

```
import numpy as np #linear algebra library of Python
import pandas as pd # build on top of numpy for data analysis, data manipulation and d
import matplotlib.pyplot as plt #plotting library of Python
```

Now let's mount Google drive so that we can upload the diabetes.csv file. You can find the code in the 'Code s

```
from google.colab import drive
drive.mount('/content/gdrive')
```



First thing that we do is take a look at the shape of the dataframe (df.shape) and take a look at first 5 lines th

```
df=pd.read_csv('/content/gdrive/My Drive/Colab Notebooks/diabetes.csv') #import file f
df.head() #shows first 5 lines including column namesdf.shape # number of rows and col
```



```
df.shape # provides # rows and # columns of the dataframe df - 768 rows and 9 columns
```



Now we will assess if the dataset has the same proportion of diabetes vs. non-diabetes cases. At the same t  
dataset we note that woman #2 has a skin thickness of zero and this is not realistic. It leads us to believe tha  
was available. This does not apply to columns columns 1 and 9 for obvious reasons.

We use a trick to count the non-zero values of the columns. We convert the data type of the dataframe df to t  
values to false=0 and all other entries to true=1 . We subsequently add up all True entries per column.

```
df.astype(bool).sum(axis=0) # counts the number of non-zeros for each column while act
```



The dataframe is unbalanced as we have 268 ones (diabetes) and thus 500 zeros (no diabetes).

The easiest option could be to eliminate all those patients with zero values, but in this way we would eliminat

Another option is to calculate the median value for a specific column and substitute the zero values for the co

```
median_BMI=df['BMI'].median()
df['BMI']=df['BMI'].replace(to_replace=0, value=median_BMI)

median_BloodPressure=df['BloodPressure'].median()
df['BloodPressure']=df['BloodPressure'].replace(to_replace=0, value=median_BloodPressu

median_Glucose=df['Glucose'].median()
df['Glucose']=df['Glucose'].replace(to_replace=0, value=median_Glucose)

median_SkinThickness=df['SkinThickness'].median()
df['SkinThickness']=df['SkinThickness'].replace(to_replace=0, value=median_SkinThickne
```

```
median_Insulin=df['Insulin'].median()  
df['Insulin']=df['Insulin'].replace(to_replace=0, value=median_Insulin)
```

```
df.head() #shows first 5 lines including column names
```



The skin thickness of woman #2 is now 23 (median of that column)

Let's create numpy arrays, one for the features (X) and one for the label (y)

```
X=df.drop('Outcome', 1).values #drop 'Outcome' column but you keep the index column  
y=df['Outcome'].values
```

We import the train\_test\_split function from sklearn to split the arrays or matrices into random train and test sets

Parameters:

test\_size : in our case 20% (default=0.25)

random\_state: is basically used for reproducing your problem the same every time it is run. If you do not use random\_state, the split you might get a different set of train and test data points and will not help you in debugging in case your results number does not matter

stratify : array-like or None (default=None) If the number of values belonging to each class are unbalanced, use stratify, basically asking the model to take the training and test set such that the class proportion is same as of the whole dataset

```
from sklearn.model_selection import train_test_split #method to split training and test data  
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.2, random_state=42)
```

```
print(X_train)
```



The last preprocessing step is feature normalization transforming the data to have mean=0 and standard deviation=1. As the similarity measure in KNN we should not forget this step. Without using this step the accuracy would be low.

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

```
print(X_train)
```



Now we are ready to use the KNN algorithm

```
from sklearn.neighbors import KNeighborsClassifier # we import the K-Nearest Neighbor classifier
neighbors=np.arange(1,30) #we will try different k - default step size is 1 - returns an array of k values

train_accuracy=np.empty(len(neighbors)) # creates an array that will be used for storing training accuracy
test_accuracy=np.empty(len(neighbors)) # creates an array that will be used for storing test accuracy
```

```
print(neighbors)
```



A lot of times when dealing with iterators, we also get a need to keep a count of iterations. Python eases the task with enumerate() for this task. Enumerate() method adds a counter to an iterable and returns it in a form of enumerate object. This object can be directly in 'for loops' or be converted into a list of tuples using list() method.

```
for i,k in enumerate(neighbors): #k goes from 1 to 19 and i is the counter
    knn=KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    train_accuracy[i]=knn.score(X_train, y_train)
    test_accuracy[i]=knn.score(X_test, y_test)
```

```
plt.title('k-NN Varying number of neighbors')
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.plot(neighbors, train_accuracy, label='Training Accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()
```



We get maximum testing accuracy for k=26, so we will setup a KNN classifier with hyperparameter k=26 (we of a test point). the high k reflects the fact that the labels are mixed and that there is no clear divide. In the co optimal k but that is not correct. This will also slightly affect the confusion matrix (recall and precision)

```
knn=KNeighborsClassifier(n_neighbors=26)
knn.fit(X_train, y_train)
knn.score(X_test, y_test) #score method represents accuracy
```



Now we will look at other classification KPIs that we discussed in our lessons: Confusion Matrix, ROC, AUC, F

```
from sklearn.metrics import confusion_matrix
y_pred=knn.predict(X_test)
confusion_matrix(y_test,y_pred)
```



Classifier not so good: true positives=30, true negatives=90, false positives=10 and false negatives=24. We could choose a different threshold in this case as we would tell a woman that she is not diabetic whereas she actually is diabetic. One option to reduce the amount of False Positives (FPs) but this will increase the amount of False Negatives (FNs) as we have seen in lesson 3. Recall in our case is  $TP/(TP+FP)=75\%$  and is not ideal as well.

### ROC (Receiver Operating Characteristic) curve

It is a plot of Recall vs. False Positive Rate (FPR) for the different possible thresholds of the classifier. It shows how close the curve follows the left-hand border and then the top border of the ROC space, the more accurate the test. The closer the curve follows the diagonal of the ROC space, the less accurate the test. The area under the curve is a measure of test accuracy.

```
from sklearn.metrics import roc_curve
y_pred_proba=knn.predict_proba(X_test)[:,-1]
fpr, tpr, thresholds=roc_curve(y_test, y_pred_proba)
```

```
plt.plot([0,1], [0,1], 'k--')
plt.plot(fpr, tpr, label='knn')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('knn(Neighbors=8) ROC curve')
plt.show()
```



```
from sklearn.metrics import roc_auc_score #area under the ROC curve
roc_auc_score(y_test, y_pred_proba)
```



We build a KNN classifier with 21 blocks of code!

