

Часть 4

K-Nearest Neighbors

К-ближайших соседей

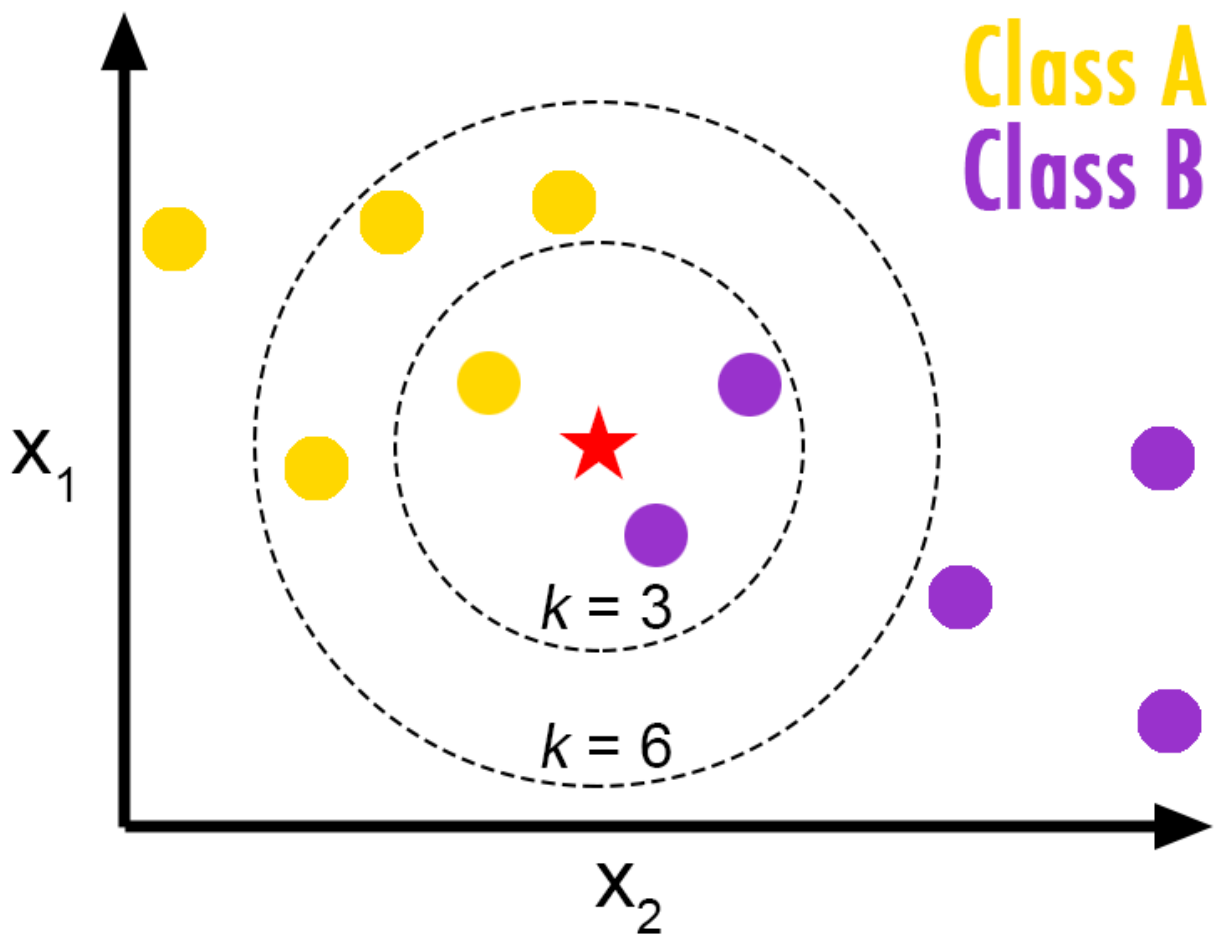
Теоретические сведения

In this Lab you will load a customer dataset, fit the data, and use K-Nearest Neighbors to predict a data point. But what is **K-Nearest Neighbors**?

K-Nearest Neighbors is an algorithm for supervised learning. Where the data is 'trained' with data points corresponding to their classification. Once a point is to be predicted, it takes into account the 'K' nearest points to it to determine it's classification.

В ходе этой работы вы исследуете набор данных о клиентах, обработаете данные и воспользуетесь алгоритмом К-ближайших соседей для прогнозирования состояния новой точки данных. K-Nearest Neighbours — это алгоритм контролируемого обучения, в котором для определения "класса" новой точки учитываются ближайшие к ней К-точек.

Here's an visualization of the K-Nearest Neighbors algorithm.



In this case, we have data points of Class A and B. We want to predict what the star (test data point) is. If we consider a k value of 3 (3 nearest data points) we will obtain a prediction of Class B. Yet if we consider a k value of 6, we will obtain a prediction of Class A.

In this sense, it is important to consider the value of k . But hopefully from this diagram, you should get a sense of what the K-Nearest Neighbors algorithm is. It considers the ' K ' Nearest Neighbors (points) when it predicts the classification of the test point.

В задаче, показанной на рисунке, у нас есть точки данных, принадлежащие классам А и В. Мы хотим определить к какому классу относится новая точка, обозначенная звездой (точка тестовых данных). Если мы рассмотрим значение k , равное 3 (3 ближайшие точки данных), мы получим прогноз класса В. Однако, если мы рассмотрим значение k , равное 6, мы получим прогноз класса А. В этом смысле важно учитывать значение k . Подбор наиболее адекватного значения – задача разработчика / исследователя.

Программа работы

1. [About the dataset](#)
2. [Data Visualization and Analysis](#)
3. [Classification](#)

Lets load required libraries

```
In [1]: import itertools
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter
import pandas as pd
import numpy as np
import matplotlib.ticker as ticker
from sklearn import preprocessing
%matplotlib inline
```

About the dataset

Imagine a telecommunications provider has segmented its customer base by service usage patterns, categorizing the customers into four groups. If demographic data can be used to predict group membership, the company can customize offers for individual prospective customers. It is a classification problem. That is, given the dataset, with predefined labels, we need to build a model to be used to predict class of a new or unknown case.

The example focuses on using demographic data, such as region, age, and marital, to predict usage patterns.

The target field, called **custcat**, has four possible values that correspond to the four customer groups, as follows: 1- Basic Service 2- E-Service 3- Plus Service 4- Total Service

Our objective is to build a classifier, to predict the class of unknown cases. We will use a specific type of classification called K nearest neighbour.

Представьте себе, что поставщик телекоммуникационных услуг сегментировал свою клиентскую базу по шаблонам использования услуг, разделив клиентов на четыре группы. Если демографические данные можно использовать для прогнозирования членства в группе, компания может настроить предложения для отдельных потенциальных клиентов. Это проблема классификации. То есть, учитывая набор данных с предопределенными метками, нам нужно построить модель, которая будет использоваться для прогнозирования класса нового или неизвестного случая. В примере основное внимание уделяется использованию демографических данных, таких как регион, возраст и брак, для прогнозирования моделей использования. Целевое поле, называемое custcat, имеет четыре возможных значения, которые соответствуют четырем группам клиентов, а именно: 1 — базовая услуга 2 — электронная услуга 3 — дополнительная услуга 4 — общая услуга Наша цель — построить классификатор, чтобы предсказать класс неизвестных случаев. Мы будем использовать особый тип классификации, называемый K ближайшим соседом.

Load Data From CSV File

```
In [2]: df = pd.read_csv('teleCust1000t.csv')
df.head()
```

```
Out[2]:    region  tenure  age  marital  address  income  ed  employ  retire  gender  reside  custcat
```

0	2	13	44	1	9	64.0	4	5	0.0	0	2	1
1	3	11	33	1	7	136.0	5	5	0.0	0	6	4
2	3	68	52	1	24	116.0	1	29	0.0	1	2	3
3	2	33	33	0	12	33.0	2	0	0.0	1	1	1
4	2	23	30	1	9	30.0	1	2	0.0	0	4	3

Data Visualization and Analysis

Let's see how many of each class is in our data set

```
In [3]: df['custcat'].value_counts()
```

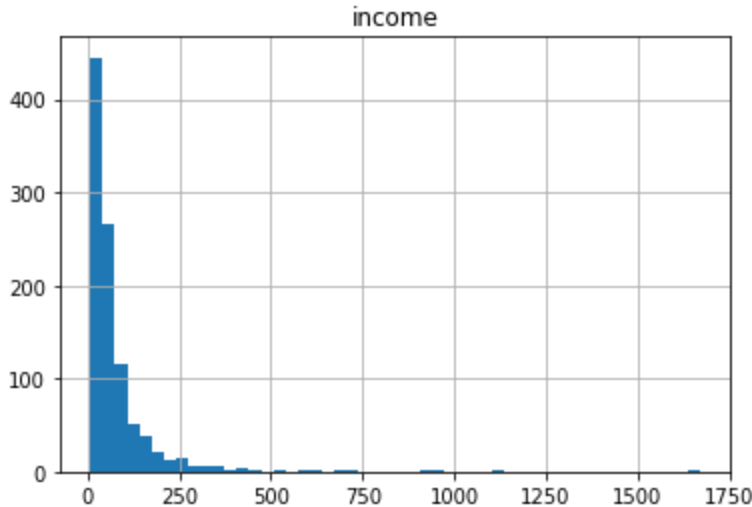
```
Out[3]: 3    281
1    266
4    236
2    217
Name: custcat, dtype: int64
```

281 Plus Service, 266 Basic-service, 236 Total Service, and 217 E-Service customers

You can easily explore your data using visualization techniques:

```
In [4]: df.hist(column='income', bins=50)
```

```
Out[4]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000020D60763708>]],
      dtype=object)
```



Feature set

Lets define feature sets, X:

```
In [5]: df.columns
```

```
Out[5]: Index(['region', 'tenure', 'age', 'marital', 'address', 'income', 'ed',
              'employ', 'retire', 'gender', 'reside', 'custcat'],
              dtype='object')
```

To use scikit-learn library, we have to convert the Pandas data frame to a Numpy array:

```
In [6]: X = df[['region', 'tenure', 'age', 'marital', 'address', 'income', 'ed',
```

```
'employ', 'retire', 'gender', 'reside']] .values #.astype(float)
x[0:5]
```

```
Out[6]: array([[ 2., 13., 44., 1., 9., 64., 4., 5., 0., 0., 2.],
 [ 3., 11., 33., 1., 7., 136., 5., 5., 0., 0., 6.],
 [ 3., 68., 52., 1., 24., 116., 1., 29., 0., 1., 2.],
 [ 2., 33., 33., 0., 12., 33., 2., 0., 0., 1., 1.],
 [ 2., 23., 30., 1., 9., 30., 1., 2., 0., 0., 4.]])
```

What are our labels?

```
In [7]: y = df['custcat'].values
y[0:5]
```

```
Out[7]: array([1, 4, 3, 1, 3], dtype=int64)
```

Normalize Data

Data Standardization give data zero mean and unit variance, it is good practice, especially for algorithms such as KNN which is based on distance of cases:

```
In [8]: X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
X[0:5]
```

```
Out[8]: array([[ -0.02696767, -1.055125 ,  0.18450456,  1.0100505 , -0.25303431,
 -0.12650641,  1.0877526 , -0.5941226 , -0.22207644, -1.03459817,
 -0.23065004],
 [ 1.19883553, -1.14880563, -0.69181243,  1.0100505 , -0.4514148 ,
  0.54644972,  1.9062271 , -0.5941226 , -0.22207644, -1.03459817,
  2.55666158],
 [ 1.19883553,  1.52109247,  0.82182601,  1.0100505 ,  1.23481934,
  0.35951747, -1.36767088,  1.78752803, -0.22207644,  0.96655883,
 -0.23065004],
 [-0.02696767, -0.11831864, -0.69181243, -0.9900495 ,  0.04453642,
 -0.41625141, -0.54919639, -1.09029981, -0.22207644,  0.96655883,
 -0.92747794],
 [-0.02696767, -0.58672182, -0.93080797,  1.0100505 , -0.25303431,
 -0.44429125, -1.36767088, -0.89182893, -0.22207644, -1.03459817,
  1.16300577]])
```

Train Test Split

Out of Sample Accuracy is the percentage of correct predictions that the model makes on data that that the model has NOT been trained on. Doing a train and test on the same dataset will most likely have low out-of-sample accuracy, due to the likelihood of being over-fit.

It is important that our models have a high, out-of-sample accuracy, because the purpose of any model, of course, is to make correct predictions on unknown data. So how can we improve out-of-sample accuracy? One way is to use an evaluation approach called Train/Test Split. Train/Test Split involves splitting the dataset into training and testing sets respectively, which are mutually exclusive. After which, you train with the training set and test with the testing set.

This will provide a more accurate evaluation on out-of-sample accuracy because the testing dataset is not part of the dataset that have been used to train the data. It is more realistic for real world problems.

Точность вне выборки — это процент правильных прогнозов, которые модель обеспечивает на данных, которые НЕ использовались для обучения модели. Выполнение обучения и тестирования

на одном и том же наборе данных, скорее всего, будет иметь низкую точность вне выборки из-за вероятности избыточного соответствия. Важно, чтобы модель имела высокую вневыборочную точность, потому что цель любой модели, конечно, состоит в том, чтобы делать правильные прогнозы на неизвестных данных. Один из способов улучшить точность вне выборки — использовать подход к оценке под названием Train/Test Split. Train/Test Split включает в себя разделение набора данных на наборы для обучения и тестирования соответственно, которые являются взаимоисключающими. После этого вы тренируетесь с тренировочным набором и тестируете с помощью тестового набора. Это обеспечит более точную оценку точности вне выборки, поскольку набор данных тестирования не является частью набора данных, который использовался для обучения данных.

```
In [9]: from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=4)

print ('Train set:', X_train.shape, y_train.shape)
print ('Test set:', X_test.shape, y_test.shape)
```

```
Train set: (800, 11) (800,)
Test set: (200, 11) (200,)
```

Classification

K nearest neighbor (KNN)

Import library

Classifier implementing the k-nearest neighbors vote.

```
In [10]: from sklearn.neighbors import KNeighborsClassifier
```

Training

Lets start the algorithm with k=4 for now:

```
In [11]: k = 4
#Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh
```

```
Out[11]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                             metric_params=None, n_jobs=None, n_neighbors=4, p=2,
                             weights='uniform')
```

Predicting

we can use the model to predict the test set:

```
In [12]: yhat = neigh.predict(X_test)
yhat[0:5]
```

```
Out[12]: array([1, 1, 3, 2, 4], dtype=int64)
```

Accuracy evaluation

In multilabel classification, **accuracy classification score** is a function that computes subset accuracy. This function is equal to the `jaccard_similarity_score` function. Essentially, it calculates how closely the actual labels and predicted labels are matched in the test set.

```
In [13]: from sklearn import metrics
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))

Train set Accuracy:  0.5475
Test set Accuracy:  0.32
```

Practice

Can you build the model again, but this time with $k=6$?

```
In [14]: # write your code here
k = 6
neigh2 = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
yhat2 = neigh.predict(X_test)

print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh2.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat2))

Train set Accuracy:  0.51625
Test set Accuracy:  0.32
```

Double-click **here** for the solution.

What about other K?

K in KNN, is the number of nearest neighbors to examine. It is supposed to be specified by the User. So, how can we choose right value for K? The general solution is to reserve a part of your data for testing the accuracy of the model. Then chose $k=1$, use the training part for modeling, and calculate the accuracy of prediction using all samples in your test set. Repeat this process, increasing the k , and see which k is the best for your model.

We can calculate the accuracy of KNN for different K_s .

```
In [15]: Ks = 10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
ConfustionMx = [];
for n in range(1,Ks):

    #Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat=neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)

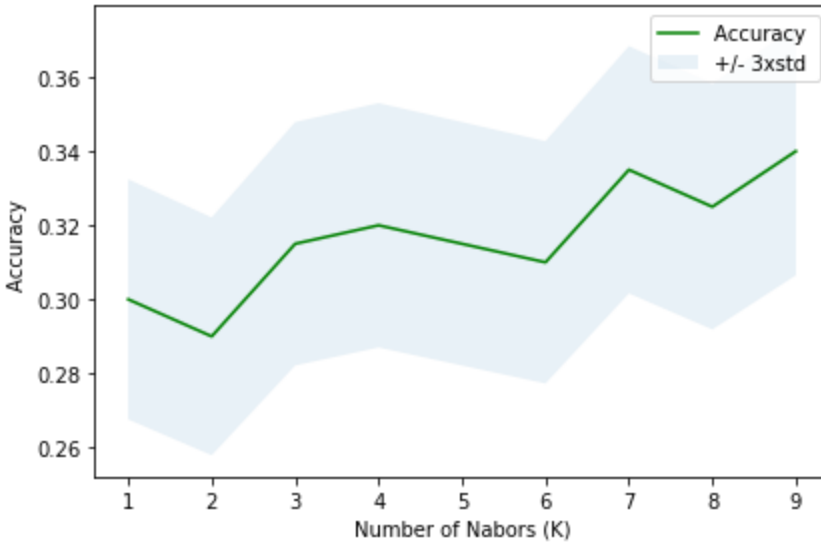
    std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])

mean_acc

Out[15]: array([0.3 , 0.29 , 0.315, 0.32 , 0.315, 0.31 , 0.335, 0.325, 0.34 ])
```

Plot model accuracy for Different number of Neighbors

```
In [16]: plt.plot(range(1,Ks),mean_acc,'g')
plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.10)
plt.legend(('Accuracy ', '+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Nabors (K)')
plt.tight_layout()
plt.show()
```



```
In [17]: print("The best accuracy was with", mean_acc.max(), "with k=", mean_acc.argmax()+1)

The best accuracy was with 0.34 with k= 9
```