**Методические указания к лабораторной работе №1**

Основной библиотекой, которую рекомендуется использовать в данном курсе работ, является библиотека scikit-learn (<http://scikit-learn.org>). Библиотека scikit-learn предоставляет реализацию целого ряда алгоритмов для обучения с учителем (Supervised Learning) и обучения без учителя (Unsupervised Learning) через интерфейс для языка программирования Python.

В данной работе нам потребуются следующие модули:

confusion\_matrix - <http://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html>

classification\_report – <http://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html>

accuracy\_score - <http://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html>

metrics - <http://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html#sklearn.metrics.roc_auc_score>

train\_test\_split - <http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html>

LogisticRegression - <http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html>

KNeighborsClassifier - <http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>

RandomForestClassifier - <http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

GaussianNB - <http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html>

А также, в зависимости от варианта задания – один из модулей встроенных генераторов датасетов –

make\_blobs - <http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html>

make\_moons - <http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html>

make\_classification - <http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html#sklearn.datasets.make_classification>

Кроме того, в работе будет использоваться библиотека *NumPy* [*http://www.numpy.org*](http://www.numpy.org)*,* позволяющая работать с многомерными массивами и высокоуровневыми математическими функциями.

Программный код для импорта указанных модулей может выглядеть следующим образом:

|  |
| --- |
| import numpy as np  from sklearn.datasets import make\_blobs  from sklearn.metrics import confusion\_matrix  from sklearn.metrics import classification\_report  from sklearn.metrics import accuracy\_score  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LogisticRegression  from sklearn.neighbors import KNeighborsClassifier  from sklearn.ensemble import RandomForestClassifier  from sklearn.naive\_bayes import GaussianNB |

Для построения графиков рекомендуется использовать библиотеку matplotlib и ее модуль pyplot <https://matplotlib.org/users/pyplot_tutorial>

А для отображения на графике области принятия решения - готовую функцию plot\_2d\_separator, которой нужно передать на вход объект classifier – модель классификатора и X – массив входных данных:

|  |
| --- |
| import matplotlib.pyplot as plt  def plot\_2d\_separator(classifier, X, fill=False, line=True, ax=None, eps=None):  if eps is None:  eps = 1.0 #X.std() / 2.  x\_min, x\_max = X[:, 0].min() - eps, X[:, 0].max() + eps  y\_min, y\_max = X[:, 1].min() - eps, X[:, 1].max() + eps  xx = np.linspace(x\_min, x\_max, 100)  yy = np.linspace(y\_min, y\_max, 100)  X1, X2 = np.meshgrid(xx, yy)  X\_grid = np.c\_[X1.ravel(), X2.ravel()]  try:  decision\_values = classifier.decision\_function(X\_grid)  levels = [0]  fill\_levels = [decision\_values.min(), 0, decision\_values.max()]  except AttributeError:  # no decision\_function  decision\_values = classifier.predict\_proba(X\_grid)[:, 1]  levels = [.5]  fill\_levels = [0, .5, 1]  if ax is None:  ax = plt.gca()  if fill:  ax.contourf(X1, X2, decision\_values.reshape(X1.shape),  levels=fill\_levels, colors=['cyan', 'pink', 'yellow'])  if line:  ax.contour(X1, X2, decision\_values.reshape(X1.shape), levels=levels, colors="black")  ax.set\_xlim(x\_min, x\_max)  ax.set\_ylim(y\_min, y\_max)  ax.set\_xticks(())  ax.set\_yticks(()) |

**Генерация выборки**

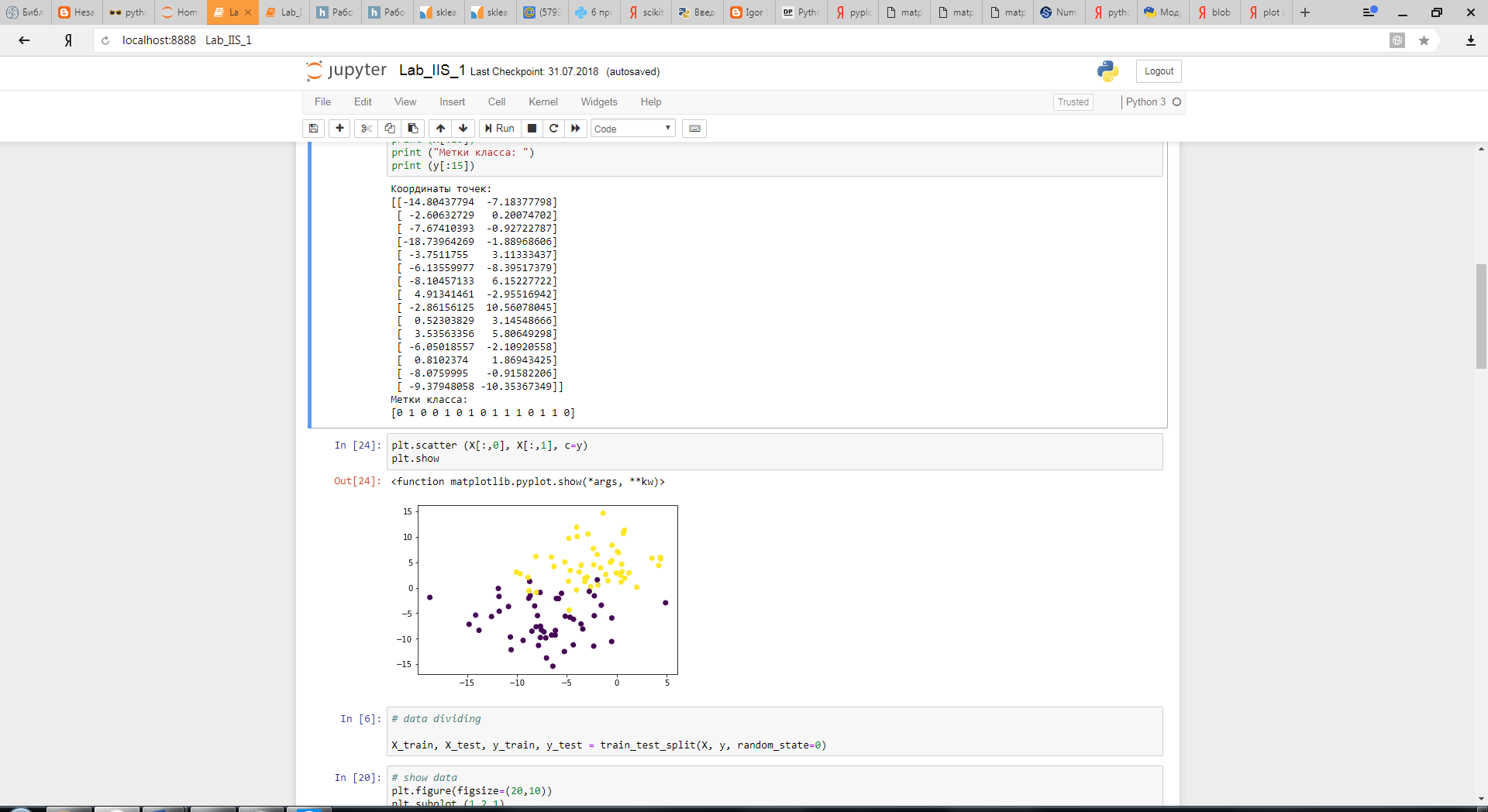
Сгенерируем данные, с которыми будем работать. В нашем случае это будут 2 «пузыря» (blob). Передадим в качестве параметра centers = 2 – количество классов-пузырей, random\_state = 66 – основа, используемая для генерации случайных чисел, cluster\_std = 4 – стандартное отклонение кластеров, shuffle = 1 – перемешиваем объекты внутри выборки.

В массив с именем Х сохраним координаты каждого объекта выборки, а в массив у – метки классов.

|  |
| --- |
| X, y = make\_blobs(centers = 2 , random\_state = 66, cluster\_std = 4, shuffle = 1) |

Посмотрим, что из себя представляют массивы Х и у. Выведем первые 15 элементов каждого из массивов.

|  |
| --- |
| print ("Координаты точек: ")  print (X[:15])  print ("Метки класса: ")  print (y[:15]) |



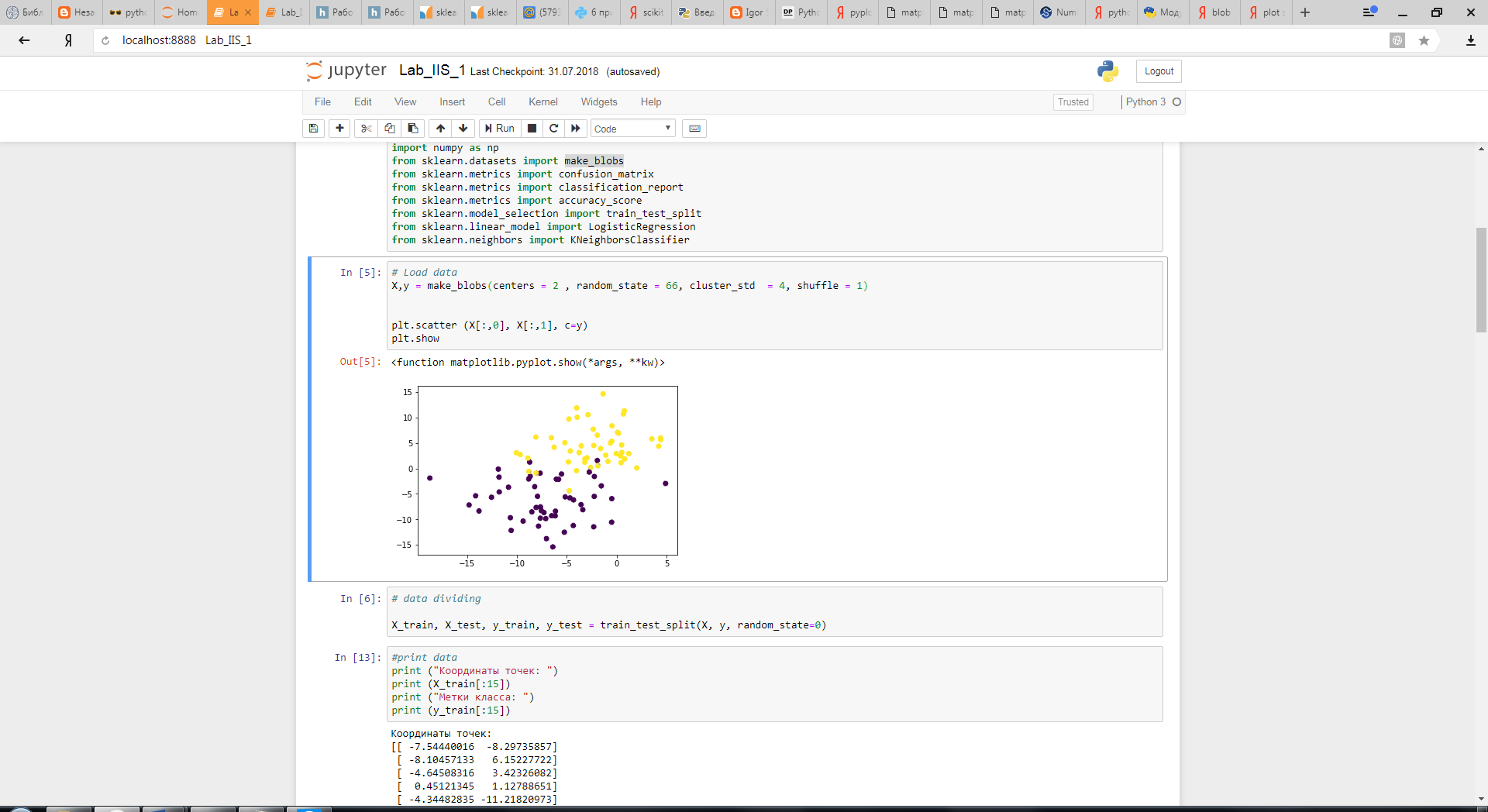
Первый элемент выборки с координатами [-14.80437794 -7.18377798] относится к классу 0,

второй элемент [ -2.60632729 0.20074702] – к классу 1 и т.д.

Отобразим на графике сгенерированные данные.

В качестве координат точек передадим первый и второй столбец массива Х, для указания цвета точки (параметр с) используем метку класса из массива у

|  |
| --- |
| plt.scatter (X[:,0], X[:,1], c=y)  plt.show |



Видно, что объекты двух классов пересекаются между собой.

Разобьем выборку на обучающее и тестовое множества, используя функцию train\_test\_split. В качестве аргументов передаем массив Х, массив у, test\_size = 0.25 – означает, что на тестовую часть пойдет 25% всей выборки, соответственно, на обучающую – 75%, а также указываем, что разбиение будет случайным, но воспроизводимым (random\_state = 1). Если параметр random\_state = None, то разбиение будет невоспроизводимым.

Функция train\_test\_split записывает результаты разбиения в 4 переменные. Назовем их X\_train, X\_test, y\_train, y\_test. В первую и вторую переменную будут записаны координаты объектов из обучающей и тестовой выборки соответственно, а в третью и четвертую – метки классов объектов из обучающей и экзаменационной выборки соответственно:

|  |
| --- |
| X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 1, ) |

Таким образом, в переменной X\_train лежат координаты, а в y\_train –метки классов соответствующих объектов из тестовой выборки. Эти переменные будут использоваться в дальнейшем для обучения модели.

X\_test, y\_test – соответственно координаты и метки классов объектов тестовой выборки. Эти переменные мы будем использовать для оценки точности модели.

**Обучение модели и классификация**

Для обучения модели и последующей классификации с использованием модулей библиотеки scikit-learn используется довольно стандартная процедура. Разберем на примере обучение и классификацию данных методом к-ближайших соседей:

1. Импортировать требуемый модуль, если этого не было сделано ранее

|  |
| --- |
| from sklearn.neighbors import KNeighborsClassifier |

1. Создать переменную - модель классификатора, указав при необходимости параметры классификации. В нашем случае мы задаем два параметра – количество ближайших соседей = 1 и евклидову метрику.

|  |
| --- |
| knn = KNeighborsClassifier(n\_neighbors=1, metric = ‘euclidean’) |

Для большинства классификаторов, если не задавать никаких параметров, они будут выбраны по умолчанию. Список доступных параметров можно посмотреть в документации, в нашем случае - <http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>

Посмотреть доступные метрики расстояний также можно в документации на DistanceMetric: <http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.DistanceMetric.html>

1. Обучить модель, используя метод fit(), передав в него координаты объектов и метки классов обучающей выборки

|  |
| --- |
| knn.fit(X\_train, y\_train) |

1. Оценить качество модели используя метод predict()и тестовую выборку.

|  |
| --- |
| prediction = knn.predict(X\_test) |

Стоит отметить, что в метод predict() подаются только координаты объектов (X\_test) без истинных меток класса (y\_test). В общем случае, когда модель полностью настроена, в данный метод могут передаваться «боевые» данные – объекты, которые нужно проклассифицировать.

В нашем случае, в переменную prediction метод вернул предсказанные метки классов для каждого объекта из переменной X\_test.

Зная истинные метки классов (переменная y\_test) мы можем оценить, насколько точно работает наша модель.

Самое простое – вывести на экран истинные и предсказанные ответы:

|  |
| --- |
| print ('Prediction and test: ')  print (prediction)  print (y\_test) |

Кроме того, можно оценить матрицу неточностей (confusion matrix) используя функцию confusion\_matrix, и передав в нее истинные и предсказанные ответы:

|  |
| --- |
| print ('Confusion matrix: ')  print (confusion\_matrix(y\_test, prediction)) |

Для оценки аккуратности классификации можно использовать функцию accuracy\_score:

|  |
| --- |
| print ('Accuracy score: ', accuracy\_score(prediction, y\_test)) |

Для оценки показателей полноты-точности и f1-меры воспользуемся функцией classification\_report:

|  |
| --- |
| print(classification\_report(y\_test, prediction)) |

Оценить показатель AUC ROC можно следующим образом:

|  |
| --- |
| from sklearn.metrics import roc\_auc\_score  roc\_auc\_score(y\_test, prediction) |

Кроме того, воспользовавшись функцией plot\_2d\_separator, описанной выше, можно наглядно отобразить на графике область принятия решений по каждому классу:

|  |
| --- |
| plt.xlabel("first feature")  plt.ylabel("second feature")  plot\_2d\_separator(knn, X, fill=True)  plt.scatter(X[:, 0], X[:, 1], c=y, s=70) |

