} & x^1_{1_2} - \mu_{12} & \dots & x^1_{1_n}\mu_{1n} \\ \dots & & & \\ x^1_{1_1} - \mu_{11} & x^1_{1_2} - \mu_{12} & \dots & x^1_{1_n}\mu_{1n} \end{bmatrix}$$

$$+ \sum_{x \in X_2} (x - \mu_2)^T (x - \mu_2)$$

- Спроецированные данные: $x'_i = \omega^T x_i^T$.
- Значение scatter matrix S_w для спроектированных данных:

$$S'_w = \sum_{i=1,2} \sum_{x' \in X_i} (x'_i - \mu'_i)^2 = \sum_{i=1,2} \sum_{x' \in X_i} (\omega^T x_i - \omega^T \mu_i)^2 =$$

$$= \sum_{i=1,2} \sum_{x_i \in X_i} (x_i - \mu_i)^T (x_i - \mu_i) \omega^T = \omega S_w \omega^T$$

- Межклассовые scatter matrix S_b для исходных и спроецированных данных:

$$S_b = (\mu_1 - \mu_2)^T(\mu_1 - \mu_2)$$

$$S'_b = (\mu'_1 - \mu'_2)^T(\mu'_1 - \mu'_2) = \omega S_b \omega$$

Подставим в выражение для ω:

$$\omega = \underset{\omega}{\operatorname{argmax}}\, I = \underset{\omega}{\operatorname{argmax}}\, J(\omega) = \underset{\omega}{\operatorname{argmax}}\, \frac{\omega S_b \omega^T}{\omega S_w \omega^T}$$

Найдем точку экстремума функции[3]1: $J(\omega)$:

$$\frac{d}{d\omega} J(\omega) = \frac{(\frac{d}{d\omega}\omega^T S_b \omega)\omega^T S_w \omega - (\frac{d}{d\omega}\omega^T S_w \omega)\omega^T S_b \omega}{(\omega^T S_w \omega)^2} =$$

$$= \frac{(2S_b\omega)\omega^T S_w\omega - (2S_w\omega)\omega^T S_b\omega}{(\omega^T S_w\omega)^2} = 0$$

$$\Rightarrow (S_b\omega)\omega^T S_w\omega - (S_w\omega)\omega^T S_b\omega = 0$$

Разделим на $\omega^T S_w\omega$:

$$S_b\omega - S_w\omega \cdot \frac{\omega^T S_b\omega}{\omega^T S_w\omega} = 0$$

$$\Rightarrow S_b\omega = I \cdot (S_w\omega)$$

---

[3] Мы пользуемся, утверждением, что

$$\frac{d}{d\omega}\omega^T A\omega = \sum_{p=0}^{n} \frac{d}{d\omega_p} x^T Ax = \sum_{p=0}^{n}\sum_{i,j=1}^{n} a_{ij}x_i x_j = \sum_{p=0}^{n}(2a_{pp}x_p + \sum_{j\neq p} a_{pj}x_j + \sum_{i\neq p} a_{ip}x_i) =$$

$$= \sum_{p=0}^{n}(\sum_{j=1}^{n} a_{pj}x_j + \sum_{i=1}^{n} a_{ip}x_i) = \sum_{p=0}^{n}((Ax)_p + (A^T x)_p) = (A + A^T)x$$

Если $\det(S_w) \neq 0$, то задача сводится к:

$$S_w^{-1} S_b \cdot \omega = I \cdot \omega$$

Т. к. S_b симметричная положительно-определенная матрица, то она может быт представлена в виде:

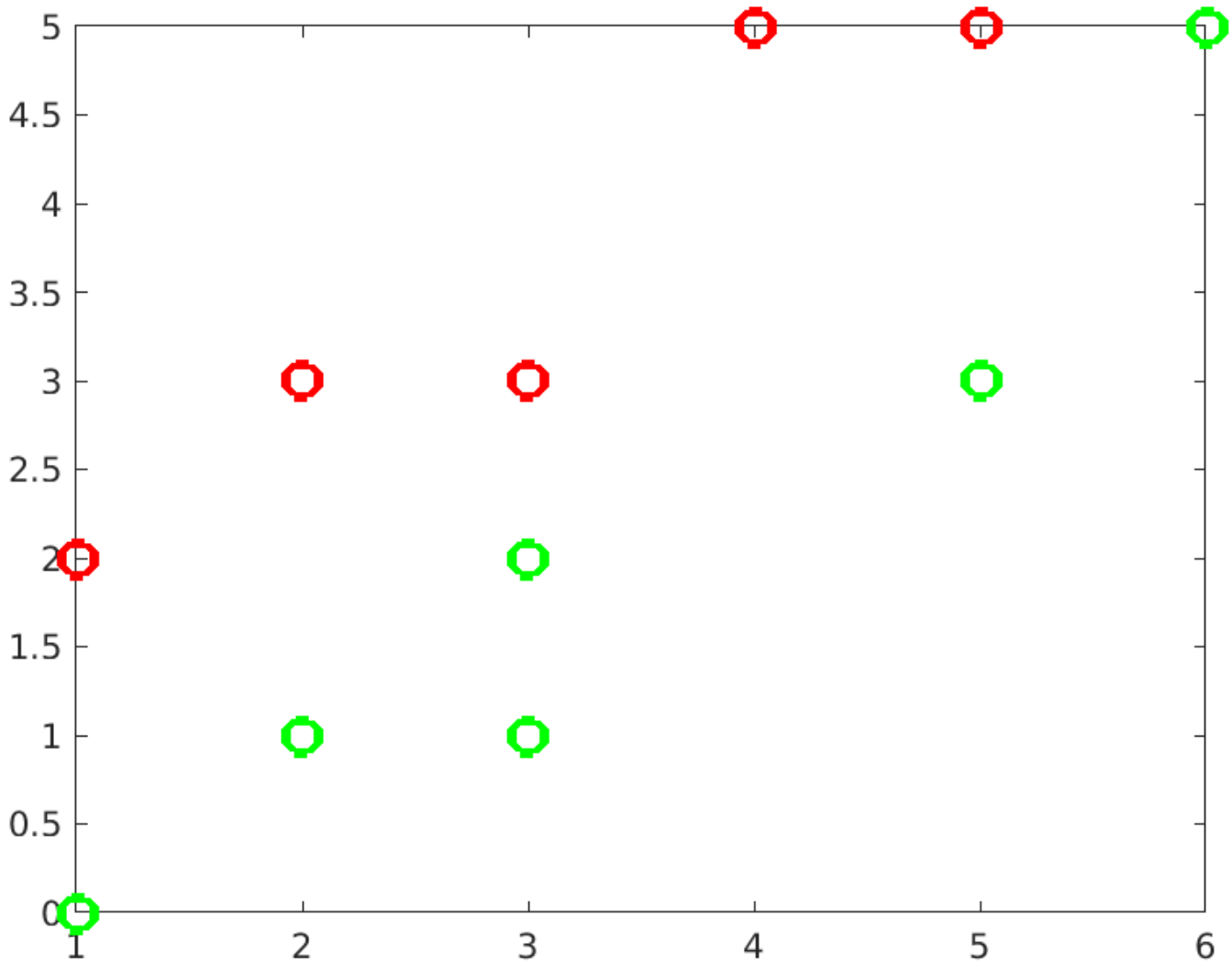
$$S_b = U \Lambda U^T = S_b^{\frac{1}{2}} S_b^{\frac{1}{2}},$$

где $S_b^{\frac{1}{2}} = U \Lambda^{\frac{1}{2}} U^T$. Подставим в уравнение, обозначив $v = S_b^{\frac{1}{2}} \omega$:

$$S_b^{\frac{1}{2}} S_w^{-1} S_b^{\frac{1}{2}} \cdot v = I \cdot v$$

$\Rightarrow \omega = \text{argmax}_\omega I = S_b^{-\frac{1}{2}} v$, где v собственный вектор, соответствующий наибольшему собственному значению матрицы $S_b^{\frac{1}{2}} S_w^{-1} S_b^{\frac{1}{2}}$.

Пример работы алгоритма



Пример работы алгоритма

- Данные: Первый класс и второй класс :

$$c_1 : \left[\ (1,2),(2,3),(3,3),(4,5),(5,5) \ \right]$$

$$c_2 : \left[\ (1,0),(2,1),(3,1),(3,2),(5,3),(6,5) \ \right]$$

- Запишем данные в матричном виде:

$$c_1 = \begin{bmatrix} 1 & 2 \\ \dots & \dots \\ 5 & 5 \end{bmatrix}, c_2 = \begin{bmatrix} 1 & 0 \\ \dots & \dots \\ 6 & 5 \end{bmatrix}$$

- Средние значения по каждому из классов и общее среднее:

$$\mu_1 = \begin{bmatrix} 3 & 3.6 \end{bmatrix}, \mu_2 = \begin{bmatrix} 3.3 & 2 \end{bmatrix}$$

$$\mu = \begin{bmatrix} 3.18 & 2.73 \end{bmatrix}$$

- Отклонение от среднего среди образцов каждого класса:

$$c_1 - \mu_1 = \begin{bmatrix} -2 & -1.6 \\ \dots & \dots \\ 2 & 1.4 \end{bmatrix}, c_2 - \mu_2 = \begin{bmatrix} -2.3 & -2 \\ \dots & \dots \\ 1.7 & 3 \end{bmatrix}$$

- Вычислим внутриклассовую(within) scatter матрицу S_w:

$$S_w = (c_1 - \mu_1)^T (c_1 - \mu_1) + (c_2 - \mu_2)^T (c_2 - \mu_2) =$$

$$\begin{bmatrix} -2 & -1.6 \\ \dots & \dots \\ 2 & 1.4 \end{bmatrix}^T \begin{bmatrix} -2 & -1.6 \\ \dots & \dots \\ 2 & 1.4 \end{bmatrix} + \begin{bmatrix} -2.3 & -2 \\ \dots & \dots \\ 1.7 & 3 \end{bmatrix}^T \begin{bmatrix} -2.3 & -2 \\ \dots & \dots \\ 1.7 & 3 \end{bmatrix}$$

$$= \begin{bmatrix} 27.3 & 24 \\ 24 & 23.2 \end{bmatrix}; \quad S_w^{-1} = \begin{bmatrix} 0.39 & -0.41 \\ -0.41 & 0.47 \end{bmatrix}$$

- Теперь межклассовую(between) scatter матрицу

$$S_b = (\mu_1 - \mu_2)^T (\mu_1 - \mu_2) = \begin{bmatrix} 0.3 & -1.45 \\ -1.45 & 6.98 \end{bmatrix}$$

-

$$S_b S_w^{-1} = \begin{bmatrix} 0.3 & -1.45 \\ -1.45 & 6.98 \end{bmatrix} \begin{bmatrix} 0.39 & -0.41 \\ -0.41 & 0.47 \end{bmatrix} = \begin{bmatrix} 0.71 & -0.8 \\ -3.43 & 3.88 \end{bmatrix}$$

- Собственные значения и соответсвующее им собственные вектора:

$$\lambda_1 = 0 \ : \begin{bmatrix} -0.98 \\ -0.2 \end{bmatrix}, \quad \lambda_2 = 4.6 \ : \begin{bmatrix} 0.66 \\ -0.75 \end{bmatrix}$$

- Находим w - искомый вектор проекции, как вектор соответствующий макс собственному значению:

$$\omega = \begin{bmatrix} 0.66 & -0.75 \end{bmatrix}^T$$

- Проецируем дату на любую прямую, имеющую направление w:

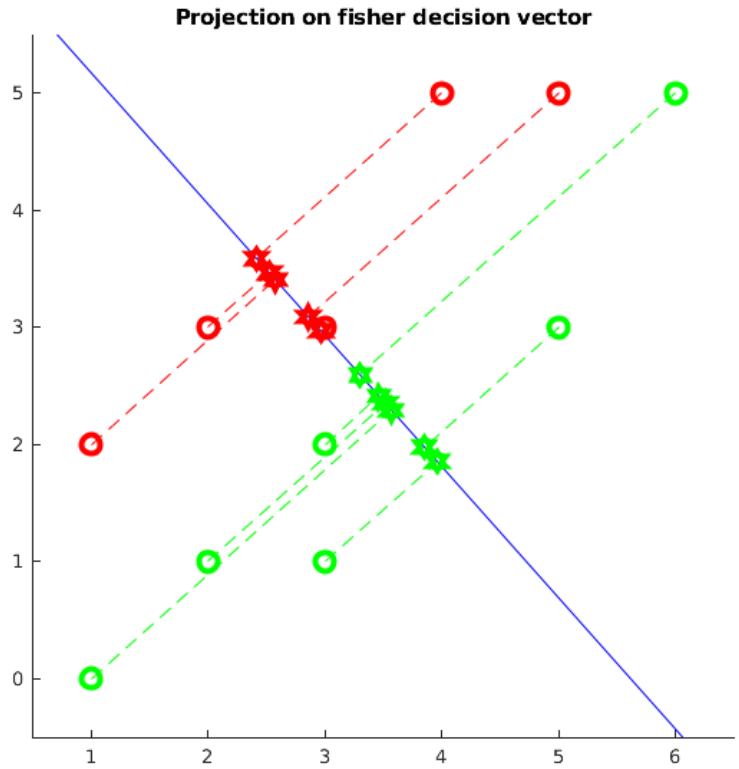


Рис. 2: Спроецированная дата легко разделяется на классы (например по расстоянию от среднего каждого класса)

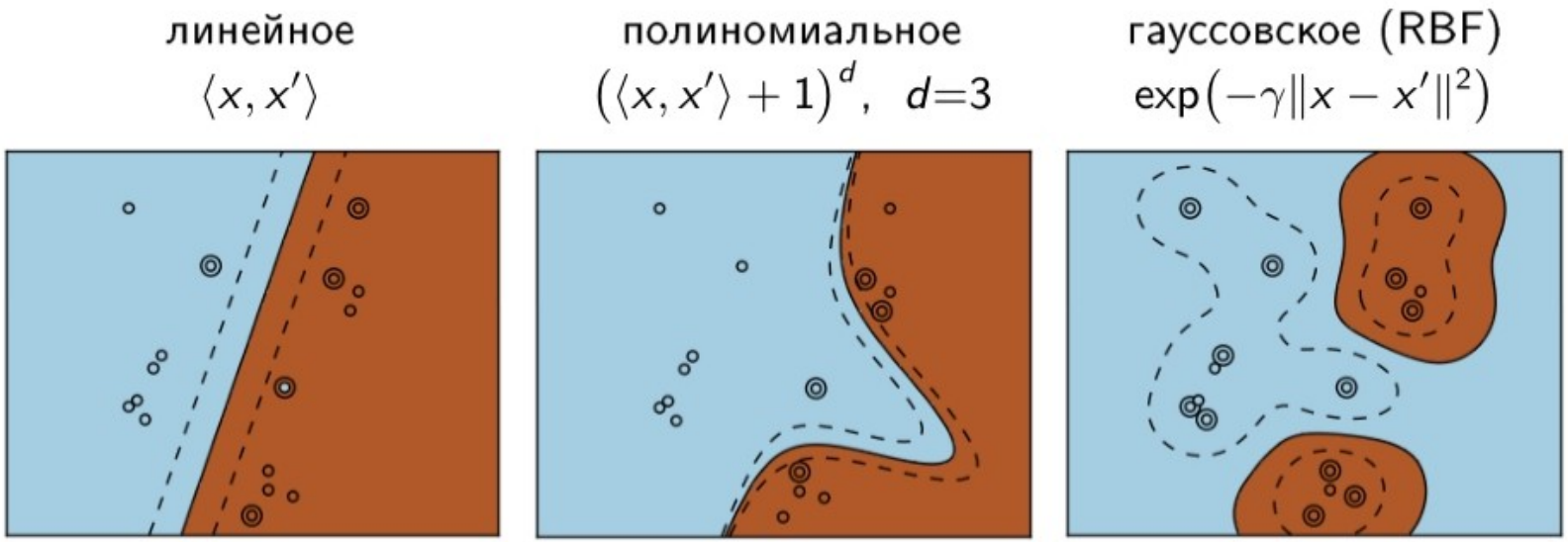
Типы ядер

Th. Мерсера

Функция K(x, z) является ядром тогда и только тогда, когда:

- Она симметрична: K(x, z) = K(z, x).
- Она неотрицательно определена, то есть для любой конечной выборки $(x_1 \dots x_l)$ матрица $K = (K(x_i, x_j))_{i,j=1}^{l}$ неотрицательно определена.

Наиболее распространенные ядра[4]:

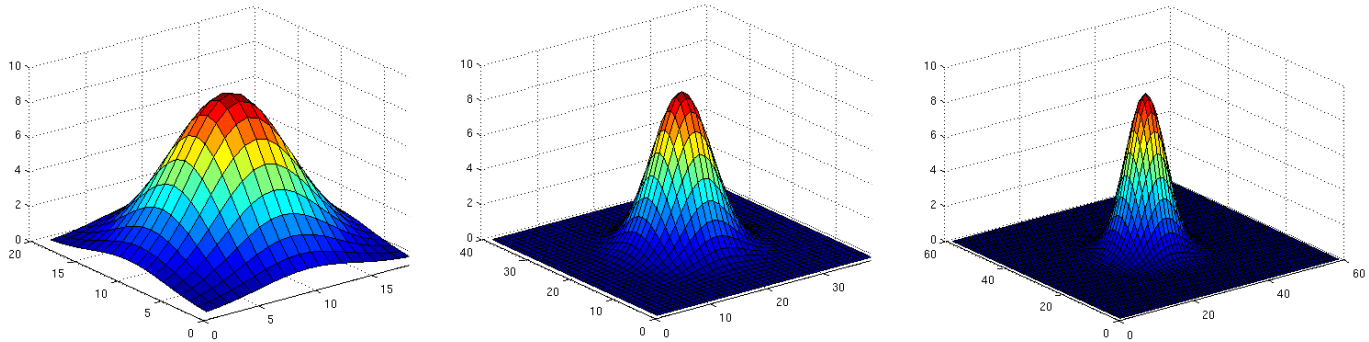


[4] Рис., сгенерирован кодом из scikit-learn example.

Работа в бесконечномерном пространстве
Gaussian Kernel

Гауссовское ядро (RBF)

$$K(x_i, x_j) = exp(\frac{-||x_i - x_j||^2}{2\sigma^2})$$



$$exp(\frac{-||x_i - x_j||^2}{2\sigma^2}) = exp(\frac{-(||x_i||^2 + ||x_j||^2)}{2\sigma^2} exp(\frac{(x_i, x_j)}{\sigma^2}) =$$

$$= C(1 - C_1(x_i, x_j) + C_2(x_i, x_j)^2 - C_3(x_i, x_j)^3 \cdots + (-1)^n C_n(x_i, x_j)^n \dots)$$

$\Rightarrow$ *Если дата разделима в ∞ пространстве, то применив RBF Kernel ее можно классифицировать в пространстве исходной размерности.*

Переход от FDA к KFDA I

$$k(x_p, x_t) = (\Phi(x_p) \cdot \Phi(x_t))$$

$$\omega = \underset{\omega}{\text{argmax}} \frac{\omega S_b^\Phi \omega^T}{\omega S_w^\Phi \omega^T}$$

$$\mu_i^\Phi = \frac{1}{l_i} \sum_{j=1}^{l_i} \Phi(x_j^i)$$

$$S_w^\Phi = S_1^\Phi + S_2^\Phi = \sum_{i=1,2} \sum_{x \in X_i} (\Phi(x_i) - \mu_i^\Phi)^T (\Phi(x_i) - \mu_i^\Phi)$$

$$S_b^\Phi = (\mu_1^\Phi - \mu_2^\Phi)^T (\mu_1^\Phi - \mu_2^\Phi)$$

Вектор ω линейная оболочка векторов $\Phi(x_i)$:

Переход от FDA к KFDA II

$$\omega = \sum_{i=1}^{l} a_i \Phi(x_i)$$

Домножим на μ_i :

$$\mu_i^{\Phi} \omega = \frac{1}{l_i} \sum_{j=1}^{l} \sum_{k=1}^{l_i} a_j k(x_j, x_k^i) = M_i a,$$

где $(M_i)_j = \frac{1}{l_i} \sum_{k=1}^{l_i} k(x_j, x_k^i)$

$$\Rightarrow \omega S_b^{\Phi} \omega^T = \omega(\mu_1^{\Phi} - \mu_2^{\Phi})^T(\mu_1^{\Phi} - \mu_2^{\Phi})\omega^T = aMa^T,$$

где $M = (M_1 - M_2)^T(M_1 - M_2)$.

Переход от FDA к KFDA III

Аналогично, получим:

$$\omega S_w^{\Phi} \omega^T = aNa^T,$$

где $N = \sum_{j=1,2} K_j(E - 1_{li})K_j^T$, где 1_{li} -матрица с элементами $1/l_i$.

$$\Rightarrow \omega = \underset{\omega}{\operatorname{argmax}} \frac{\omega S_b^{\Phi} \omega^T}{\omega S_w^{\Phi} \omega^T} = \frac{a^T M a}{a^T N a}$$

Таким образом, задача свелась к уже решенной:

Переход от FDA к KFDA IV

$\Rightarrow$ a - собственный вектор, соответсвующий наиб. собств. знач. $N^{-1}M$.

$$\omega = \sum_{i=1}^{l} a_i \Phi(x_i)$$

$\Rightarrow$ Проекция данных на ω:

$$(\omega, \Phi(x)) = \sum_{i=1}^{l} a_i k(x_i, x)$$

KFDA для многоклассовой задачи

- В случае **c** классов данные можно спроецировать в $(c-1)$-мерное пространство.
- w - матрица проекции, состоящая из $(c-1)$ вектора.
- Чтобы значение функции J(ω) был скаляр, функция записывается как:

$$J(a) = \frac{\det(\omega S_b \omega^T)}{\det(\omega S_w \omega^T)}$$

Алгоритм

$$(M_i)_j = \frac{1}{l_i} \sum_{k=1}^{l_i} k(x_j, x_k^i)$$

$$M = \sum_{i=1}^{c} (M_0 - M_i)^T (M_0 - M_i)$$

$$N = \sum_{j=1}^{c} K_j (Id - 1_{l_i}) K_j^T$$

a - собственный вектор $(M^{-1}N)$, соотв. максимальному собственному значению.

$$(\omega \cdot \Phi(x)) = K \cdot a$$

Дискриминантный анализ. Общие принципы и методы

- DA предполагает, что размерность пр-ва(кол-во predictors p) меньше p кол-во образцов n. Точность классификатора "сохраняется"при $n \geq 5p$.
- DA предсказывает вероятность попадения образцов, с определнным значением предикта, в конкретный класс отдельно для каждого класса.
- Используя Th. Байеса по этим вероятностям DA находит для определенного образца и класса *discriminant score*, определяющее вероятность его принадлежности к данному классу.
- DA относит образец к классу, для которого discriminant score макс.

Предположив, что образцы в каждом классе имеют ***нормальное распределение*** получим формулу, задающую discriminat score для k-го класса (подробнеее см. в статье):

$\delta_k(x) = -\frac{l_k}{2} x^T S_k^{-1} x + l_k x^T S_k^{-1} \mu_k - \frac{l_k}{2} \mu_k^T S_k^{-1} \mu_k - \frac{1}{2} log|S_k| + \frac{1}{2} log l_k + + log(\pi_k)$ (1), π_k доля образцов класса k среди всех образцов.

Другие методы DA. LDA и QDA

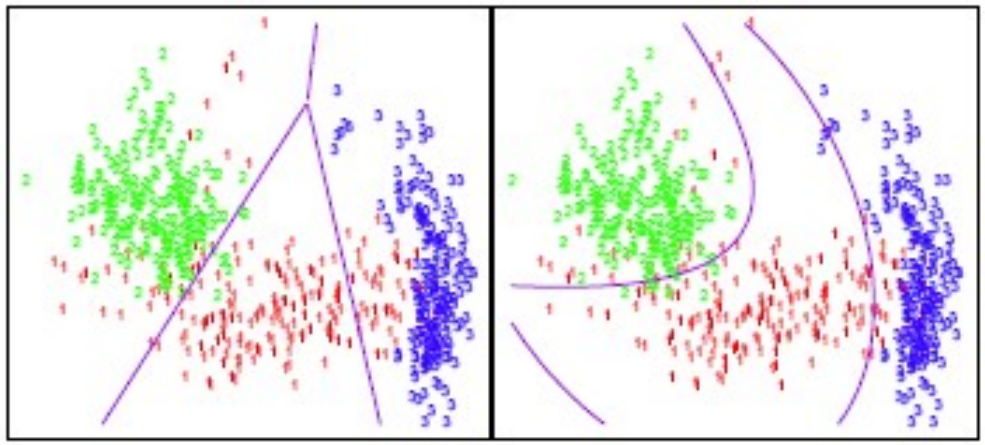
Формула (1) задает квадратичное Decision Boundary ($\delta_1(x) = \delta_2(x)$) и лежит в основе метода **QDA**.

В **LDA** используется предположение, что распределение образцов в каждом классе (σ) одинаково, и формула(1) упрощается:

LDA

$\delta_k(x) = xl_k S_k^{-1}\mu_k - \frac{l_k}{2}\mu_k^T S_k^{-1} - \mu_k + log(\pi_k)$

l_k - количество образцов в классе k. Decision Boundary имеет линейную форму.



Распределние образцов внутри классов различно, и QDA работает лучше