Systemy Sztucznej Inteligencji - kolokwium #1

Numpy

1. **np.sum(array)** - suma danej listy
2. **np.exp(potęga)** - e do potęgi
3. **np.sqrt()** - pierwiastek kwadratowy
4. **np.argmax()** zwraca indeks maksymalnego elementu tablicy

Pandas

https://www.learndatasci.com/tutorials/python-pandas-tutorial-complete-introduction-for-beginners/

1. **df = pd.read_csv(…)**
2. **list(df.columns)** - array z nazwami kolumn (nie można tego zrobić dla wierszy)
3. **df.iloc[0]** - weź wiersz o indeksie 0
4. **df.select_dtypes(exclude="object")** - zwraca serię (nie dataframe) bez kolumn które mają obiekty, string jest obiektem
5. **df.loc[:, "nazwa_kolumny"]** - weź wszystkie wartości z kolumny "nazwa_kolumny"
6. **df.loc[3:6, "nazwa_kolumny"]** - weź wartości od indeksu 3 do 6 włącznie z kolumny "nazwa_kolumny"
7. **len(df)** - ile jest wierszy
8. **df.at[indeks wiersza, "nazwa_kolumny"]** (odpowiednio int, string) - zwraca pojedyńczy rekord z bazy
9. **df["nazwa_kolumny"]** - zwraca po prostu kolumnę ale nie jako Dataframe tylko Series
10. **df[["nazwa_kolumny_1", "nazwa_kolumny_2"]]** - zwraca kolumny jako Dataframe
11. **df["nazwa_kolumny"]** = array - dokleja lub podmienia kolumnę "nazwa_kolumny" na to co znajduje się w liście array
12. **df.values** - zwraca dwuwymiarowy array wszystkich rekordów, tyle że bez nazw kolumn i wierszy
13. **df.values[index_x][index_y]** - zwraca dokładnie jeden element z arraya df.values który znajduje się pod indeksami
14. **df.sort_values(by="nazwa_kolumny")** - sortuje nasz df po kolumnie rosnąco (jeśli dodamy ascending=0 - malejąco)

⚠️ **Parameters of sort_values():**

by - str or list of str. Name or list of names to sort by.
• if axis is 0 or ‘index’ then by may contain index levels and/or column labels.
• if axis is 1 or ‘columns’ then by may contain column levels and/or index labels.

axis - {0 or ‘index’, 1 or ‘columns’}, default 0. Axis to be sorted.

ascending - bool or list of bool, default True. Sort ascending vs. descending. Specify list for multiple sort orders. If this is a list of bools, must match the length of the by.

inplace - bool, default False. If True, perform operation in-place.

kind - {‘quicksort’, ‘mergesort’, ‘heapsort’, ‘stable’}, default ‘quicksort’. Choice of sorting algorithm. See also `numpy.sort()` for more information. Mergesort and stable are the only stable algorithms. For DataFrames, this option is only applied when sorting on a single column or label.

15. **df[”nazwa_kolumny”].value_counts()** - zwraca statystykę wszystkich wartości które wystąpiły w kolumnie, dodatkowo sortuje tą listę malejąco
16. **df[”nazwa_kolumny”].value_counts()[0]** - zwraca liczbę wystąpień pierwszego rekordu
17. **df[”nazwa_kolumny”].value_counts().index[0]** - zwraca nazwę pierwszego rekordu (tego co wystąpił najczęściej)
18. **df = pd.DataFrame(data)** - tworzy DataFrame (tabelkę) z data
19. **display(df)** - printuje tabelkę
20. **df[”nazwa_kolumny”].idxmax()** - zwraca indeks największej wartości w kolumnie, analogicznie idxmin - najmniejszej wartości
21. **df[”nazwa_kolumny”].unique().tolist()** - zwraca array unikalnych wartości z kolumny
22. **df.columns.tolist()** - zwraca array nazw kolumn
23. **df[df[”nazwa_kolumny”] == “nazwa_która_jest_warunkiem”]** - zwraca dataframe - tylko te wartości w których pod nazwą kolumny znajduje się konkretna nazwa którą chcieliśmy (taki select)

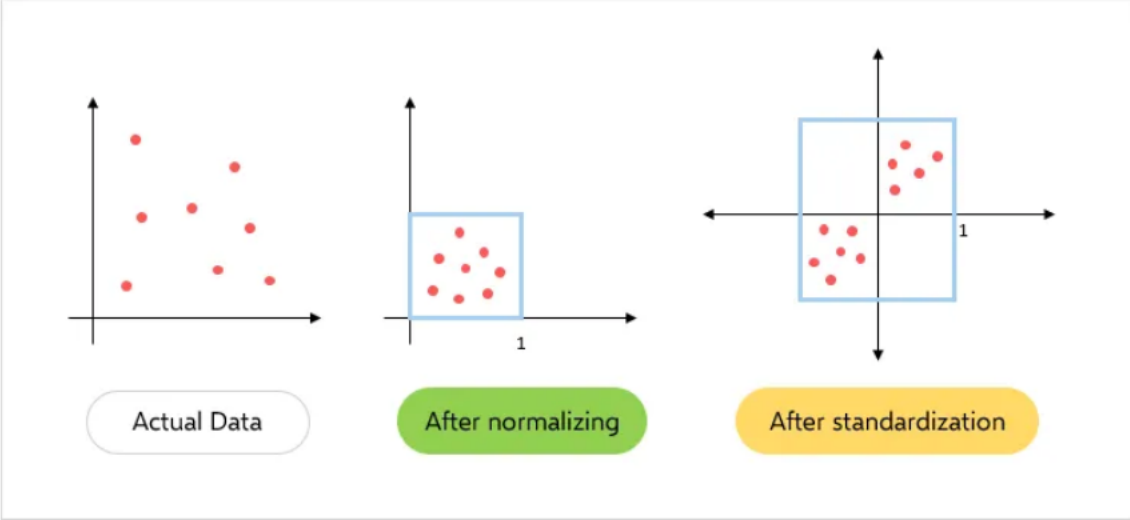
Zbiory

1. treningowy - najczęściej większa część głównego zbioru który służy przede wszystkim do obliczeń
2. testowy - mniejsza część głównego zbioru. To do niego “dokleja się” kolumnę z przewidywaniami
3. walidacyjny - dokładnie ten sam co testowy tylko z kolumną z prawidłową kolumną, którą porównuje się z kolumną “doklejoną” do testowego. Dzięki niemu określamy dokładność przewidywań.

Algorytmy

Shuffle Knutha:

```
def shuffle(x):
        for i in range(len(x)-1, 0, -1):
            j = rn.randint(0, i-1)
            x.iloc[i], x.iloc[j] = x.iloc[j], x.iloc[i]
```



Normalizacja min-max:

$$v'_i = \frac{v_i - min_A}{max_A - min_A}$$

```
def normalization(x):
        values = x.select_dtypes(exclude="object")
        columnNames = values.columns.tolist()

        for column in columnNames:
            data = x.loc[:, column]
            maximum = max(data)
            minimum = min(data)
            for row in range(0, len(x), 1):
                xprim = (x.at[row, column] - minimum)/(maximum - minimum)
                x.at[row, column] = xprim
```

Standaryzacja

$$z = \frac{x - \mu}{\sigma}$$

```
# μ - średnia arytmetyczna
# σ - odchylenie
def standardize(x):
    values = x.select_dtypes(exclude="object")
    columnNames = values.columns.tolist()

    for column in columnNames:
        data = x.loc[:, column]
        mu = sum(data)/len(data)
        summ = 0
        for element in data:
            summ += pow(element-mu, 2)
        sigma = (summ/len(data))**1/2

        for row in range(0, len(x), 1):
            xprim = (x.at[row, column] - sigma)/mu
            x.at[row, column] = xprim
```

Dzielenie zbioru

```
def split(x, k): # k = 0,7 oznacza 70% zbioru treningowego, 30% zbioru testowego
        return x[:int(k*len(x))], x[int(k*len(x)):]
```

Dokładność przewidywań

```
def calculateAccuracy(testingSet, validatingSet):
        truePositiveCounter = 0
        counter = 0
        for row in testingSet.values:
            if row[-1] == validatingSet.values[counter][-1]:
                truePositiveCounter += 1
            counter += 1
        accuracy = truePositiveCounter/len(testingSet)
        print("The accuracy is", int(accuracy*100), "%")
```

KNN

```
def knn(k, validatingSet, trainingSet): # k to ile chcemy sąsiadów
        testingSet = validatingSet.select_dtypes(exclude=["object"]) #kopia zbioru x bez nazw kwiatów oraz nagłówków
        classes = []
        counter = 0
        for row in testingSet.values:
            distances = [] # tworzymy kolumnę distances
            test_vector = testingSet.select_dtypes(exclude=["object"]).iloc[counter] # wybieramy próbkę
            for i in trainingSet.values:
                distances.append(KNN.distance(test_vector, i)) # liczymy odległości między wektorem z setu testowego a każdym z treningoweg
            trainingSet["distance"]=distances # doklejamy kolumnę z dystansami do setu treningowego
            trainingSet=trainingSet.sort_values(by="distance")
            finalSet = trainingSet.head(k) # wybieramy k z najkrótszym dystansem
            classCounts = finalSet["variety"].value_counts() # liczymi liczbę wystąpień danej klasy
            predictedClass = classCounts.index[0] # wybieramy tą z największą liczbą wystąpień
            classes.append(predictedClass) # dorzucamy ją do listy
            counter += 1
        testingSet["class"]=classes
```

Zbiory miękkie

```
data = {
        'product_name': ['tomato', 'paprika', 'carrot', 'lemon'],
        'sweet': [1, 1, 0, 0],
        'sour': [1, 0, 0, 1],
        'spicy': [0, 1, 0, 0],
        'salty': [0, 0, 1, 0],
        'red': [1, 1, 0, 0],
        'frozen': [0, 0, 1, 0],
        'canned': [1, 1, 1, 0],
        'fresh': [1, 1, 1, 1]
        }

df = pd.DataFrame(data)

# liczenie sum w rzędach
def addSumsColumn(parameters, weights):
    sums = []
    for i in parameters.values:
        j = 0
        sum = 0
        for element in i:
            sum += element*weights[j]
            j += 1
        sums.append(sum)
    return sums

# example 1
exampleFirst = [0.5, 0.7, 0, 0.1, 0.5, 0, 0.5, 1]
df["sums"] = addSumsColumn(parameters, exampleFirst)
print("Best vegetable is", df.loc[df["sums"].idxmax(), "product_name"])
```

Naiwny algorytm Bayesa (Gaussowski rozkład prawdopodobieństwa)

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

```
class NaiveBayes:
    @staticmethod
    def mean(column): # średnia z kolumny
        return sum(column)/len(column)

    @staticmethod
    def stv(column, mean): # odchylenie standardowe z kolumny
        tmp=0
        for i in column:
            tmp+=(i-mean)**2
        return np.sqrt(tmp/len(column))

    @staticmethod
    def gaussProbability(x, mu, sigma): # rozłożenie prawdopodobieństwa
        exponent=np.exp(-0.5*((x-mu)/sigma)**2)
        return 1/(sigma*np.sqrt(2*np.pi))*exponent

    @staticmethod
        def classify(sample, x): # bierzemy jedną próbkę i zbiór
            results = []
            for className in x['variety'].unique().tolist(): # dla każdej z trzech klas...
                probability = 1
                columns = x.columns.tolist()[:4] # kolumny tylko z liczbami
                for columnIndex, columnName in enumerate(columns): # przeiteruj po kolumnach
                    data=x[x['variety']==className] # data to ta część tabelki dla której variety == nazwa obecnie rozważanej klasy
                    dataFromOneColumnForOneClass = data.loc[:, columnName] # weź teraz jedną całą kolumnę z data
                    mu = NaiveBayes.mean(dataFromOneColumnForOneClass) # policz z niej średnią
                    sigma = NaiveBayes.stv(dataFromOneColumnForOneClass, mu) # policz z niej odchylenie
                    probability *= NaiveBayes.gaussProbability(sample[columnIndex], mu, sigma) # jak masz już wszystko co potrzeba, policz
                results.append(probability) # jak skończysz wszystkie kolumny dla klasy dodaj do listy wynik
            pred = x['variety'].unique().tolist()[np.argmax(results)] # jak skończysz wszystkie klasy, wybierz tą która ma największe prawd
            return pred

classes = []
for row in testingSet.values: # dla każdego rekordu z setu testowego użyj alg. bayesa
    prediction = NaiveBayes.classify(row, trainingSet)
    classes.append(prediction)
testingSet["class"]=classes # doklej przewidywane klasy do setu testowego
```

Wzory

Metryka euklidesowa

$$= \left(\sum_{i=1}^{n} (p_i - q_i)^2\right)^{1/2}$$

```
def distance(v1, v2): # mamy dwa wektory
        tmp=0
```

```
        for i in range(len(v1)):
            tmp += (v1[i]-v2[i])**2
        return tmp**(1/2)
```

Metryka Manhattan

$$= \sum_{i=1}^{n} |p_i - q_i|$$

Metryka Minkowsky’ego

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}.$$

Wykresy w seaborn dla irysów

1. **sns.pairplot(iris, hue="variety")** - siatka wykresów
2. **iris.plot(kind="scatter", x="sepal.length", y="sepal.width")** - pojedyńczy wykres