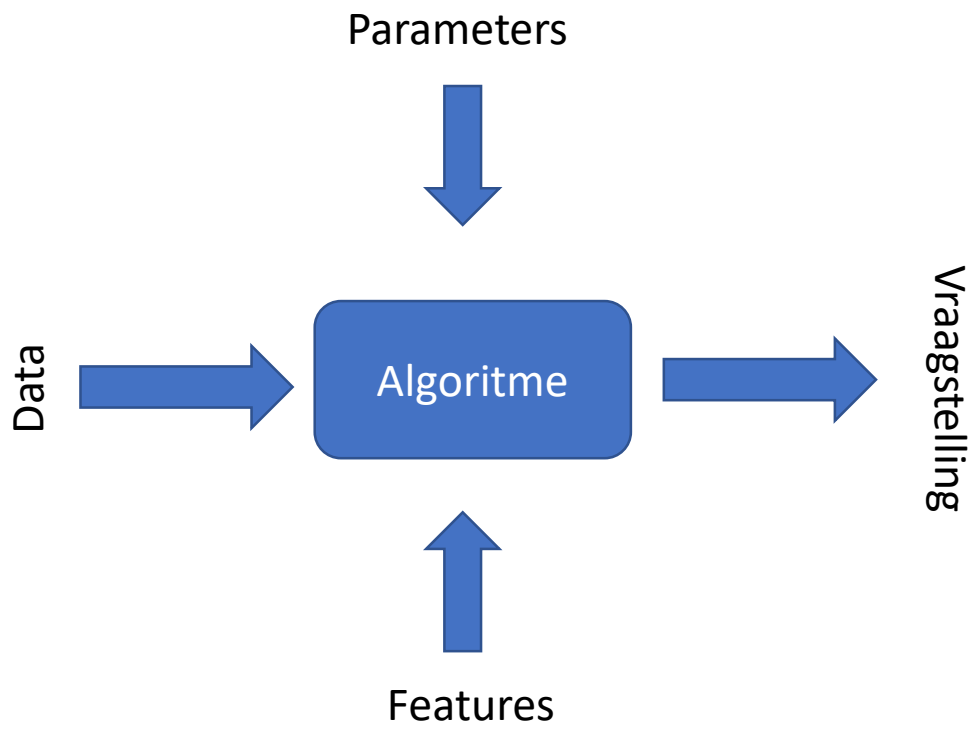
Oefening 1 – Quick Titanic

In [37]:

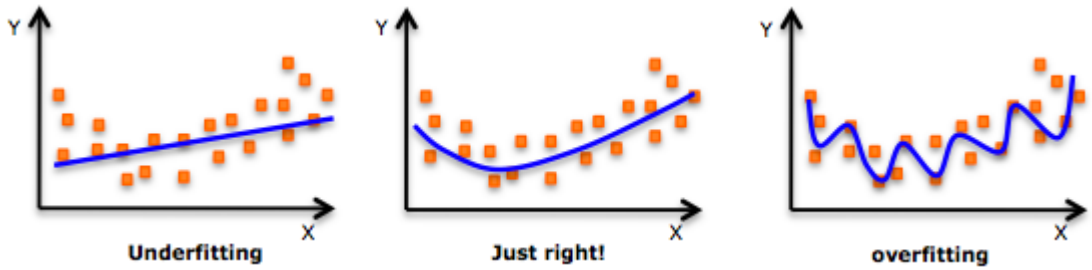
```
import pandas as pd
import numpy as np

from sklearn import preprocessing

from sklearn.svm import SVC


train_df = pd.read_csv('./data/train.csv').dropna()
test_df = pd.read_csv('./data/test.csv').dropna()

X_train = train_df.drop(["Survived"], axis=1)
Y_train = train_df["Survived"]

X_test  = test_df.drop("PassengerId", axis=1).copy()
```

```
le = preprocessing.LabelEncoder()

for column_name in X_train.columns:
    if X_train[column_name].dtype == object:
        X_train[column_name] = le.fit_transform(X_train[column_name].astype(str))

svc = SVC(kernel="rbf", gamma="auto")
svc.fit(X_train, Y_train)

Y_train_pred=svc.predict(X_train)

print(svc.score(X_train, Y_train_pred))
```

Stochastic in Machine Learning



Under/overfitting



Feature selection

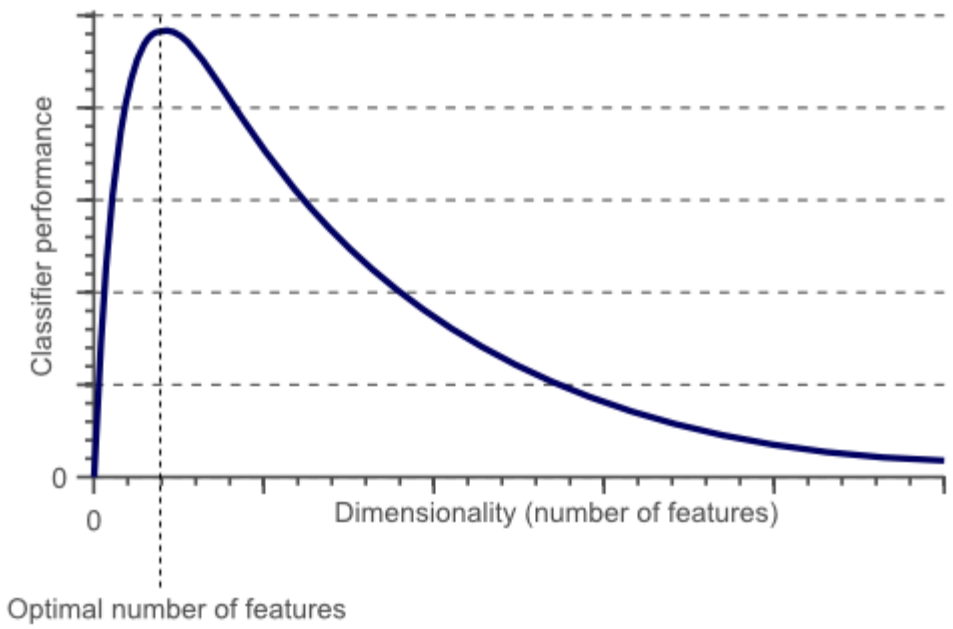


Figure 1: Relationship between Classifier Performance and Dimensionality [2]

It is statistically proven that when performing a Machine Learning task there exist an optimal number of features which should be used for every specific task (Figure 1). If more features are added than the ones which are strictly necessary, then our model performance will just decrease

SVM

Supervised algoritme voor classificatie en regressie

SVM introductie

- Support Vector Machine
- Supervised
- Classificatie & Regressie
- In de basis is dit een lineair model
  - Niet verplicht
- “The goal of a support vector machine is to find the optimal separating hyperplane which maximizes the margin of the training data.” -> Alexandre Kowalczyk
- -1 | +1

Hyperplane

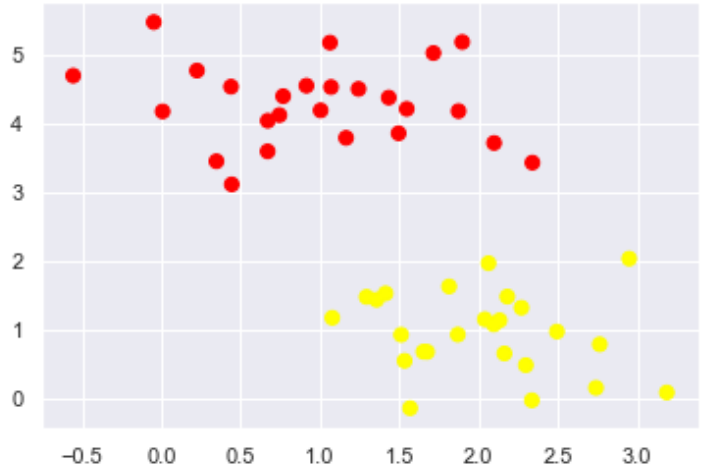
- Hyperplane definitie is $w^T x = 0$ in een 2-dimensionale ruimte
  - $w^T x >= 1$ voor een positieve labeling
  - $w^T x <= -1$ voor een negatieve labeling

In general, a hyperplane in $\mathbb{R}^n$ is an $(n-1)$-dimensional subspace of $\mathbb{R}^n$

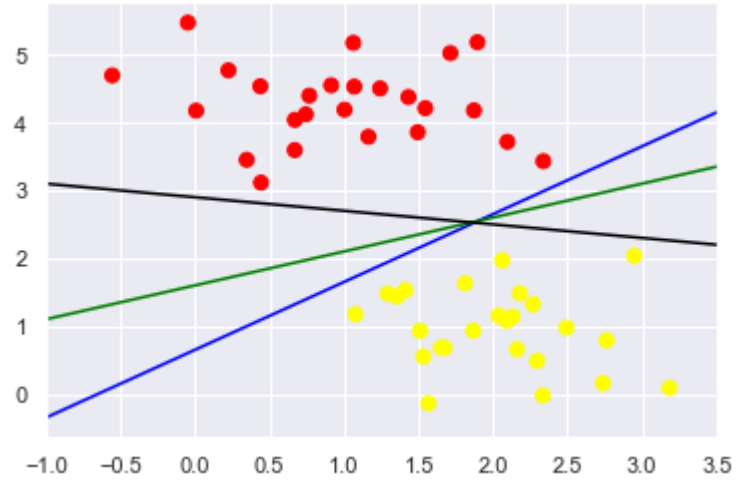
SVM aannames

- De gelabelde data kan via een hyperplane worden gescheiden
- Deze hyperplane kan als volgt worden gedefinieerd in een 2-dimensionale ruimte: $y = ax + b$
- Door te variëren met a en b kan de hyperplane op meerdere manieren worden gedefinieerd, er is dus meer dan 1 oplossing mogelijk
- SVM is een algoritme welke, gegeven de beschikbare gelabelde data, een optimale hyperplane definitie zoekt
- Optimale hyperplane in SVM wordt bepaald door Quadratic Programming
- Werkt met numerieke input
- Basis SVM gaat om binaire classificatie

Simpel voorbeeld

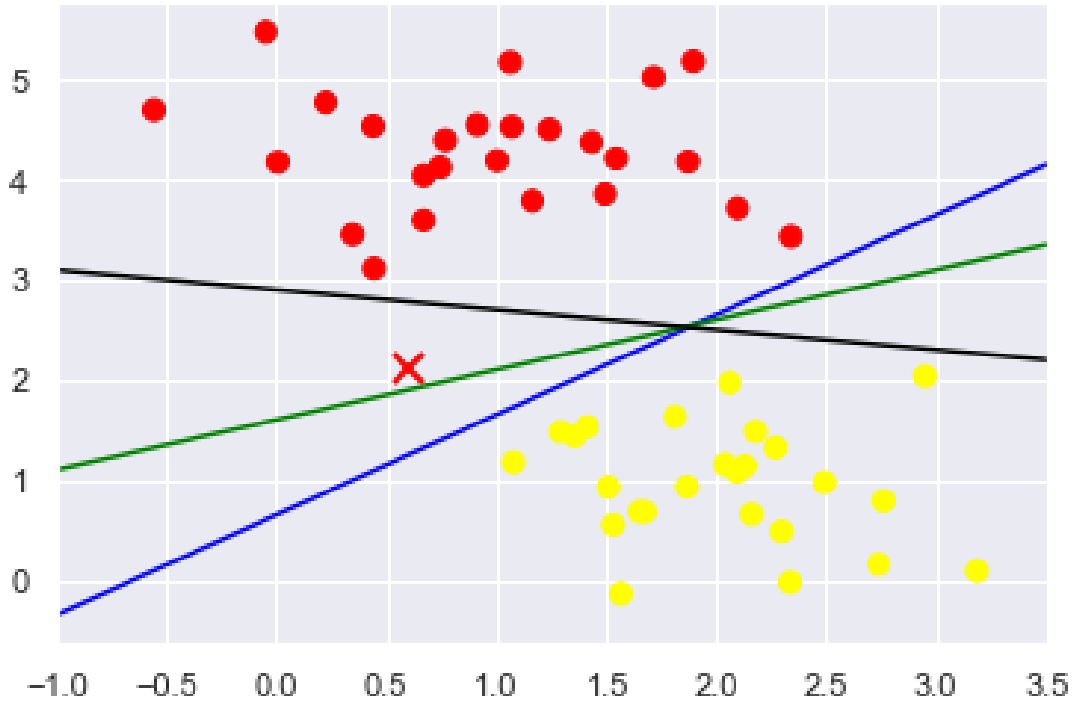



```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# use seaborn plotting defaults
import seaborn as sns; sns.set()

from sklearn.datasets.samples_generator import make_blobs
X, y = make_blobs(n_samples=50, centers=2,
                  random_state=0, cluster_std=0.60)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn');
```



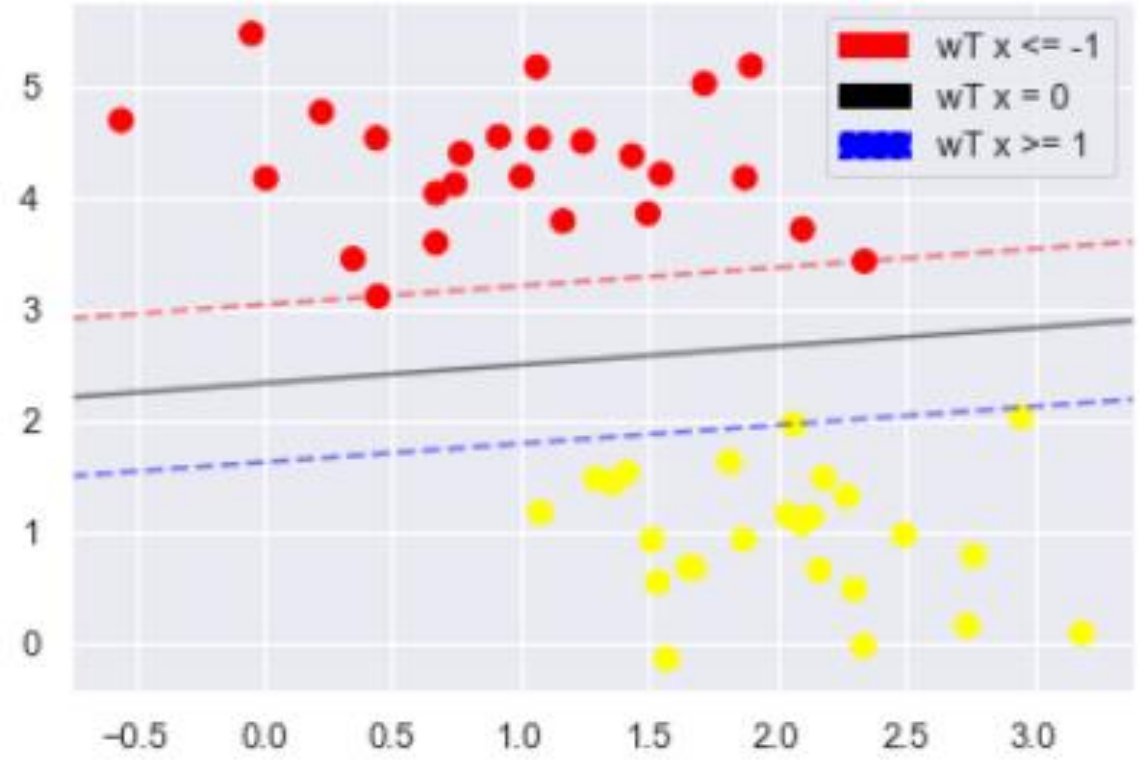

```
xfit = np.linspace(-1, 3.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')

for m, b, c in [(1, 0.65, "blue"), (0.5, 1.6, "green"), (-0.2, 2.9, "black")]:
    plt.plot(xfit, m * xfit + b, '-k', color=c)

plt.xlim(-1, 3.5);
```

Wat als…




```
xfit = np.linspace(-1, 3.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plt.plot([0.6], [2.1], 'x', color='red', markeredgewidth=2, markersize=10)

for m, b, c in [(1, 0.65, "blue"), (0.5, 1.6, "green"), (-0.2, 2.9, "black")]:
    plt.plot(xfit, m * xfit + b, '-k', color=c)

plt.xlim(-1, 3.5);
```

Support Vectors



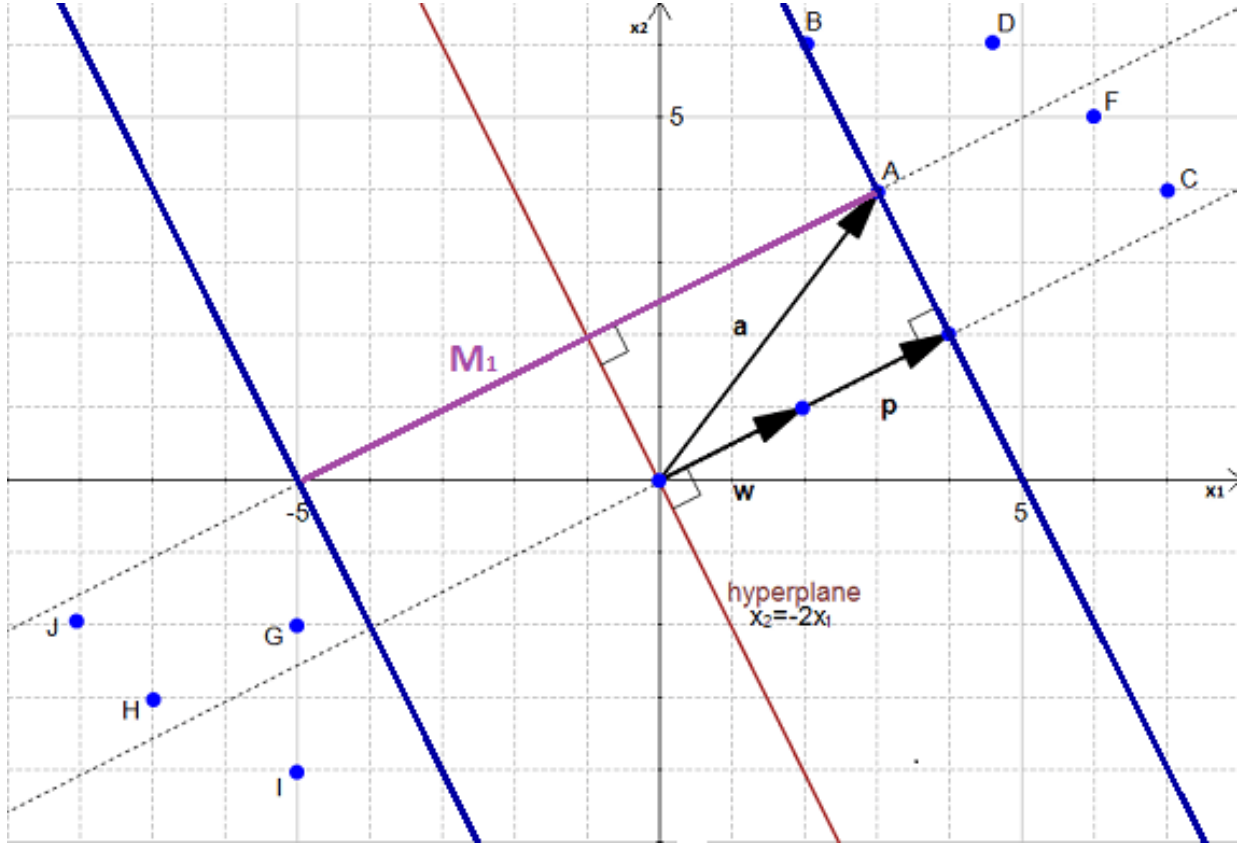

```python
# Simpel model opbouw:
from sklearn.svm import SVC # "Support vector classifier"

import matplotlib.patches as mpatches

model = SVC(kernel='linear', C=1E10)
model.fit(X, y)

# Hulpmethode voor het visualiseren:
def plot_svc_decision_function(model, ax=None, plot_support=True):
    """Plot the decision function for a 2D SVC"""
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # create grid to evaluate model
    x = np.linspace(xlim[0], xlim[1], 30)
    y = np.linspace(ylim[0], ylim[1], 30)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)

    # plot decision boundary and margins
    ax.contour(X, Y, P, colors=['red', 'black', 'blue'],
               levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--']
              )
    sVHandles = [mpatches.Patch(linestyle='--', linewidth=.1, color="red"), mpatches.Patch(linestyle='-', color="black"),
                 mpatches.Patch(linestyle='--', color="blue")]
    ax.legend(sVHandles, ['wT x <= -1','wT x = 0','wT x >= 1'])

    # plot support vectors
    if plot_support:
        ax.scatter(model.support_vectors_[:, 0],
                   model.support_vectors_[:, 1],
                   s=300, linewidth=1, facecolors='none');
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(model);
```

“The goal of a support vector machine is to find the optimal separating hyperplane which maximizes the margin of the training data.”

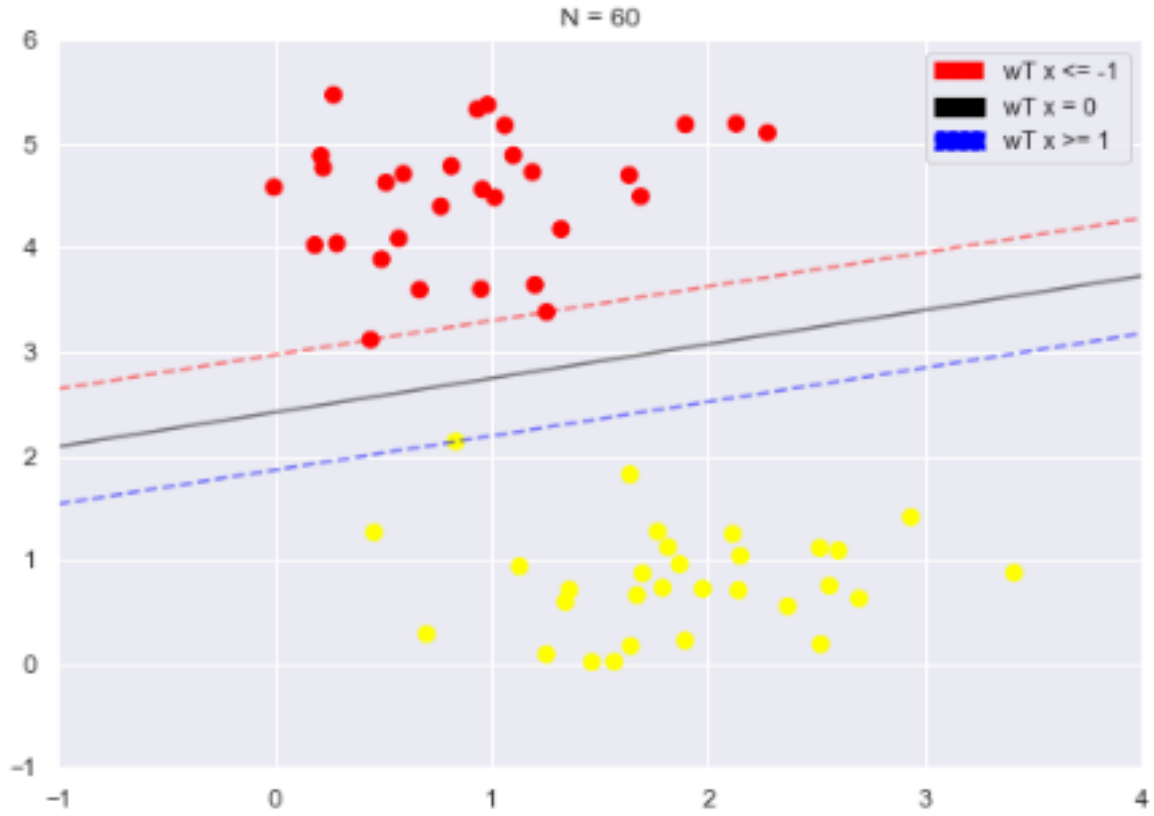
Wat zijn nu support vectors?

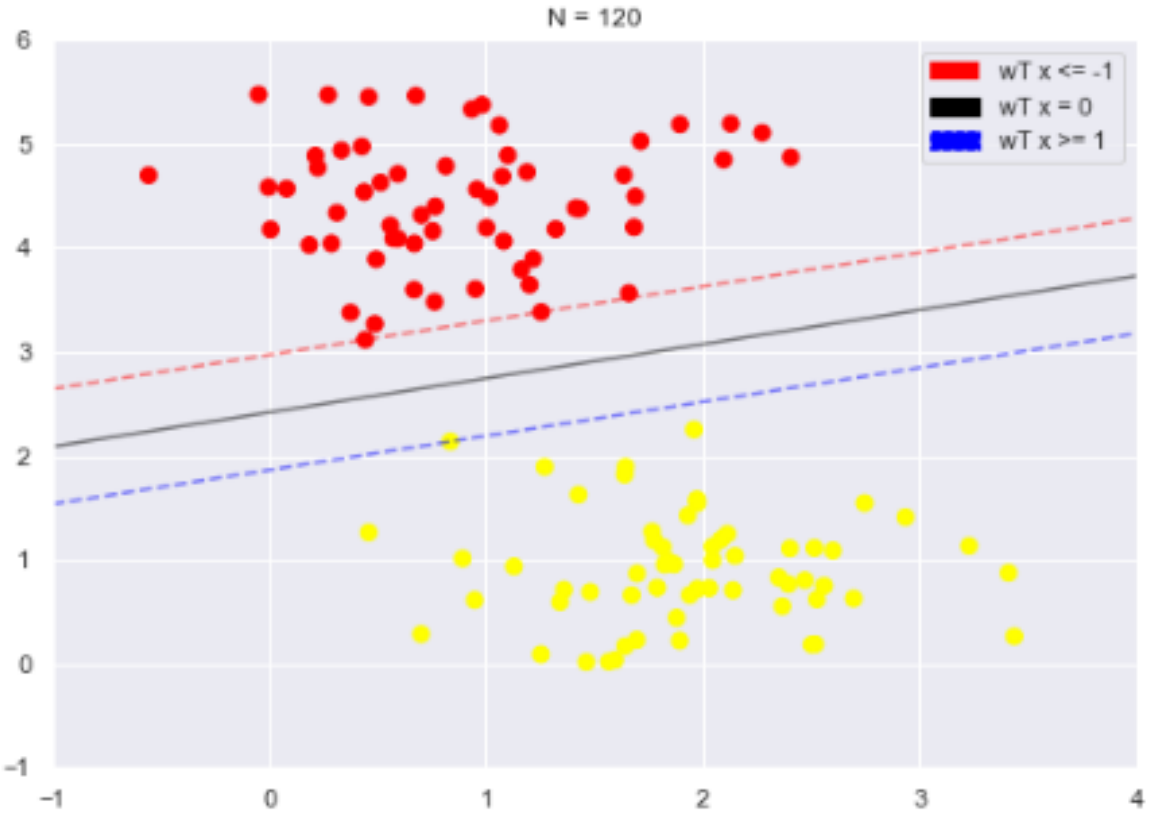
- Zijn de data punten welke het dichtst bij de hyperplane liggen
- Het zijn de data punten welke het moeilijkste zijn te classificeren
- Zijn direct verantwoordelijk voor de definitie van de optimale hyperplane

Optimale hyperplane



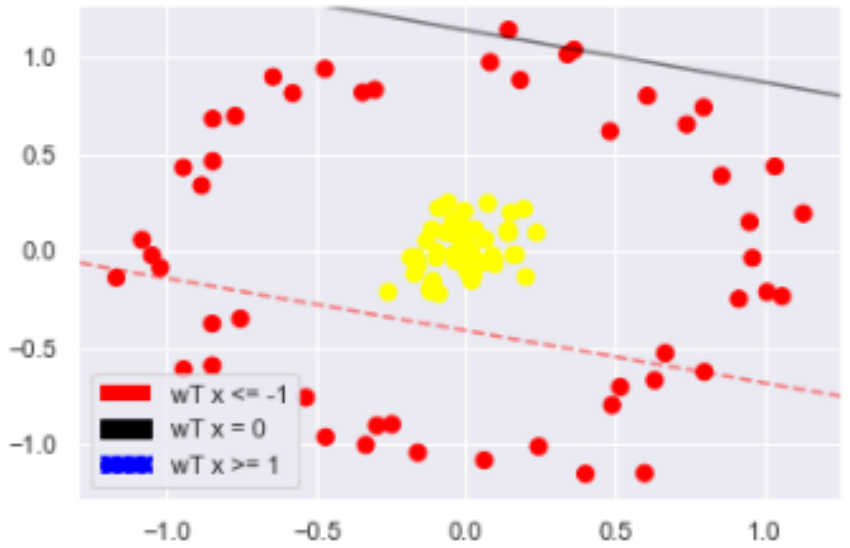
Kracht van SVM





SVM Kernels

```
from sklearn.datasets.samples_generator import make_circles
X, y = make_circles(100, factor=.1, noise=.1)

clf = SVC(kernel='linear').fit(X, y)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(clf, plot_support=False);
```



SVM Kernels

- Als er geen lineaire oplossing mogelijk is
- Patroon-herkennings algoritme welke gebruikt wordt door SVM in N-dimensionale ruimtes

Types SVM

- SVC
  - Linear
  - Poly
  - RBF
  - Sigmoid
- SVR
- Multi class

SVM parameters

- c: Dit bepaalt hoe hard het criterium "niemandsland" is. Hoe hard is het dat er geen data punten binnen de marge van de gestelde hyperplane mogen liggen. Een kleinere waarde voor c staat toe dat er data punten binnen de marge mogen liggen. Een hoge waarde voor c staat het niet toe dat er datapunten binnen de marge mogen liggen
- γ: Alleen voor rbf, poly en sigmoid kernels

Oefening 2 –Titanic

```
#All imports:
# data analysis and wrangling
import pandas as pd
import numpy as np
import random as rnd

# visualization
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# machine learning
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
```

De data

Er zijn 2 data sets klaar gezet, 1 om het model te trainen en 1 om mee te testen. De set om mee te trainen heeft al de overlevings voorspelling per passagier terwijl de test set deze niet heeft

```
train_df = pd.read_csv('./data/train.csv')
test_df = pd.read_csv('./data/test.csv')
combine = [train_df, test_df]
```

```
print(train_df.columns.values)
```

```
['PassengerId' 'Survived' 'Pclass' 'Name' 'Sex' 'Age' 'SibSp' 'Parch'
 'Ticket' 'Fare' 'Cabin' 'Embarked']
```

Beschrijving van de features

- Categorical: Survived, Sex en Embarked
- Ordinal: Pclass
- Numerical: Age, Fare, SibSp en Parch
- Alpha: Ticket en Cabin

```
# preview the data
print(train_df.shape)
train_df.head()
```

```
(891, 12)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```
train_df.tail()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.00	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.00	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.45	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75	NaN	Q

```
train_df.info()
print('_'*40)
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
PassengerId    418 non-null int64
Pclass         418 non-null int64
Name           418 non-null object
Sex            418 non-null object
Age            332 non-null float64
SibSp          418 non-null int64
Parch          418 non-null int64
Ticket         418 non-null object
Fare           417 non-null float64
Cabin          91 non-null object
Embarked       418 non-null object
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
```

Typefouten

- Name

Is hier een aanpassing voor nodig?

Null en lege waarden

- Cabin
- Age

Is hier een aanpassing voor nodig?

```
#Numerical values:
train_df.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Beschrijving van de data

Distributie van de numerieke waarden

- 891 passagiers in de training set
- Waardes in "Survived" is 0 | 1
- Ongeveer 38% van deze set heeft het overleefd
- > 75% reisde zonder ouders/kinderen

```
#Categorical data:
train_df.describe(include=['O'])
```

	Name	Sex	Ticket	Cabin	Embarked
count	891	891	891	204	889
unique	891	2	681	147	3
top	Ford, Miss. Robina Maggie "Ruby"	male	1601	C23 C25 C27	S
freq	1	577	7	4	644

Beschrijving van de categorische data

- 891 unieke namen
- 2 unieke waardes voor Sex, 65% man
- 147 unieke cabins
- De meesten zijn ingestapt op locatie S (Southampton)
- Tickets bevatten ongeveer 24% aan duplicaten:

```
train_df['Ticket'].value_counts()[lambda x: x>1]
```

```
1601            7
347082          7
CA. 2343        7
3101295         6
CA 2144         6
347088          6
S.O.C. 14879    5
382652          5
W./C. 6608      4
19950           4
349909          4
2666            4
LINE            4
113760          4
113781          4
17421           4
347077          4
```

Te valideren aannames

- **Correlatie** Wat is de correlatie van elke feature tov Survival.
- **Data aanvulling**
  - Moeten we/kunnen we Age aanvullen?
  - Wat doen we met Embarked?
- **Data correctie**
  - Kunnen we Ticket als feature laten afvallen?
  - Kunnen we de Cabin feature uberhaupt gebruiken?
  - Passenger Id heeft geen toegevoegde waarde en kan worden laten vallen.
  - Wat te doen met de Name feature? Kan een naam een correlatie hebben met de survival rate?
- **Feature creation** Kunnen we een aantal features expliciet toevoegen?
  - Bijvoorbeeld een feature Familie gebaseerd op de features Parch en SibSp?
  - Kunnen we de titel van een persoon uit het naam veld halen en deze gebruiken als nieuwe feature?
  - Heeft het zin om op leeftijd te groeperen? Dus om een ordinale feature te maken van de numerieke leeftijds (Age) feature?
- **Generieke aannames:**
  - Vrouwen hebben een grotere kans op overleving
  - Kinderen jonger dan ... hebben een grotere kans op overleving
  - Upper-class passagiers hebben een grotere kans op overleving

Correlatie

Eerst gaan we de correlatie van de verschillende features met de Survival feature bekijken. Zonder aanpassingen aan de data kunnen we dit alleen doen voor features zonder null-waarden. Daarnaast heeft het alleen zin om dit te doen op categorische (Sex), ordinale (Pclass) en discrete (SibSP, Parch) data types.

```
#Pclass:
train_df[['Pclass', 'Survived']].groupby(['Pclass'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

	Pclass	Survived
0	1	0.629630
1	2	0.472826
2	3	0.242363

```
train_df[["Sex", "Survived"]].groupby(['Sex'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

	Sex	Survived
0	female	0.742038
1	male	0.188908

```
train_df[["SibSp", "Survived"]].groupby(['SibSp'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

	SibSp	Survived
1	1	0.535885
2	2	0.464286
0	0	0.345395
3	3	0.250000
4	4	0.166667
5	5	0.000000
6	8	0.000000

```
train_df[["Parch", "Survived"]].groupby(['Parch'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

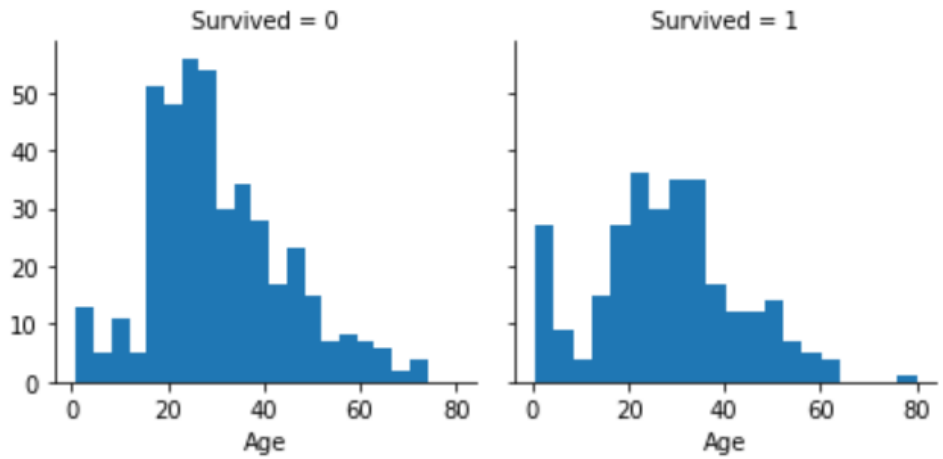
	Parch	Survived
3	3	0.600000
1	1	0.550847
2	2	0.500000
0	0	0.343658
5	5	0.200000
4	4	0.000000
6	6	0.000000

Visualisatie

Voor de correlatie tussen numerieke waardes en Survival rate maken we gebruik van visualisatie. Voor Age plotten we een histogram:

```
g = sns.FacetGrid(train_df, col='Survived')
g.map(plt.hist, 'Age', bins=20)
```

```
<seaborn.axisgrid.FacetGrid at 0x1680953d978>
```



Wat we hier zien:

- Kinderen jonger dan 4 jaar oud hebben een hoge overlevingskans
- Iedereen rond de 80 jaar oud heeft het overleefd
- Tussen de 15 en 25 jaar oud is er een groot aantal van niet-overlevenden
- De grootste groep passagiers is tusssen de 15 en 25 jaar oud

Kijkende naar de aannames welke we hierboven hebben gedaan:

- We kunnen leeftijd als feature gaan gebruiken
- We moeten proberen de ontbrekende leeftijd waardes (null-values) aan te vullen
- Het heeft zin om de leeftijd feature te groeperen

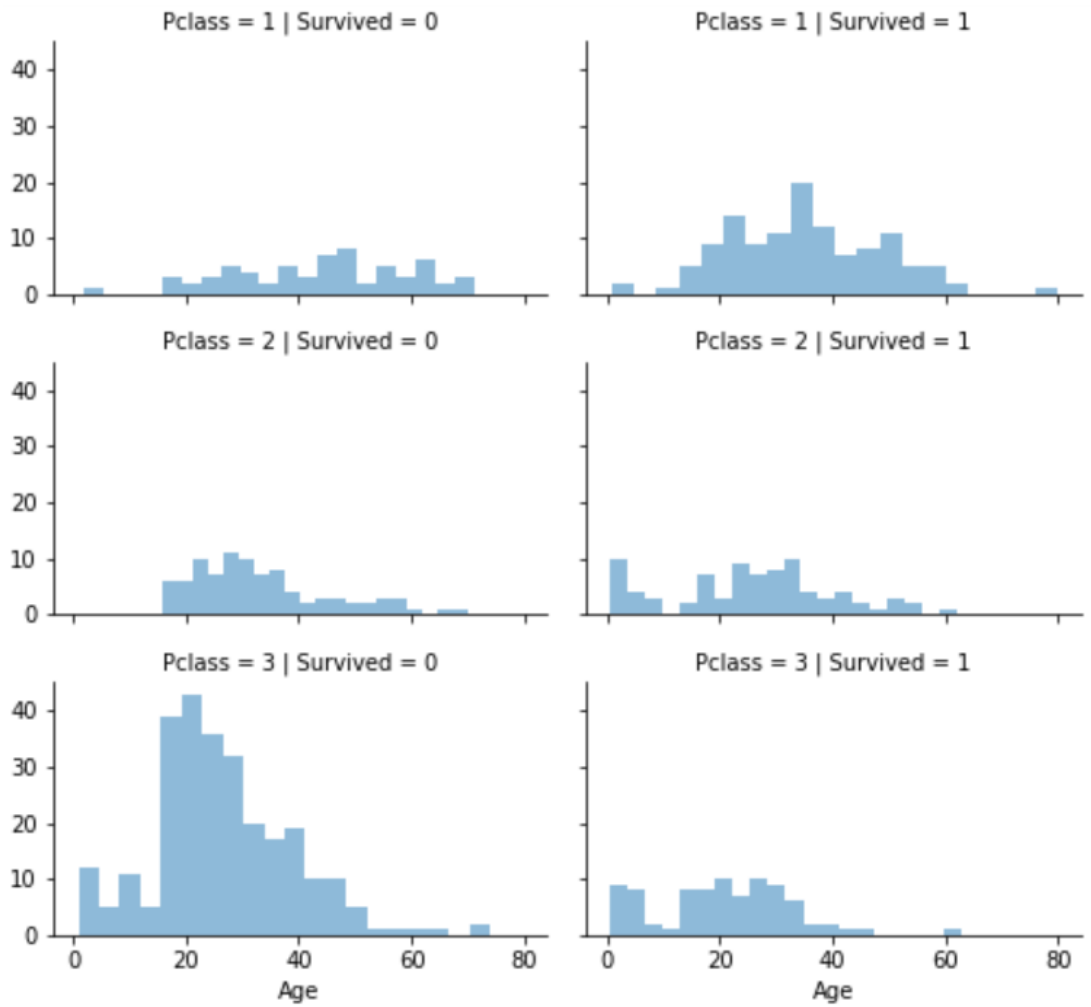
Combinatie van numerieke en ordinale features

We gaan nu verschillende features combineren in één plot om onderlinge correlaties te proberen te identificeren:

```
grid = sns.FacetGrid(train_df, col='Survived', row='Pclass', height=2.2, aspect=1.6)
grid.map(plt.hist, 'Age', alpha=.5, bins=20)
grid.add_legend();
```

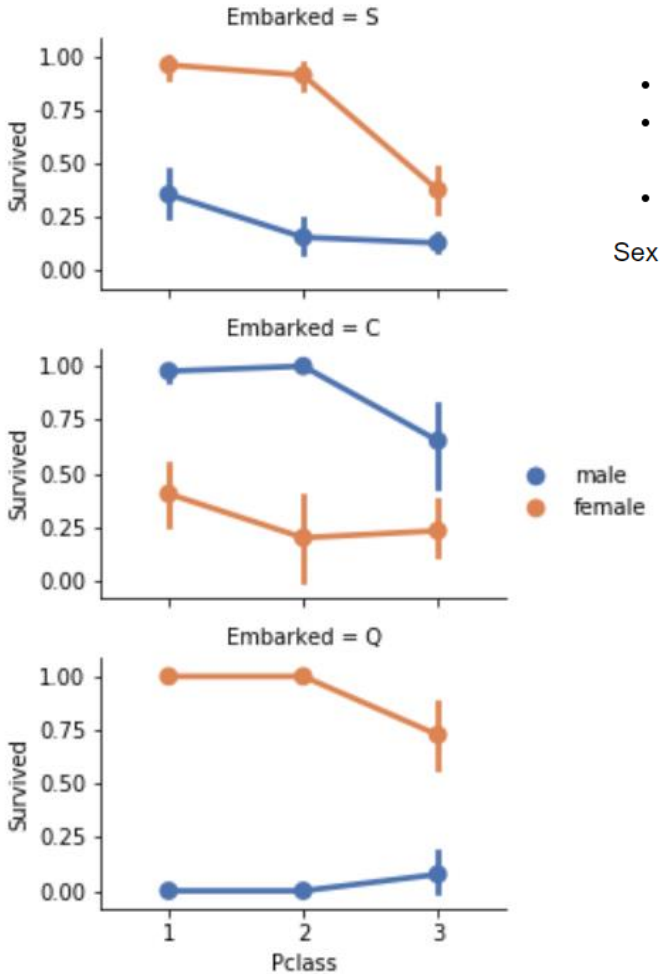
- Klasse 3 heeft ruim de meeste passagiers maar ook ruim het minst aantal overlevenden.
- Alle kinderen in klasse 2 hebben het overleeft. Een redelijk groot deel van de kinderen in klasse 3 heeft het ook overleefd. In klasse 1 kwamen bijna geen kinderen voor
- In klasse 1 zijn relatief de meeste overlevenden zoals ook 1 van onze aannames is
- We zien hier de invloed van leeftijd en klasse gecombineerd op de survival rate

Het lijkt er dus op dat we klasse kunnen gebruiken als feature voor ons classificatie probleem



Correlatie van categorische features

```
ordered_embarked = train_df["Embarked"].value_counts().index
grid = sns.FacetGrid(train_df, row='Embarked', height=2.2, aspect=1.6, row_order=ordered_embarked)
grid.map(sns.pointplot, 'Pclass', 'Survived', 'Sex', palette='deep')
grid.add_legend()
```



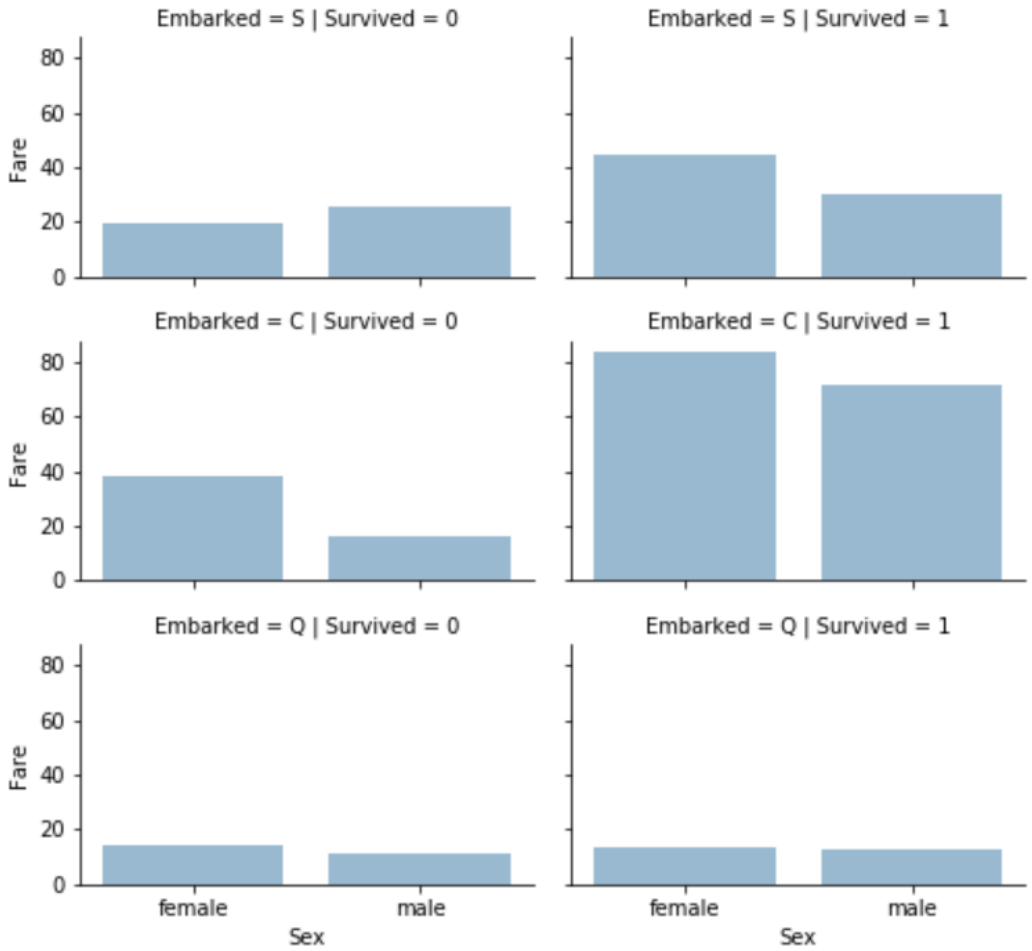
- Behalve voor de passagiers die in Cherbourg (C) zijn ingestapt is er voor vrouwen een hogere survival rate dan voor mannen
- Dat de survival rate voor mannen hoger is dan voor vrouwen voor Embarked = C kan duiden op een correlatie tussen klasse en embarked en tussen klasse en survived (wat we eerder ook al hadden gezien). Het toont niet direct een correlatie aan tussen embarked en survived.
- Mannelijke overlevingskansen voor klasse 3 in Embarked = Q is hoger dan in klasse 2 (voor Embarked = Q)

Sex kan als feature gebruikt worden en Embarked ook. Embarked moet echter wel worden aangevuld gezien de ontbrekende waardes

Correlatie van categorische en numerieke waardes ¶

We gaan nu de correlatie van Embarked (categorisch, non-numeriek), Fare (Numeriek, continue) en survival (categorisch numeriek):

```
grid = sns.FacetGrid(train_df, row='Embarked', col='Survived', height=2.2, aspect=1.6)
grid.map(sns.barplot, 'Sex', 'Fare', alpha=.5, ci=None)
grid.add_legend()
```



- Het is te zien dat een hogere Fare, behalve bij Embarked = Q, een hogere overlevingskans oplevert
- Ook hier is weer te zien dat de plaats van embarkment invloed heeft op de survival rate

Fare kunnen we meenemen als feature

Data correctie, features verwijderen, nieuwe features aanmaken

Als eerste gaan we de features verwijderen waarvan we hebben gezien dat ze geen of weinig bijdrage leveren aan de voorspelling voor overleving. Gezien de eerdere bevindingen gaan we de Cabin en Ticket features verwijderen, uiteraard op de train én test set:

```
train_df = train_df.drop(['Ticket', 'Cabin'], axis=1)
test_df = test_df.drop(['Ticket', 'Cabin'], axis=1)
combine = [train_df, test_df]
```

Aanmaken nieuwe features

Als eerste kijken we naar het extraheren van de titel uit het naam veld. Als onderdeel van deze oefening kijken we daarna uiteraard ook of er een correlatie tussen de nieuwe feature "title" en survival is te ontdekken.

```
for dataset in combine:
    dataset['Title'] = dataset.Name.str.extract(' ([A-Za-z]+)\.', expand=False)

pd.crosstab(train_df['Title'], train_df['Sex'])
```

Sex	female	male
Title		
Capt	0	1
Col	0	2
Countess	1	0
Don	0	1
Dr	1	6
Jonkheer	0	1
Lady	1	0
Major	0	2

```
for dataset in combine:
    dataset['Title'] = dataset['Title'].replace(['Lady', 'Countess','Capt', 'Col',\
 	'Don', 'Dr', 'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona'], 'Rare')

    dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')

train_df[['Title', 'Survived']].groupby(['Title'], as_index=False).mean()
```

	Title	Survived
0	Master	0.575000
1	Miss	0.702703
2	Mr	0.156673
3	Mrs	0.793651
4	Rare	0.347826

We kunnen zien dat sommige titels een goede voorspelling opleveren voor de survival rate (Miss/Mrs vs Mr).

Omzetten van de categorische titel waarden in ordinale waarden:

```
title_mapping = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Rare": 5}
for dataset in combine:
    dataset['Title'] = dataset['Title'].map(title_mapping)
    dataset['Title'] = dataset['Title'].fillna(0)

train_df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	7.2500	S	1
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	71.2833	C	3
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	7.9250	S	2
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	53.1000	S	3
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	8.0500	S	1

Nu het verwijderen van de laatste features welke we niet gaan gebruiken: Name en PassengerId

```
train_df = train_df.drop(['Name', 'PassengerId'], axis=1)
test_df = test_df.drop(['Name'], axis=1)
combine = [train_df, test_df]
train_df.shape, test_df.shape
```

```
((891, 9), (418, 9))
```

Omzetten van categorische waarden in numerieke waarden

Voor Sex gaan we male/female omzetten in 0/1.

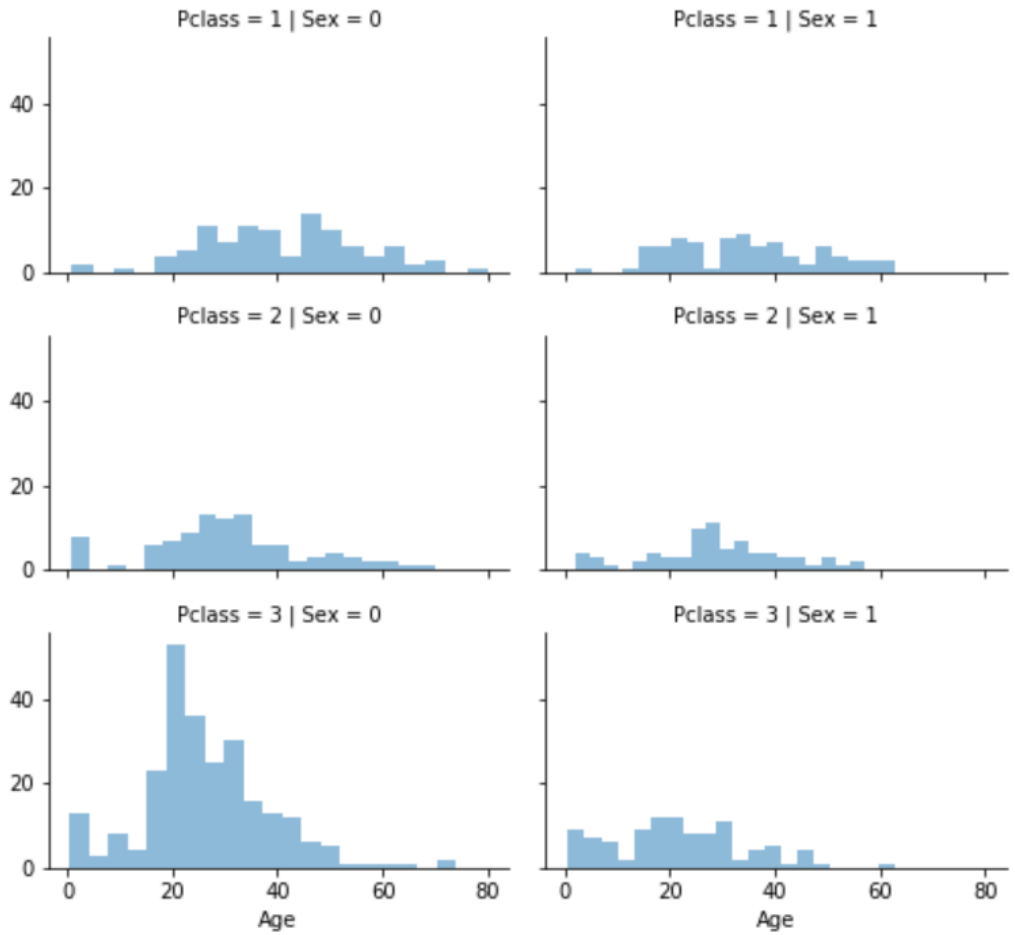
```
for dataset in combine:
    dataset['Sex'] = dataset['Sex'].map( {'female': 1, 'male': 0} ).astype(int)

train_df.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	0	3	0	22.0	1	0	7.2500	S	1
1	1	1	1	38.0	1	0	71.2833	C	3
2	1	3	1	26.0	0	0	7.9250	S	2
3	1	1	1	35.0	1	0	53.1000	S	3
4	0	3	0	35.0	0	0	8.0500	S	1

Aanvullen van ontbrekende waarden

Gezien niet alle features waardes hebben voor elke passagier, gaan we nu een aantal invullingen doen, beginnende met de age feature. Er zijn verschillende methoden voor om ontbrekende waarden aan te vullen, hier kiezen we voor het gebruiken van de correlatie welke we eerder hebben gezien tussen Age, Sex en Pclass

```
grid = sns.FacetGrid(train_df, row='Pclass', col='Sex', height=2.2, aspect=1.6)
grid.map(plt.hist, 'Age', alpha=.5, bins=20)
grid.add_legend()
```



```
guess_ages = np.zeros((2,3))
guess_ages
```

```
array([[0., 0., 0.],
```

```
for dataset in combine:
    for i in range(0, 2):
        for j in range(0, 3):
            guess_df = dataset[(dataset['Sex'] == i) & \
                               (dataset['Pclass'] == j+1)]['Age'].dropna()

            # age_mean = guess_df.mean()
            # age_std = guess_df.std()
            # age_guess = rnd.uniform(age_mean - age_std, age_mean + age_std)

            age_guess = guess_df.median()

            # Convert random age float to nearest .5 age
            guess_ages[i,j] = int( age_guess/0.5 + 0.5 ) * 0.5

    for i in range(0, 2):
        for j in range(0, 3):
            dataset.loc[ (dataset.Age.isnull()) & (dataset.Sex == i) & (dataset.Pclass == j+1),\
                    'Age'] = guess_ages[i,j]

    dataset['Age'] = dataset['Age'].astype(int)

train_df.head()
```

We gaan nu de Age feature omzetten in een nieuwe AgeBand feature waarbij de Age wordt ingedeeld in 5 leeftijdsgroepen:

```
train_df['AgeBand'] = pd.cut(train_df['Age'], 5)
train_df[['AgeBand', 'Survived']].groupby(['AgeBand'], as_index=False).mean().sort_values(by='AgeBand', ascending=True)
```

	AgeBand	Survived
0	(-0.08, 16.0]	0.550000
1	(16.0, 32.0]	0.337374
2	(32.0, 48.0]	0.412037
3	(48.0, 64.0]	0.434783
4	(64.0, 80.0]	0.090909

En gelijk Age omzetten in ordinale waarden:

```
for dataset in combine:
    dataset.loc[ dataset['Age'] <= 16, 'Age'] = 0
    dataset.loc[(dataset['Age'] > 16) & (dataset['Age'] <= 32), 'Age'] = 1
    dataset.loc[(dataset['Age'] > 32) & (dataset['Age'] <= 48), 'Age'] = 2
    dataset.loc[(dataset['Age'] > 48) & (dataset['Age'] <= 64), 'Age'] = 3
    dataset.loc[ dataset['Age'] > 64, 'Age']
train_df.head()
```

```
train_df = train_df.drop(['AgeBand'], axis=1)
combine = [train_df, test_df]
train_df.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	0	3	0	1	1	0	7.2500	S	1
1	1	1	1	2	1	0	71.2833	C	3
2	1	3	1	1	0	0	7.9250	S	2
3	1	1	1	2	1	0	53.1000	S	3
4	0	3	0	2	0	0	8.0500	S	1

Toevoegen van een nieuwe feature FamilySize

Het aantal familieleden wordt bepaald door de som van Parch en SibSp:

```
for dataset in combine:
    dataset['FamilySize'] = dataset['SibSp'] + dataset['Parch'] + 1

train_df[['FamilySize', 'Survived']].groupby(['FamilySize'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

	FamilySize	Survived
3	4	0.724138
2	3	0.578431
1	2	0.552795
6	7	0.333333
0	1	0.303538
4	5	0.200000
5	6	0.136364
7	8	0.000000
8	11	0.000000

Om het expliciet te maken voegen we ook een feature "IsAlone" toe om aan te geven of een persoon alleen op de boot zit of niet:

```
for dataset in combine:
    dataset['IsAlone'] = 0
    dataset.loc[dataset['FamilySize'] == 1, 'IsAlone'] = 1

train_df[['IsAlone', 'Survived']].groupby(['IsAlone'], as_index=False).mean()
```

	IsAlone	Survived
0	0	0.505650
1	1	0.303538

De nieuwe feature IsAlone gaan we gebruiken in plaats van FamilySize, Parch en SibSp. Deze laatste 2 gaan we dan ook uit de set verwijderen:

```
train_df = train_df.drop(['Parch', 'SibSp', 'FamilySize'], axis=1)
test_df = test_df.drop(['Parch', 'SibSp', 'FamilySize'], axis=1)
combine = [train_df, test_df]

train_df.head()
```

De laatste feature welke we gaan toevoegen is Age*Class

```
for dataset in combine:
    dataset['Age*Class'] = dataset.Age * dataset.Pclass

train_df.loc[:, ['Age*Class', 'Age', 'Pclass']].head(10)
```

	Age*Class	Age	Pclass
0	3	1	3
1	2	2	1
2	3	1	3
3	2	2	1
4	6	2	3
5	3	1	3
6	3	3	1
7	0	0	3
8	3	1	3
9	0	0	2

Aanvullen van Embarked

Ook Embarked heeft ontbrekende waarden. Gezien het om 2 records gaat, kiezen we hier voor een andere strategie om deze op te vullen, namelijk gewoon met de meest voorkomende waarden:

```
freq_port = train_df.Embarked.dropna().mode()[0]
freq_port
```

```
'S'
```

```
for dataset in combine:
    dataset['Embarked'] = dataset['Embarked'].fillna(freq_port)

train_df[['Embarked', 'Survived']].groupby(['Embarked'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

	Embarked	Survived
0	C	0.553571
1	Q	0.389610
2	S	0.339009

Nu alleen de categorische waarden in Embarked omzetten naar numeriek:

```
for dataset in combine:
    dataset['Embarked'] = dataset['Embarked'].map( {'S': 0, 'C': 1, 'Q': 2} ).astype(int)

train_df.head()
```

	Survived	Pclass	Sex	Age	Fare	Embarked	Title	IsAlone	Age*Class
0	0	3	0	1	7.2500	0	1	0	3
1	1	1	1	2	71.2833	1	3	0	2
2	1	3	1	1	7.9250	0	2	1	3
3	1	1	1	2	53.1000	0	3	0	2
4	0	3	0	2	8.0500	0	1	1	6

Aanvullen van Fare

Ook hier vullen we de ontbrekende waarde simpel aan, deze keer met de mediaan:

```
test_df['Fare'].fillna(test_df['Fare'].dropna().median(), inplace=True)
test_df.head()
```

	PassengerId	Pclass	Sex	Age	Fare	Embarked	Title	IsAlone	Age*Class
0	892	3	0	2	7.8292	2	1	1	6
1	893	3	1	2	7.0000	0	3	0	6
2	894	2	0	3	9.6875	2	1	1	6
3	895	3	0	1	8.6625	0	1	1	3
4	896	3	1	1	12.2875	0	3	0	3

Ook voor Fare gaan we een bandbreedte definieren:

```
train_df['FareBand'] = pd.qcut(train_df['Fare'], 4)
train_df[['FareBand', 'Survived']].groupby(['FareBand'], as_index=False).mean().sort_values(by='FareBand', ascending=True)
```

En dus ook weer omzetten naar numerieke waarden

```
for dataset in combine:
    dataset.loc[ dataset['Fare'] <= 7.91, 'Fare'] = 0
    dataset.loc[(dataset['Fare'] > 7.91) & (dataset['Fare'] <= 14.454), 'Fare'] = 1
    dataset.loc[(dataset['Fare'] > 14.454) & (dataset['Fare'] <= 31), 'Fare']   = 2
    dataset.loc[ dataset['Fare'] > 31, 'Fare'] = 3
    dataset['Fare'] = dataset['Fare'].astype(int)

train_df = train_df.drop(['FareBand'], axis=1)
combine = [train_df, test_df]

train_df.head(10)
```

	Survived	Pclass	Sex	Age	Fare	Embarked	Title	IsAlone	Age*Class
0	0	3	0	1	0	0	1	0	3
1	1	1	1	2	3	1	3	0	2
2	1	3	1	1	1	0	2	1	3

En ook de test set:

```
test_df.head(10)
```

	PassengerId	Pclass	Sex	Age	Fare	Embarked	Title	IsAlone	Age*Class
0	892	3	0	2	0	2	1	1	6
1	893	3	1	2	0	0	3	0	6
2	894	2	0	3	1	2	1	1	6

Voorspellen!

Het heeft even geduurd maar eindelijk zijn de datasets klaar om te gebruiken voor voorspelling.

```
X_train = train_df.drop("Survived", axis=1)
Y_train = train_df["Survived"]
X_test  = test_df.drop("PassengerId", axis=1).copy()
X_train.shape, Y_train.shape, X_test.shape
```

```
((891, 8), (891,), (418, 8))
```

```
svc = SVC(kernel="rbf", gamma="auto")
svc.fit(X_train, Y_train)
Y_train_pred=svc.predict(X_train)
Y_pred = svc.predict(X_test)
acc_svc = round(svc.score(X_train, Y_train) * 100, 2)
print (acc_svc)
```