DOĞUŞ UNIVERSITY

PERCEPTRONS ALGORITHM AND K-NEAREST NEIGHBOURS ALGORITHM USING PYTHON

Data Mining and Knowledge Discovery (ISE 448)
Computer Engineering, Engineering Department

Adam Janowski
202103001118

08.01.2022

ISTANBUL, 2022

Table of Contents

PERCEPTRON ALGORITHM **2**

THEORY OF PERCEPTRON ALGORITHM 2
NUMERIC QUESTION 4
Manual calculations *4*
Python Code *6*
Data implementation *6*
Algorithm in Python *6*
Output *6*
REAL LIFE QUESTION 7
Import of libraries *7*
Dataset presentation *7*
Data modeling *8*
Measures of effectiveness *8*
Algorithm in Python *10*
Final Outcome *10*

K-NEAREST NEIGHBORS ALGORITHM **11**

THEORY OF K-NEAREST NEIGHBORS 11
NUMERIC QUESTION 12
KNN IN PYTHON 14
Importing libraries *14*
Loading data *14*
Preprocessing data *15*
Elbow method for finding optimal value of K *15*
Measures of accuracy *16*

BIBLIOGRAPHY **17**

Perceptron Algorithm

Theory of Perceptron Algorithm

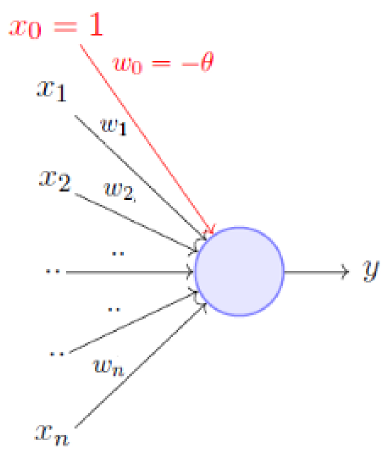
The Perceptron algorithm is a two-class (binary) classification machine learning algorithm. It consists of a single node or neuron that takes a row of data as input and predicts a class label. This is achieved by calculating the weighted sum of the inputs and a bias (set to 1). The weighted sum of the input of the model is called the activation.

1. **Activation** = Weights * Inputs + Bias

If the activation is above 0.0, the model will output 1.0; otherwise, it will output 0.0.

- **Predict 1**: If Activation > 0.0
- **Predict 0**: If Activation <= 0.0

Given that the inputs are multiplied by model coefficients, like linear regression and logistic regression, it is good practice to normalize or standardize data prior to using the model.



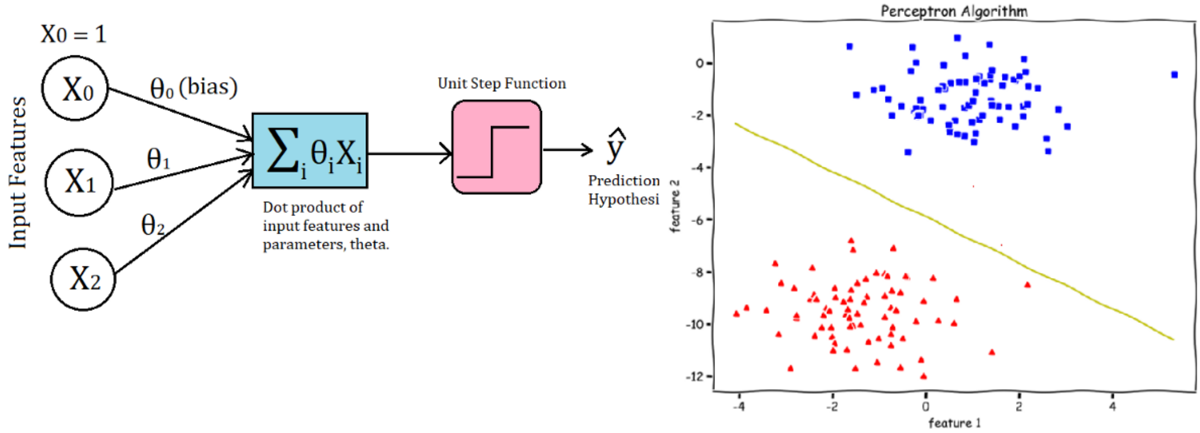
A more accepted convention,

$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$

$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

$$where, \quad x_0 = 1 \quad and \quad w_0 = -\theta$$

The Perceptron is a linear classification algorithm. This means that it learns a decision boundary that separates two classes using a line (called a hyperplane) in the feature space. As such, it is appropriate for those problems where the classes can be separated well by a line or linear model, referred to as linearly separable.



The coefficients of the model are referred to as input weights and are trained using the stochastic gradient descent optimization algorithm.

Examples from the training dataset are shown to the model one at a time, the model makes a prediction, and error is calculated. The weights of the model are then updated to reduce the errors for the example. This is called the Perceptron update rule. This process is repeated for all examples in the training dataset, called an epoch. This process of updating the model using examples is then repeated for many epochs.
Model weights are updated with a small proportion of the error each batch, and the proportion is controlled by a hyperparameter called the learning rate, typically set to a small value. This is to ensure learning does not occur too quickly, resulting in a possibly lower skill model, referred to as premature convergence of the optimization (search) procedure for the model weights.

- weights(t + 1) = weights(t) + learning_rate * (expected_i – predicted_) * input_i

Training is stopped when the error made by the model falls to a low level or no longer improves, or a maximum number of epochs is performed.

The initial values for the model weights are set to small random values. Additionally, the training dataset is shuffled prior to each training epoch. This is by design to accelerate and improve the model training process. Because of this, the learning algorithm is stochastic and may achieve different results each time it is run. As such, it is good practice to summarize the performance of the algorithm on a dataset using repeated evaluation and reporting the mean classification accuracy.

The learning rate and number of training epochs are hyperparameters of the algorithm that can be set using heuristics or hyperparameter tuning.

Numeric Question

Manual calculations

X_1	X_2	Y
0	0	0
0	1	0
1	0	0
1	1	1

x_1 = 0.9 and x_2 = 0.9, x_b = 0.5 {b=bias, b=1}

Round 1:

First instance x1 = 0 and x2 = 0:

Sum unit: $\Sigma = x_1 * w_1 + x_2 * w_2 + b * w_b = 0 * 0.9 + 0 * 0.9 + 1 * 0{,}5 = \mathbf{0{,}5}$

Activation unit checks sum unit is greater than a threshold (which would be 0.5). If this rule is satisfied, then it is fired and the unit will return 1, otherwise it will return 0. Sum unit was 0,5 for the 1st instance, so activation unit would return 0 because it not more than 0.5.

Second instance x1 = 0 and x2 = 1:

Sum unit: $\Sigma = x_1 * w_1 + x_2 * w_2 + b * w_b = 0 * 0.9 + 1 * 0.9 + 1 * 0{,}5 = \mathbf{1{,}4}$

Activation unit will return 1 because sum unit is greater than 0.5. However, output of this instance should be 0, therefore this instance is not predicted correctly and that's why, weights should be updated based on the error. To calculate new weights, error (ε = actual–prediction = 0–1 = -1) times learning rate value should be added to the weights. Learning rate would be α = 0.5.

$w_1 = w_1 + \alpha * \varepsilon = 0.9 + 0.5 * (-1) = 0.9 - 0.5 = 0.4$

$w_2 = w_2 + \alpha * \varepsilon = 0.9 + 0.5 * (-1) = 0.9 - 0.5 = 0.4$

$w_b = w_b + \alpha * \varepsilon = 0.5 + 0.5 * (-1) = 0.5 - 0.5 = 0$

Third instance x1 = 1 and x2 = 0:

Sum unit: $\Sigma = x_1 * w_1 + x_2 * w_2 + b * w_b = 1 * 0.4 + 0 * 0.4 + 1 * 0 =$ **0.4**

Activation unit will return 0, because output of the sum unit is 0.5 and it is less than 0.5. Weights won't be updated this time.

Fourth instance x1 = 1 and x2 = 1:

Sum unit: $\Sigma = x_1 * w_1 + x_2 * w_2 + b * w_b = 1 * 0.4 + 1 * 0.4 + 1 * 0 =$ **0.8**

Activation unit will return 1, because output of the sum unit is 0.8, what is greater than the threshold value. As the 4th instance is predicted correctly, there is no need to update anything.

Round 2:

First instance x1 = 0 and x2 = 0:

Sum unit: $\Sigma = x_1 * w_1 + x_2 * w_2 + b * w_b = 0 * 0.4 + 0 * 0.4 + 1 * 0 = 0$

Activation unit will return 0 because sum unit is 0.4, what is less than the threshold value 0.5. The instance is classified correctly and there is no need to update weights.

Second instance x1 = 0 and x2 = 1:

Sum unit: $\Sigma = x_1 * w_1 + x_2 * w_2 + b * w_b = 0 * 0.4 + 1 * 0.4 + 1 * 0 =$ **0.4**

Activation unit will return 0 because sum unit is less than the threshold 0.5. Its output should be 0 as well, what means, that it is classified correctly no update of weights is needed.

Python Code

The same calculation will be done in Python to prove correctness of it. The algorithm will be modeled manually to emphasize way of working of the algorithm.

Data implementation

Firstly, the data has to be implemented in numpy array, and all weights need to be initialized.

```
atributes = np.array([ [0, 0], [0, 1], [1, 0], [1, 1]])
labels = np.array([0, 0, 0, 1])

# 'b' stands for bias
# 'w' define weight of the perceptron,
# 'threshold' define a umbral,
# 'alpha' is a learning rate,
# 'epoch' is a number of process to train the model.

b=1
w = [+0.9, +0.9] #initial random values for weights
wb = 0.5
threshold = 0.5
alpha = 0.5 #learning rate
epoch = 1000 #learning time

print("learning rate: ", alpha,", threshold: ", threshold)
```

```
learning rate:  0.5 , threshold:  0.5
```

Algorithm in Python

Secondly, it is needed to determine the operation of the algorithm. The perceptron algorithm will operate on well-defined loops, more fully explained below.

```
# The main code where we train the model (model in this case is fit the weight values).
for i in range(0, epoch):
    print("epoch ", i+1)
    global_delta = 0 #this variable is used to terminate the for loop if learning completed in early epoch
    for j in range(len(atributes)):
        actual = labels[j]
        sum = atributes[j][0]*w[0] + atributes[j][1]*w[1] + b*wb

        if sum > threshold: #then fire
            predicted = 1
        else: #do not fire
            predicted = 0

        delta = actual - predicted
        global_delta = global_delta + abs(delta)

        #update weights with respect to the error
        for k in range(0, 2):
            w[k] = w[k] + delta * alpha
        wb = wb + delta * alpha

        print(atributes[j][0]," ", operator, " ", atributes[j][1], " -> actual: ", actual, ", predicted: ", predicted,

    if global_delta == 0:
        break

    print("----------------------------")
```

Output

```
epoch  1
0   and   0  -> actual:  0 , predicted:  0  (w:  0.9 0.9 wb:  0.5 )
0   and   1  -> actual:  0 , predicted:  1  (w:  0.4 0.4 wb:  0.0 )
1   and   0  -> actual:  0 , predicted:  0  (w:  0.4 0.4 wb:  0.0 )
1   and   1  -> actual:  1 , predicted:  1  (w:  0.4 0.4 wb:  0.0 )
------------------------------
epoch  2
0   and   0  -> actual:  0 , predicted:  0  (w:  0.4 0.4 wb:  0.0 )
0   and   1  -> actual:  0 , predicted:  0  (w:  0.4 0.4 wb:  0.0 )
1   and   0  -> actual:  0 , predicted:  0  (w:  0.4 0.4 wb:  0.0 )
1   and   1  -> actual:  1 , predicted:  1  (w:  0.4 0.4 wb:  0.0 )
```

Real life question

Import of libraries

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from sklearn.metrics import classification_report
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot
df = pd.read_csv ('/Users/ja/Downloads/creditcard.csv')
```

Dataset presentation

The dataset contains transactions made by credit cards in September 2013 by European cardholders. It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, the original features cannot be provided in detail. Feature 'Class' is the response variable, and it takes value 1 in case of fraud and 0 otherwise.

```python
#Data preview
print(df.head())
```

```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  ...       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
```

```python
print(df.describe())
```

```
                Time            V1            V2            V3            V4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean    94813.859575  3.918649e-15  5.682686e-16 -8.761736e-15  2.811118e-15
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01

                 V5            V6            V7            V8            V9  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean  -1.552103e-15  2.040130e-15 -1.698953e-15 -1.893285e-16 -3.147640e-15
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01

       ...           V21           V22           V23           V24  \
count  ...  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   ...  1.473120e-16  8.042109e-16  5.282512e-16  4.456271e-15
std    ...  7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min    ... -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%    ... -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%    ... -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%    ...  1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max    ...  2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                V25           V26           V27           V28         Amount  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  284807.000000
mean   1.426896e-15  1.701640e-15 -3.662252e-16 -1.217809e-16      88.349619
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01     250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01       0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02       5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02      22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02      77.165000
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01   25691.160000

               Class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 31 columns]
```

```
yes = df[df['Class']==1]['Class'].count()
total = df['Class'].count()
print(f'Fraud values in dataset is {yes} from {total}, what is only {round((yes/total)*100,2)}% of all dataset.')
print('That shows how dataset is highly inbalances')
```

```
Fraud values in dataset is 492 from 284807, what is only 0.17% of all dataset.
That shows how dataset is highly inbalances
```

Data modeling

```
#Split the data into a learn and a testset:
datasets = train_test_split(df.iloc[:,:-1],
                            df.iloc[:,-1],
                            test_size=0.1)

train_data, test_data, train_labels, test_labels = datasets

#Create a Perceptron instance and fit the training data:
p=Perceptron()
p.fit(train_data, train_labels)
```

```
#Classification of train data
print(classification_report(p.predict(train_data), train_labels))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    256202
           1       0.00      0.01      0.00       124

    accuracy                           1.00    256326
   macro avg       0.50      0.50      0.50    256326
weighted avg       1.00      1.00      1.00    256326
```

```
#Classification of test data
print(classification_report(p.predict(test_data), test_labels))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     28466
           1       0.00      0.00      0.00        15

    accuracy                           1.00     28481
   macro avg       0.50      0.50      0.50     28481
weighted avg       1.00      1.00      1.00     28481
```

Measures of effectiveness

Precision

Precision can be seen as a measure of a classifier's exactness. For each class, it is defined as the ratio of true positives to the sum of true and false positives

Recall

Recall is a measure of the classifier's completeness; the ability of a classifier to correctly find all positive instances. For each class, it is defined as the ratio of true positives to the sum of true positives and false negatives.

F1 score

The F1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0.

Support

Support is the number of actual occurrences of the class in the specified dataset. Imbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier and could indicate the need for stratified sampling or rebalancing

However, given the class imbalance ratio, it is recommended to do measuring the accuracy using the Area Under the Precision-Recall Curve (AUPRC). Confusion matrix accuracy is not meaningful for unbalanced classification.

AUC - ROC curve

Area Under the Curve" (AUC) of "Receiver Characteristic Operator" (ROC). Is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1. By analogy, the Higher the AUC, the better the model is at distinguishing between patients with the disease and no disease. The ROC curve is plotted with TPR against the FPR where TPR is on the y-axis and FPR is on the x-axis.

Precision-Recall (PR) Curve

A PR curve is simply a graph with Precision values on the y-axis and Recall values on the x-axis. In other words, the PR curve contains TP/(TP+FN) on the y-axis and TP/(TP+FP) on the x-axis.

It is important to note that Precision is also called the Positive Predictive Value (PPV). Recall is also called Sensitivity, Hit Rate or True Positive Rate (TPR). It is desired that the algorithm should have both high precision, and high recall. However, most machine learning algorithms often involve a trade-off between the two. A good PR curve has greater AUC (area under curve).

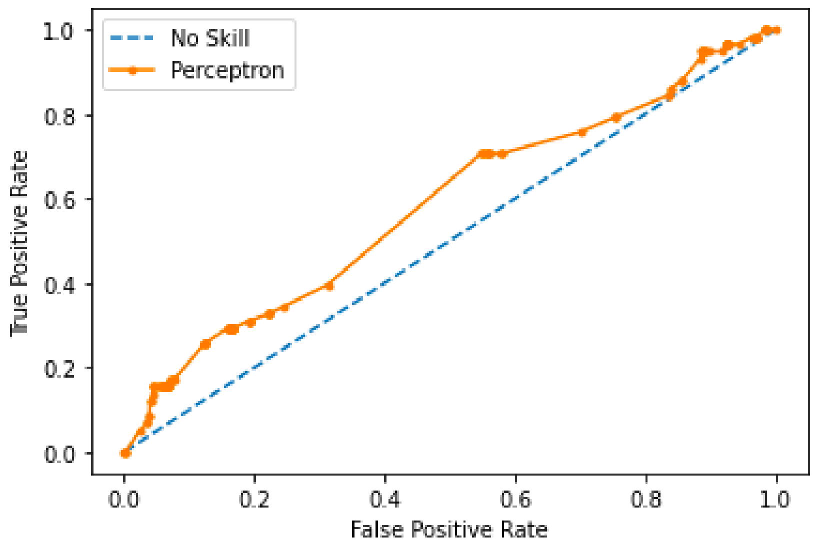
Algorithm in Python

```
# generate a no skill prediction (majority class)
ns_probs = [0 for _ in range(len(test_labels))]
# predict probabilities
clf_isotonic = CalibratedClassifierCV(p, cv=10, method='isotonic')
clf_isotonic.fit(train_data, train_labels)

lr_probs = clf_isotonic.predict_proba(test_data)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
# calculate scores
ns_auc = roc_auc_score(test_labels, ns_probs)
lr_auc = roc_auc_score(test_labels, lr_probs)
# summarize scores
print('No Skill: ROC AUC=%.3f' % (ns_auc))
print('Perceptron: ROC AUC=%.3f' % (lr_auc))
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(test_labels, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(test_labels, lr_probs)
# plot the roc curve for the model
pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
pyplot.plot(lr_fpr, lr_tpr, marker='.', label='Perceptron')
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
# show the legend
pyplot.legend()
# show the plot
pyplot.show()
```

Final Outcome

```
No Skill: ROC AUC=0.500
Perceptron: ROC AUC=0.584
```



K-Nearest Neighbors Algorithm

Theory of K-Nearest Neighbors

K Nearest Neighbors is a simple algorithm that stores all the available cases and classifies the new data or case based on a similarity measure. It is mostly used to classifies a data point based on how its neighbors are classified.

'k' in KNN is a parameter that refers to the number of nearest neighbors to include in the majority of the voting process. Choosing the right value of K is a process called parameter tuning and is important for better accuracy. Finding the value of k is not easy. It is needed to find out the best value of k by trial-and-error process and assuming that training data is unknown. Choosing smaller values for K can be noisy and will have a higher influence on the result. On the other hand, larger values of K will have smoother decision boundaries which mean lower variance but increased bias. Also, computationally expensive. In general, practice, choosing the value of k is k = sqrt(N) where N stands for the number of samples in your training dataset. It is better to keep the value of k odd number, in order to avoid confusion between two classes of data.

K-nearest neighbor algorithm essentially boils down to forming a majority vote between the K most similar instances to a given "unseen" observation. Similarity is defined according to a distance metric between two data points. A popular one is the Euclidean distance method showed below, however there are Other methods are Manhattan, Minkowski, and Hamming distance methods. For categorical variables, the hamming distance must be used.

$$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Pros of KNN:

- Simple to implement
- Flexible to feature/distance choices
- Naturally handles multi-class cases
- Can do well in practice with enough representative data

Cons of KNN:

- Need to determine the value of parameter K (number of nearest neighbors)
- Computation cost is quite high because we need to compute the distance of each query instance to all training samples.
- Storage of data
- Must know we have a meaningful distance function.

Numeric Question

There is given height, weight and T-shirt size of some customers and it is needed to predict the T-shirt size of a new customer given only height and weight information. The new customer height is 161, and weight is 61. Data including height, weight and T-shirt size information is shown below.

Height	Weight	T Shirt Size
158	58	M
158	59	M
158	63	M
160	59	M
160	60	M
163	60	M
163	61	M
160	64	L
163	64	L
165	61	L
165	62	L
165	65	L
168	62	L
168	63	L
168	66	L
170	63	L
170	64	L
170	68	L

The next step is to find distance measure which will be used as a similarity between new sample and training cases and then finds the k-closest customers to new customer in terms of height and weight. Euclidean distance between first observation and new observation is as follows.

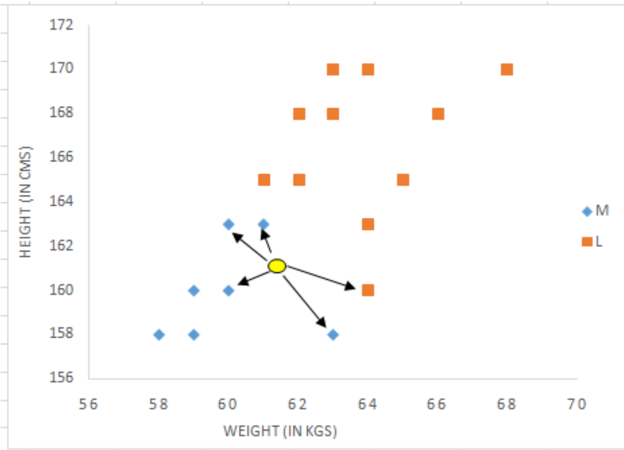
=SQRT((161-158)^2+(61-58)^2)

Similarly, distances of all the training cases with new case will be calculated and the rank in terms of distance will be given. The smallest distance value will be ranked 1 and considered as nearest neighbor. Let k be 5. Then the algorithm searches for the 5 customers closest to new case (most similar in terms of attributes) and see what categories those 5 customers were in. If 4 of them had 'Medium T shirt sizes' and 1 had 'Large T shirt size' then your best guess for test case is 'Medium T shirt. See the calculation shown in the snapshot below.

Height	Weight	T Shirt Size	Distance
158	58	M	4,24
158	59	M	3,61
158	63	M	3,61
160	59	M	2,24
160	60	M	1,41
163	60	M	2,24
163	61	M	2,00
160	64	L	3,16
163	64	L	3,61
165	61	L	4,00
165	62	L	4,12
165	65	L	5,66
168	62	L	7,07
168	63	L	7,28
168	66	L	8,60
170	63	L	9,22
170	64	L	9,49
170	68	L	11,40

From the snapshot it is visible that in 5 the closets neighbors, 4 of them has size M and only one L.

In the graph below, binary dependent variable (T-shirt size) is displayed in blue and orange color. 'Medium T-shirt size' is in blue color and 'Large T-shirt size' in orange color. New customer information is exhibited in yellow circle. Four blue highlighted data points and one orange highlighted data point are close to yellow circle. so the prediction for the new case is blue highlighted data point which is Medium T-shirt size.



KNN in Python

Importing libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Loading data

```
df = pd.read_csv('../input/adult-income-dataset/adult.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   age              48842 non-null  int64
 1   workclass        48842 non-null  object
 2   fnlwgt           48842 non-null  int64
 3   education        48842 non-null  object
 4   educational-num  48842 non-null  int64
 5   marital-status   48842 non-null  object
 6   occupation       48842 non-null  object
 7   relationship     48842 non-null  object
 8   race             48842 non-null  object
 9   gender           48842 non-null  object
 10  capital-gain     48842 non-null  int64
 11  capital-loss     48842 non-null  int64
 12  hours-per-week   48842 non-null  int64
 13  native-country   48842 non-null  object
 14  income           48842 non-null  object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
```

Preprocessing data

```
# Change income column into 0's and 1's
df.income = [1 if each=='>50K' else 0 for each in df.income]
# Create 2 sep data frames one with numerical data, other with categorical data
df_cat = df.select_dtypes(include='object')
df_nums = df.select_dtypes(exclude='object')
# Normalizing numerical data
for i in ['age', 'fnlwgt', 'educational-num', 'capital-gain', 'capital-loss','hours-per-week']:
    df_nums[i] = (df_nums[i]-np.mean(df_nums[i]))/(np.std(df_nums[i]))
# Create dummy variables for different categories
df_cat = df_cat.replace(to_replace ="?",value ="Private")
df_cat = pd.get_dummies(df_cat)
# concatenate both the data frames into final data frame
df = pd.concat([df_nums,df_cat], axis=1)
```

Elbow method for finding optimal value of K

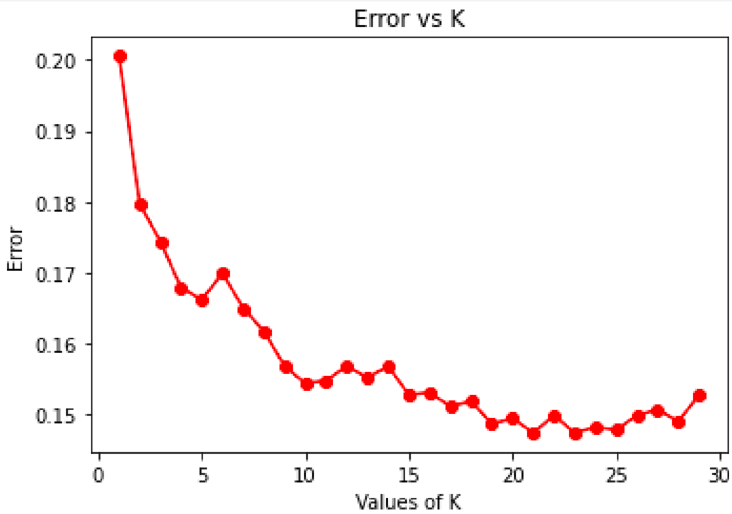
```
from sklearn.model_selection import train_test_split
x = df.drop('income',axis=1)
y = df['income']

from sklearn.neighbors import KNeighborsClassifier
x_train, x_test, y_train, y_test  = train_test_split(x,y,test_size=0.05,random_state=101)

error= []

for i in range(1,30):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train,y_train)
    pred = knn.predict(x_test)
    error.append(1-(accuracy_score(y_test,pred)))

plt.plot(range(1,30),error,'r-',marker='o')
plt.xlabel('Values of K')
plt.ylabel('Error')
plt.title('Error vs K')
```



Measures of accuracy

```python
knn = KNeighborsClassifier(n_neighbors=21)
knn.fit(x_train,y_train)
pred = knn.predict(x_test)

from sklearn.metrics import accuracy_score,classification_report,plot_confusion_matrix,confusion_matrix,plot_precision_recall_curve,plot_roc_curve
print(f'accuracy score: {accuracy_score(y_test,pred)}')
print('Classification report: ')
print(classification_report(y_test, pred))
```

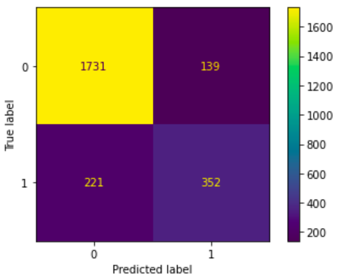
```
accuracy score: 0.852640196479738
Classification report:
              precision    recall  f1-score   support

           0       0.89      0.93      0.91      1870
           1       0.72      0.61      0.66       573

    accuracy                           0.85      2443
   macro avg       0.80      0.77      0.78      2443
weighted avg       0.85      0.85      0.85      2443
```

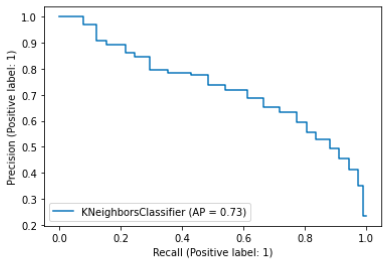
```python
plot_confusion_matrix(knn,x_test,y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f130651edd0>
```



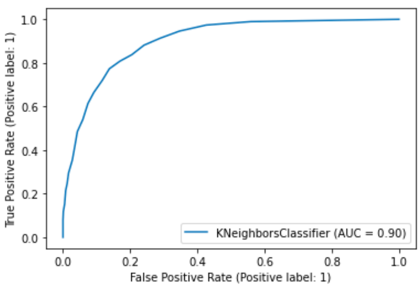

```python
plot_precision_recall_curve(knn,x_test,y_test)
```

```
<sklearn.metrics._plot.precision_recall_curve.PrecisionRecallDisplay at 0x7f1326c50f50>
```




```python
plot_roc_curve(knn,x_test,y_test)
```

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f1306334a10>
```



Analyzing the results from the top, can be deducted, that as accuracy score is 85% the model is works correctly. Confusion matrix tells us that True Positive is much higher from True Negative. However, that comes from the fact that the data is imbalanced. Therefore, ROC curve is also plotted, reaching AUC on level of 90% what is highly satisfying. Precision Recall gives slightly lower level of AUC reaching 73%. However, in over all the obtained results show that the model using the KNN method correctly classified the data.

Bibliography

1. https://www.geeksforgeeks.org/k-nearest-neighbor-algorithm-in-python/
2. https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
3. https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761
4. https://www.analyticsvidhya.com/blog/2021/04/simple-understanding-and-implementation-of-knn-algorithm/
5. https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_knn_algorithm_finding_nearest_neighbors.htm
6. https://www.listendata.com/2017/12/k-nearest-neighbor-step-by-step-tutorial.html
7. https://realpython.com/knn-python/
8. https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/
9. https://machinelearningmastery.com/perceptron-algorithm-for-classification-in-python/
10. https://www.sciencedirect.com/topics/computer-science/perceptron-algorithm
11. https://towardsdatascience.com/perceptron-learning-algorithm-d5db0deab975
12. https://en.wikipedia.org/wiki/Perceptron