

## Inlämningsuppgift i kursen Programmera mera i Python, sommaren 2021

### **Syfte och mål**

Syftet med inlämningsuppgiften är att du ska visa, inte minst för dig själv, att du kan skapa ett Pythonprogram utifrån en definierad problembeskrivning där färdigheter som du fått från denna kurs kommer till användning. Målet med uppgiften är att du efter slutförd uppgift har fått en ökad färdighet i programmering och har fått en insikt i hur man bryter ned en programmeringsuppgift i mindre delar och lösa var del för sig för att sedan sätta ihop de olika delarna till en helhet.

### **Förutsättningar:**

Uppgiften baseras i huvudsak på modulerna NumPy och Matplotlib och består av deluppgifter där varje deluppgift bedöms var för sig. För att erhålla godkänt betyg måste alla deluppgifter vara godkända.

Några enkla regler som ska följas:

- koden ska vara genomtänkt, lättläst och kommenterad.
- skriv varför en kodrad finns och inte vad den gör.
- variabler ska ha meningsfulla namn.
- globala variabler får inte användas.
- du får inte lämna in kod som genererar felmeddelande eller varningar. Då blir det omedelbart retur.
- du får inte lämna in bortkommenterad kod.
- båda programmen ska innehålla en viss felkontroll i enlighet med vad som beskrivs Canvas-modulen 'Mera om Python' under avsnittet felhantering.
- källkoden ska skrivas i utvecklingsverktyget Spyder version 5 eller i ett motsvarande program och sparad i .py-format.

### **Efter att du slutfört uppgiften**

Efter att du färdigutvecklat programmet och utfört de simuleringar som ingår i uppgiften (och att du bedömer att programmet fungerar korrekt) ska källkodsfilen till programmet laddas upp i Canvas. Inlämningsuppgiften examineras genom att läraren kör programmet och kontrollerar resultatet vid val av de olika menyalternativen. För menyalternativ 2 kommer CSV-filer med förväntade resultat att användas vid examinationen.

Spara källkodsfilen under namnet: ***efternamn\_förnamn\_Sxxxxxxx\_inlämning.py*** där Sxxxxxxx är ditt inloggnings-ID.

## Uppgiftsbeskrivning

Inom området Artificiell intelligens (AI) har stora framsteg gjorts under det senaste årtiondet. Deep Learning är sannolikt den del inom AI där utvecklingen gått snabbast och dess teknik används i allt från när du pratar med Siri eller får filmrekommendationer från någon filmkanal till att vara en viktig del i självkörande fordon. Tekniken som används i Deep Learning baseras på en speciell typ av algoritmer där man använder en vidareutvecklad variant av neurala nätverk. Hanteringen av dessa algoritmer är både svåra och beräkningskrävande. I många mindre AI applikationer används därför inte denna teknik utan i stället förlitar man sig på enklare algoritmer. I denna uppgift har vi därför tänkt att du ska bekanta dig med en sådan enklare algoritm. Du kommer först att implementera algoritmen och därefter utföra simuleringar för att ta reda på hur bra algoritmen är på att klassificera okända objekt. Innan vi beskriver algoritmen ger vi en kort introduktion till området klassificering inom AI, det vill säga det område som denna inlämningsuppgift fokuserar på.

### Kortfattat om klassificering

Klassificering inom AI innebär att en algoritm avgör till vilken klass ett okänt objekt tillhör och där klass ska uppfattas som ett känt objekt som definieras genom ett antal egenskaper som karaktäriserar objektet i fråga. Om det objekt som ska klassificeras uppvisar liknande egenskaper som objektet i en viss klass, kan man på goda grunder göra antagandet att det okända objektet tillhör denna specifika klass. Baserat på denna beskrivning kan sedan AI-algoritmer för klassificering utvecklas.

### Träning och träningsdata

Innan en algoritm kan användas måste den tränas. Rent praktiskt innebär det att man mäter upp egenskaperna hos ett (stort) antal kända objekt som sedan tilldelas till de klasser som ingår i klassificeringen. En sådan datamängd benämns träningsdata. Träningsdata beskriver alltså klassobjekts karakteristiska drag (dvs egenskaper) och utgörs ofta av mätbara parametrar såsom färg, vikt, längd etc. Förutom egenskaperna (som alltid är numeriska data och kan uppfattas som koordinater i ett koordinatsystem) innehåller varje post i träningsdata också alltid postens klasstillhörighet. Om de objekt som utgör träningsdata i en viss klass uppvisar en viss spridning (dvs egenskaperna inte är identiska) kommer dessa träningsdata att befinna sig inom ett (begränsat) område i koordinatsystemet (datarymden). Mäter vi upp egenskaperna hos objekt som utgör träningsdata från en annan klass, kommer dessa data också att befinna sig inom ett annat begränsat område i datarymden. Om – och detta är viktigt – dessa båda områden ligger tillräckligt åtskilda åt i datarymden borde det vara möjligt för en algoritm att avgöra till vilken av dessa två klasser som ett okänt objekt tillhör utifrån dess placering i datarymden.

### Objekten som ska klassificeras

De objekt som ska klassificeras ska ha exakt samma struktur som träningsdata. Det vill säga att de ska innehålla information om klasstillhörighet och egenskaper. Det bör nämnas att i en verklig applikation känner man (givetvis) inte klasstillhörigheten – det är ju den som ska bestämmas. I denna uppgift vill vi bestämma antalet felklassificeringar och därigenom ta reda på algoritmens precision. För att kunna göra detta måste vi därför känna till klasstillhörigheten för de objekt som ska klassificeras.

## kNN algoritmen

Efter att vi har bekantat oss med grunderna i klassificering är vi nu redo att implementera en klassificeringsalgoritm och vi väljer den så kallade kNN-algoritmen (k Nearest Neighbour). Algoritmen bygger på att först beräkna de Euklidiska avstånden mellan det objekt som ska klassificeras och samtliga träningsdata och därefter bestämma den klass som avståndsmässigt ligger närmast det okända objektet. Efter att man beräknat avstånden sorterar man dem i ökande ordning. Därefter låter man de  $k$  stycken kortaste avstånden mellan det okända objektet som ska klassificeras och posterna i träningsdata utgöra beslutsgrund för algoritmen. Algoritmen beslutar att det objekt som ska klassificeras kommer att tillhöra den klass som förekommer flest gånger i de  $k$  stycken data som utgör beslutsgrunden.

### Euklidiska avståndet

Det Euklidiska avståndet definieras som det kortaste avståndet mellan två koordinater i datarymden. För klassificering av objekt som består av två egenskaper beräknas avståndet  $D$  enligt nedanstående formel

$$D = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$$

där objektet som ska klassificeras har koordinaterna  $(x_0, y_0)$  och en post i träningsdata har koordinaterna  $(x_1, y_1)$  och ska uppfattas som tvådimensionella koordinatpar vilka kan ritas in i ett tvådimensionellt diagram.

För att beräkna avståndet  $D$  vid klassificering av objekt med tre egenskaper gäller följande formel:

$$D = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2}$$

där  $(x_0, y_0, z_0)$  och  $(x_1, y_1, z_1)$  ska uppfattas som tredimensionella koordinatpar vilka kan ritas in i ett tredimensionellt diagram.

**Beskrivning av de ingående delarna i programmet.**

Vi känner nu till hur kNN-algoritmen fungerar. Nästa steg blir att beskriva de funktioner vi behöver i Python för att lösa uppgiften. Programmet ska bestå av följande delar och vi använder modulerna NumPy och Matplotlib:

1. En egendefinierad funktion som genererar träningsdata.
2. En egendefinierad funktion som utgör kNN-algoritmen och returnerar resultatet av en utförd klassificering.
3. En egendefinierad funktion som analyserar och presenterar utförd klassificering.
4. Ett enkelt menyprogram som binder samman ovanstående egendefinierade funktioner till ett sammanhållet program.

**En viktig del av inlämningsuppgiften är att Du ska skriva källkoden till ovanstående egendefinierade funktioner och till menyprogrammet med hjälp av nedanstående beskrivningar.**

### Egendefinierad funktion som genererar träningsdata.

För att generera träningsdata behöver vi veta klasstillhörighet och egenskaper (koordinater) till varje post i träningsdata. Ett naturligt sätt att organisera denna data är att använda en matris. En rad i matrisen utgör en post av träningsdata. Första kolumnen innehåller klasstillhörighet och är ett numeriskt värde där siffran 1 betyder klass 1, siffran 2 betyder klass 2 osv. Övriga kolumner innehåller egenskapernas numeriska värden, det vill säga koordinater. Nedan visas ett exempel på hur en matris kan se ut med två klasser och två egenskaper:

$$\begin{bmatrix} 1 & 23.3 & 45.7 \\ 1 & 31.4 & 50.3 \\ 1 & 28.4 & 49.1 \\ \dots & \dots & \dots \\ \dots & \dots & \dots \\ \dots & \dots & \dots \\ 2 & 245.6 & 331.8 \\ 2 & 257.9 & 357.6 \\ 2 & 239.6 & 326.4 \\ \dots & \dots & \dots \\ \dots & \dots & \dots \\ \dots & \dots & \dots \end{bmatrix}$$

Figur 1

För att förtydliga ovanstående resonemang visar vi nedan också det principiella utseendet för en matris med tre klasser och tre egenskaper:

$$\begin{bmatrix} 1 & 23.3 & 45.7 & 77.3 \\ 1 & 31.4 & 50.3 & 91.9 \\ 1 & 28.4 & 49.1 & 87.8 \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ 2 & 245.6 & 331.8 & 168.5 \\ 2 & 257.9 & 357.6 & 171.5 \\ 2 & 239.6 & 326.4 & 182.9 \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ 3 & 75.7 & 127.5 & 432.6 \\ 3 & 72.1 & 132.2 & 467.9 \\ 3 & 91.3 & 118.6 & 418.6 \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \end{bmatrix}$$

Figur 2

Vi skapar ovanstående matriser genom att förutsätta att egenskaperna är normalfördelade värden med kända medelvärden och standardavvikelser. Dessa värden är indata till programmet i menyalternativ 1 och matas in vid körning och lagras i en matris enligt nedanstående struktur.

Nedanstående matris, *medel\_std\_matris*, visar strukturen hur medelvärden och standardavvikelser ska vara lagrade för två klasser och två egenskaper:

$$medel\_std\_matris = \begin{bmatrix} 1 & mv\_e1\_1 & std\_e1\_1 & mv\_e2\_1 & std\_e2\_1 \\ 2 & mv\_e1\_2 & std\_e1\_2 & mv\_e2\_2 & std\_e2\_2 \end{bmatrix}$$

Figur 3.

där *mv\_e1\_1* betyder medelvärde för egenskap 1 och klass 1 och *std\_e2\_2* betyder standardavvikelse för egenskap 2 och klass 2 osv.

För att visa hur matrisen växer med ett ökat antal klasser och egenskaper visas nedanstående matris vilken tillhör en klassificering med tre egenskaper och tre klasser:

$$medel\_std\_matris = \begin{bmatrix} 1 & mv\_e1\_1 & std\_e1\_1 & mv\_e2\_1 & std\_e2\_1 & mv\_e3\_1 & std\_e3\_1 \\ 2 & mv\_e1\_2 & std\_e1\_2 & mv\_e2\_2 & std\_e2\_2 & mv\_e3\_2 & std\_e3\_2 \\ 3 & mv\_e1\_3 & std\_e1\_3 & mv\_e2\_3 & std\_e2\_3 & mv\_e3\_3 & std\_e3\_3 \end{bmatrix}$$

Figur 4.

Normalfördelade värden kan skapas med några olika metoder i NumPy. Till exempel kan man använda metoden *random.normal*. Se referensdokumentationen för denna metod på NumPy:s hemsida.

Den egendefinierade funktionen som genererar träningsdata beskrivs här:

*kNN\_data(antal\_data\_per\_klass, medel\_std\_matris)*

Parametrar: *antal\_data\_per\_klass* (heltal)

- anger antalet träningsdata som ska genereras per klass.

*medel\_std\_matris* (ndarray)

- en NumPy array med innehåll enligt den principiella struktur som visas i Figur 3 och Figur 4.

Returnerar: (ndarray)

- en NumPy array med data med ett innehåll enligt den principiella struktur som visas i Figur 1 och Figur 2.

### Egendefinierad funktion som utgör kNN-algoritmen och returnerar resultatet av en utförd klassificering

Den egendefinierade funktionen som utgör själva kNN algoritmen beskrivs här:

*kNN\_algoritm(k, training\_data, okanda\_objekt)*

Parametrar: *k* (heltal)

- anger antal träningsdata som utgör beslutsgrunden. Om  $k = 7$  bestäms klasstillhörigheten utifrån de 7 träningsdata som har de kortaste avstånden till objektet som ska klassificeras.

*training\_data* (ndarray)

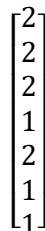
- Numpy array som innehåller träningsdata enligt den principiella struktur som visas i Figur 1 och Figur 2. Algoritmen ska kunna hantera träningsdata med maximalt tre (3) egenskaper och tre (3) klasser.

*okanda\_objekt* (ndarray)

- Numpy array som innehåller de okända objekt som ska klassificeras och består av egenskapernas värden. Alltså samma struktur som i Figur 1 och Figur 2 fast där kolumnen med klasstillhörighet (kolumn 1) utelämnats.

Returnerar: Funktionen returnerar två matriser av typen *ndarray* vilka beskrivs nedan:

- 1) En *ndarray* som innehåller resultatet av en klassificering. Matrisen har en (1) kolumn (se Figur 5) med lika många rader som antalet okända objekt som har klassificerats. I första raden finns resultatet av det första okända objektet som klassificerades osv. Figur 5 visar ett exempel på hur en sådan matris kan se ut efter att 7 okända objekt har klassificerats:


$$\begin{bmatrix} 2 \\ 2 \\ 2 \\ 1 \\ 2 \\ 1 \\ 1 \end{bmatrix}$$

Figur 5.

Tolkningen av ovanstående matris är att algoritmen beslutade att objekten som klassificerades tillhör klasserna 2,2,2,1,2,1,1.

- 2) sorterad lista (*ndarray*), i stigande ordning, som innehåller de  $k$  stycken kortaste avstånden och som utgör beslutsgrunden för algoritmen. Det längsta avståndet som finns i denna lista kommer att användas för att rita en cirkel i funktionen *kNN\_analysera*.

### Egendefinierad funktion som analyserar och presenterar utförd klassificering

Den egendefinierade funktionen som analyserar och presenterar en utförd klassificering beskrivs här:

*kNN\_analysera(k, training\_data, okanda\_objekt, sorterad\_lista, kNN\_resultat, korrekt\_resultat = None)*

Parametrar: *k* (heltal)

- anger antal träningsdata som utgör beslutsgrunden. Om  $k = 7$  bestäms klasstillhörigheten utifrån de 7 träningsdata som har de kortaste avstånden till objektet som ska klassificeras.

*training\_data* (ndarray)

- en Numpy array som innehåller träningsdata enligt den principiella struktur som visas i Figur 1 och Figur 2.

*okanda\_objekt* (ndarray)

- en Numpy array som innehåller de okända objekt som ska klassificeras och består av egenskapernas värden. Alltså samma struktur som i Figur 1 och Figur 2 fast där kolumnen med klasstillhörighet (kolumn 1) utelämnats.

*sorterad\_lista* (ndarray)

- en Numpy array som innehåller de  $k$  stycken kortaste avstånden som beräknades i funktionen *kNN\_algoritm*. Denna parameter används för att kunna rita cirkeln i menyalternativ 1. Den måste skickas med i menyalternativ 2 men kommer då inte att användas.

*kNN\_resultat* (ndarray)

- är den NumPy array som returneras av funktionen *kNN\_algoritm* och som innehåller resultatet av en utförd klassificering.

*korrekt\_resultat* (ndarray)

- en NumPy array som innehåller klasstillhörigheten till de okända objekt som ska klassificeras. I en verklig applikation känner man (givetvis) inte klasstillhörigheten – det är ju den som ska bestämmas. Vi vill kunna bestämma antalet felklassificeringar och därigenom ta reda på algoritmens precision. För att kunna göra detta måste vi känna till klasstillhörigheten för de objekt som ska klassificeras.
- Denna parameter utesluts vid menyalternativ 1. Den får då defaultvärdet *None*, vilket kan användas i funktionen för att avgöra om utskrift skall ske enligt menyalternativ 1 eller 2.

Returnerar: *True* om analysen och plottningen lyckades annars *False*

Detta ska funktionen *kNN\_analysera* utföra:



1. Skriva ut resultatet enligt nedanstående uppställning:

för menyalternativ 1:

```
-----  
Aktuellt k-värde: 11  
Algoritmen klassificerade det okända objektet till klass 2  
-----
```

för menyalternativ 2:

```
-----  
Aktuellt k-värde: 7  
Antal utförda klassificeringar: 20  
Antal felklassificeringar: 2  
Procent felklassificeringar: 10.00 %  
-----
```

2. Rita punktdiagram

Funktionen skapar ett punktdiagram och placerar koordinaterna för träningsdata och objekten som klassificerats antingen i ett 2- eller 3-dimensionellt punktdiagram beroende på antalet egenskaper. Varje klass ska ha en egen färg i punktdiagrammet för att man enkelt ska kunna se klasstillhörighet. Felklassificerat okänt objekt ska ha svart färg.

(Exempel på utformning av dessa diagram visas under avsnittet **Utdata från programmet** längre fram i detta dokument)

Menyprogram som binder samman de egendefinierade funktionerna till ett sammanhållet program.

I denna uppgift ska du skriva ett enkelt menyprogram som består av nedanstående menyalternativ och som binder ihop de funktioner som du utvecklat:

1. Visa grafiskt en klassificering bestående av två klasser och ett okänt objekt.
2. Utför klassificering och presentera resultatet när träningsdata och okända objekt läses från CSV-fil.
3. Avsluta programmet.

Nya menyval ska kunna matas in tills användaren väljer menyalternativ 3.

#### Menyalternativ 1:

Under detta menyalternativ genereras en grafisk presentation över en klassificering av ett (1) okänt objekt och där träningsdata består av två klasser med två egenskaper.

Användaren ska mata in följande information:

- k-värdet.
- antal träningsdata per klass .
- medelvärde och standardavvikelse för respektive klass olika egenskaper. Informationen sparas i en matris enligt strukturen i Figur 3.
- det okända objektets koordinater (x0, y0).

Funktionen *kNN\_data* ska användas för att generera träningsdata och funktionen *kNN\_algoritm()* används för att utföra klassificeringen.

Presentationen av en utförd klassificering består av nedanstående två delar:

1. textinformation som anger klasstillhörighet för objektet som klassificerats.
2. ett punktdiagram där träningsdata för de två klasserna finns inritade. Träningsdata som för klass 1 har röd färg och träningsdata för klass 2 har blå. Det okända objektet ska ritas som en gul cirkel. För att få en grafisk bild över hur algoritmen beslutar klasstillhörighet ska även en cirkel med centrum i koordinaten för det okända objektet (x0, y0) ritas. Dess radie ska vara lika med det längsta avstånd som ingår i beslutsgrunden. Cirkeln kommer då att innesluta de träningsdata som ingår i beslutsgrunden och man kan enkelt grafiskt, direkt i punktdiagrammet, avgöra om klassificeringen är korrekt. Cirkeln ritas med metoden *plt.Circle((x0,y0),radie,fill=False)* och där x0, y0 är koordinaten för cirkelns centrum, *radie* är cirkelns radie och *fill=False* anger att cirkeln är genomskinlig varvid inneslutna träningsdata blir synliga. Referensdokumentationen för *plt.Circle* finns på Matplotlibs hemsida.

### Menyalternativ 2:

Under detta menyalternativ utförs klassificering, analys och presentation enligt nedanstående beskrivning:

Träningsdata och objekt som ska klassificeras hämtas från CSV-filer. Träningsdata ska finnas lagrad i en fil (t.ex. *tranings\_data.csv*) med en principiell struktur enligt Figur 1 eller Figur 2. Objekten som ska klassificeras ska finnas lagrad i en annan fil (t.ex. *okanda\_objekt.csv*) med samma principiella struktur som hos träningsdata fast där klasstillhörigheten saknas. Klassificeringen utförs av funktionen *kNN\_algoritm* och analys och presentation av funktionen *kNN\_analysera*.

Givna förutsättningar för filen med träningsdata:

- Varje klass har lika många träningsdata.
- Träningsdata för klass 1 ligger först i filen därefter kommer träningsdata för klass 2 osv.

För att bestämma antal felklassificeringar måste det även finnas en CSV-fil (t.ex. *resultat.csv*) som innehåller klasstillhörigheten för de okända objekten. Informationen från denna fil läses in i matrisen *korrekt\_resultat* som är en inparameter till funktionen *kNN\_analysera*. Genom att jämföra klassificeringsresultatet som returnerats från funktionen *kNN\_algoritm* med innehållet i *korrekt\_resultat* kan antalet felklassificeringar enkelt beräknas.

Inför en klassificering, analys och presentation matar användaren in följande information:

- k-värdet.
- filnamnet på filen som innehåller träningsdata.
- filnamnet på filnamnet som innehåller objekten som ska klassificeras.
- filnamnet på filen som innehåller de okända objektens klasstillhörighet.

Efter inmatningen är klar utförs klassificeringen av funktionen *kNN\_algoritm* och därefter analyseras och presenteras resultatet genom att anropa funktionen *kNN\_analysera*.

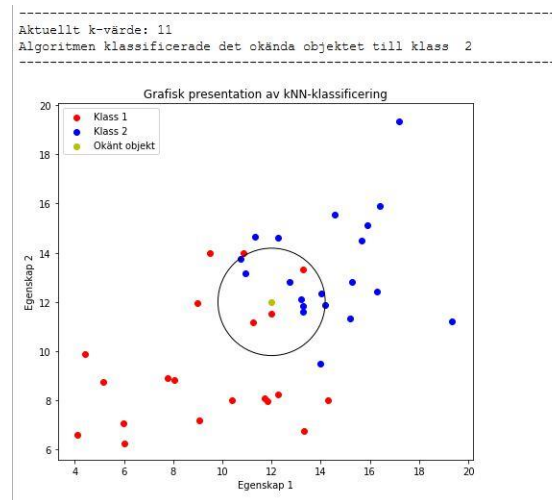
### Menyalternativ 3:

Programmet avslutas.

## Utdata från programmet

Nedan visas exempel på programkörning från de två menyalternativen.

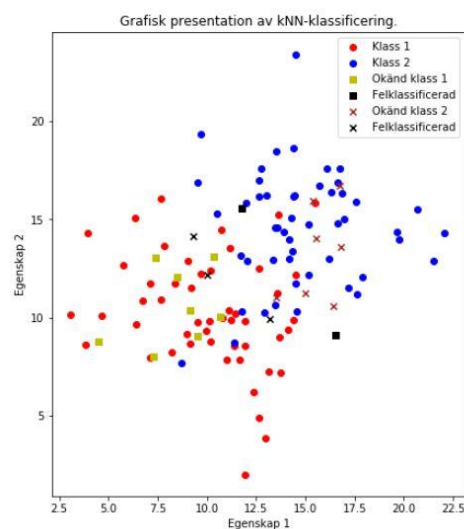
### Menyalternativ 1:



### Menyalternativ 2:

2-dimensionellt punktdiagram. Både träningsdata och okända objekt består av två klasser:

```
-----
Aktuellt k-värde: 5
Antal utförda klassificeringar: 20
Antal felklassificeringar: 5
Procent felklassificeringar: 25.00 %
-----
```

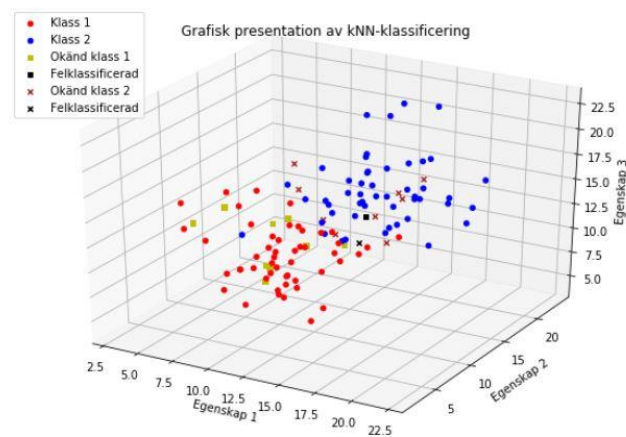


Vi noterar i diagrammet att algoritmen klassificerade fem okända objekt felaktigt.

3-dimensionellt punktdiagram. Både träningsdata och okända objekt består av två klasser:

För att (förhoppningsvis) minska antalet felklassificeringar har vi här lagt till ytterligare en egenskap (egenskap 3) till föregående datamängd med två egenskaper. Resultatet har blivit att antalet felklassificeringar minskat från fem (5) till två (2), alltså en signifikant förbättring. Skulle ytterligare egenskaper bidra till ett ännu bättre resultat? Fundera gärna kring detta. Flera egenskaper ökar samtidigt beräkningsbelastning. Även val av k-värde påverkar resultat och beräkningsbelastning. I en praktisk tillämpning försöker man hitta en bra avvägning mellan dessa.

```
-----  
Aktuellt k-värde: 5  
Antal utförda klassificeringar: 20  
Antal felklassificeringar: 2  
Procent felklassificeringar: 10.00 %  
-----
```



Bifogat inlämningsuppgiften finns källkodsfilen `exempel_3D_punktdiagram.py` som skapar ett 3D punktdiagram. Utgå gärna från den källkoden för att skapa 3D punktdiagram som passar denna uppgift.

Obs! programsatsen `from mpl_toolkits import mplot3d` måste finnas i ditt program för att ett 3D-punktdiagram ska kunna skapas.

## Simuleringar

Efter att du har färdigställt programmet är det dags att utföra klassificeringar under menyalternativ 1 och 2.

### Menyalternativ 1:

Under detta menyalternativ lär du känna hur algoritmen arbetar. Börja med att utföra en klassificering bestående av träningsdata med 10 träningsdata per klass och sätt  $k = 5$ .

Välj följande medelvärden och standardavvikelser för träningsdata:

	Klass 1	Klass 2
Medelvärde egenskap 1:	50	100
Medelvärde egenskap 2:	50	100
Standardavvikelse egenskap 1:	5	5
Standardavvikelse egenskap 2:	5	5

Välj koordinaten för det okända objektet till  $(x_0, y_0) = (75, 75)$ , det vill säga att det okända objektet placeras ungefär mitt emellan de båda klasserna. Utför klassificeringen och studera resultatet. Den cirkel som ritats ska innehålla totalt  $k$  stycken träningsdata. Algoritmen ska ha klassificerat det okända objektet till den klass som förekommer flest gånger inom cirkeln. Stämmer detta?

Utför sedan en ny klassificering där du placerar koordinaterna för det okända objektet närmare en av klasserna och studera resultatet. Får du förväntat resultat?

Fortsätt att utföra klassificeringar under detta menyalternativ genom att ändra  $k$ -värde och antal träningsdata per klass och placera det okända objektet på olika platser i datarymden. Får du förväntade resultat?

(Utmaningen när man implementerar kNN-algoritmen i en verklig applikation är (bland annat) att hitta det  $k$ -värde som över tid ger det bästa klassificeringsresultatet. Ofta måste man trimma algoritmen efter att den implementerats genom att justera detta värde och även lägga till mer träningsdata).

### Menyalternativ 2:

Använd de filer som bifogats inlämningsuppgiften och utför en klassificering, analys och presentation. Studera resultatet med avseende på antal felklassificeringar. Studera också punktdiagrammet. Verkar det stämma?

Går det att förbättra klassificeringsresultatet genom att ändra  $k$ -värdet? Prova några olika  $k$ -värden och reflektera över resultatet.