Белорусский государственный университет информатики и радиоэлектроники

# Кафедра ЭВМ

### Отчет по лабораторной работе

Проверил:

Татур М.М.

Выполнил:

студент группы 6М2411

Белов А.В.

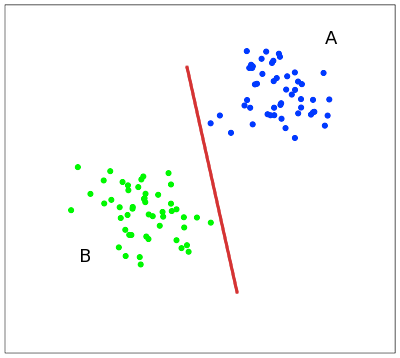
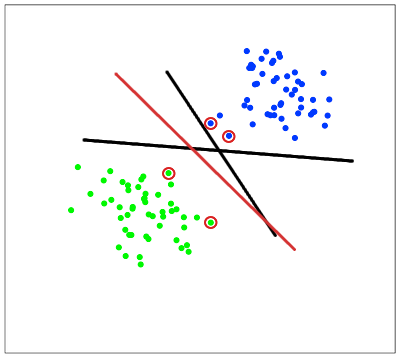
Минск 2017

1. **Цель работы**

Изучение технологий принятия решений на примере построения классификатора.

1. **Краткие теоретические сведения**

**kNN** расшифровывается как *k Nearest Neighbor* или *k Ближайших Соседей* — это один из самых простых алгоритмов классификации, также иногда используемый в задачах регрессии. Благодаря своей простоте, он является хорошим примером, с которого можно начать знакомство с областью Machine Learning. В данной статье рассмотрен пример написания кода такого классификатора на python, а также визуализация полученных результатов.  
*Задача классификации* в машинном обучении — это задача отнесения объекта к одному из заранее определенных классов на основании его формализованных признаков. Каждый из объектов в этой задаче представляется в виде вектора в N-мерном пространстве, каждое измерение в котором представляет собой описание одного из признаков объекта. Допустим нам нужно классифицировать мониторы: измерениями в нашем пространстве параметров будут величина диагонали в дюймах, соотношение сторон, максимальное разрешение, наличие HDMI-интерфейса, стоимость и др. Случай классификации текстов несколько сложнее, для них обычно используется матрица термин-документ ([описание на machinelearning.ru](http://www.machinelearning.ru/wiki/index.php?title=%D0%92%D0%B5%D0%BA%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C)).  
Для обучения классификатора необходимо иметь набор объектов, для которых заранее определены классы. Это множество называется *обучающей выборкой*, её разметка производится вручную, с привлечением специалистов в исследуемой области. Например, в задаче [Detecting Insults in Social Commentary](http://www.kaggle.com/c/detecting-insults-in-social-commentary) для заранее собранных тестов комментариев человеком проставлено мнение, является ли этот комментарий оскорблением одного из участников дискуссии, само же задание является примером бинарной классификации. В задаче классификации может быть более двух классов (многоклассовая), каждый из объектов может принадлежать более чем к одному классу (пересекающаяся).

Идею метода SVM удобно проиллюстрировать на следующем простом примере: даны точки на плоскости, разбитые на два класса (рис. 1). Проведем линию, разделяющую эти два класса (красная линия на рис. 1). Далее, все новые точки (не из обучающей выборки) автоматически классифицируются следующим образом:  
  
точка выше прямой попадает в класс **A**,  
точка ниже прямой — в класс **B**.  
  
  
  
Такую прямую назовем *разделяющей* прямой. Однако, в пространствах высоких размерностей прямая уже не будет разделять наши классы, так как понятие «ниже прямой» или «выше прямой» теряет всякий смысл. Поэтому вместо прямых необходимо рассматривать гиперплоскости — пространства, размерность которых на единицу меньше, чем размерность исходного пространства. В https://habrastorage.org/storage/habraeffect/3e/89/3e89720bd70e886270233185080bfd1a.png, например, гиперплоскость — это обычная двумерная плоскость.  
  
В нашем примере существует несколько прямых, разделяющих два класса (рис. 2):   
  
  
  
С точки зрения точности классификации лучше всего выбрать прямую, расстояние от которой до каждого класса максимально. Другими словами, выберем ту прямую, которая разделяет классы наилучшим образом (красная прямая на рис.2). Такая прямая, а в общем случае — гиперплоскость, называется оптимальной разделяющей гиперплоскостью.  
  
Вектора, лежащие ближе всех к разделяющей гиперплоскости, называются *опорными векторами* (support vectors). На рисунке 2 они помечены красным.

1. **Реализация**

Для реализации методов KNN и SVM использовалась библиотека scikit-learn для языка Python, а также вспомогательные библиотеки работы с данными numpy и построения графиков matplotlib.

Перечень импортируемых функций:

|  |
| --- |
|  |

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.colors import ListedColormap

from sklearn import neighbors, svm, datasets

from sklearn.metrics import confusion\_matrix, roc\_curve, auc, f1\_score, precision\_score, recall\_score

Пример вывода результатов кластеризации:

# Plot the decision boundary. For that, we will assign a color to each

# point in the mesh [x\_min, x\_max]x[y\_min, y\_max].

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),

Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])

Z\_classes = clf.predict(X)

Z = Z.reshape(xx.shape)

plt.pcolormesh(xx, yy, Z, cmap=cmap\_light)

# Plot also the training points

plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap\_bold)

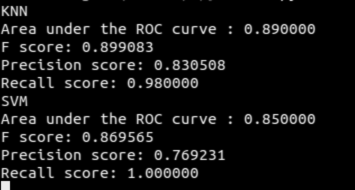
plt.xlim(xx.min(), xx.max())

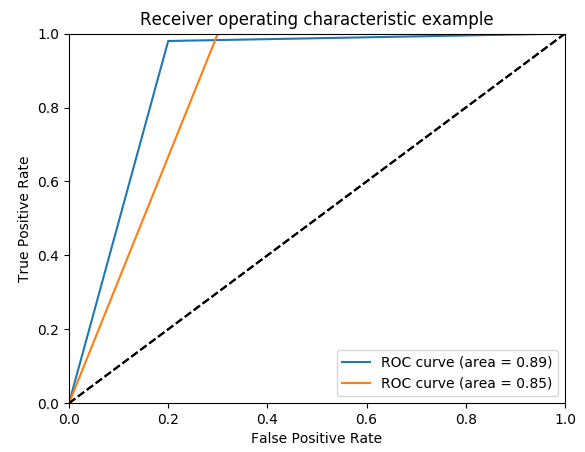
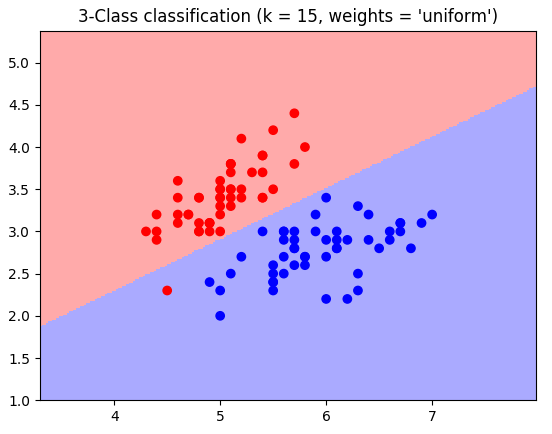
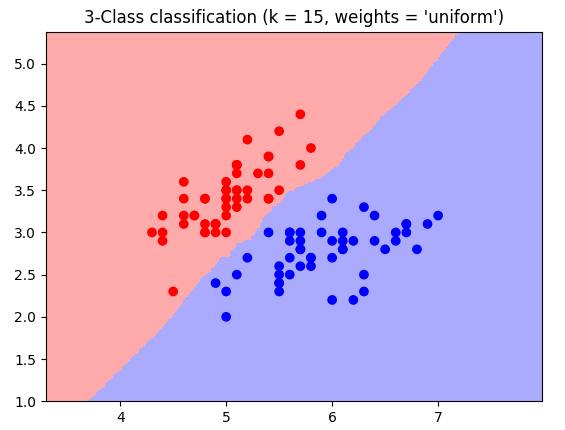
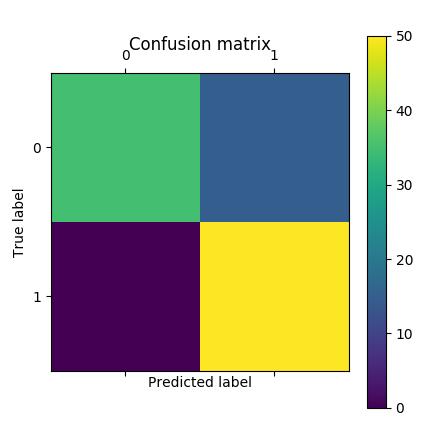
plt.ylim(yy.min(), yy.max())

plt.title("3-Class classification (k = %i, weights = '%s')"

% (n\_neighbors, weights))

1. **Результат работы программы**





1. **Вывод**

Для датасета Iris были реализованы классификаторы на основе метода K ближайших соседей и опорных векторов, построены ROC-кривые и матрицы соответствия. Были подсчитаны основные метрики, характерные для алгоритмов классификации.