Assignment 2: Gaussian Classifier, Bias-Variance Decomposition, Evaluation Measures

Automatic Testing Guidelines

Automatic unittesting requires you, as a student, to submit a notebook which contains strictly defined objects. Strictness of definition consists of unified shapes, dtypes, variable names, and more.

Within the notebook, we provide detailed instructions which you should follow in order to maximize your final grade. Please keep in mind:

- Don't add any cells but use the ones provided by us. You may notice that most cells are tagged such that the unittest routine can recognise them.
- We highly recommend you to develop your code within the provided cells. You can implement helper functions where needed unless you put them in the same cell they are actually called. Always make sure that implemented functions have the correct output and given variables contain the correct data type. Don't import any other packages than listed in the cell with the "imports" tag.
- Never use variables you defined in another cell in your functions directly; always pass them to the function as a parameter. In the unittest they won't be available either.

Good luck! :)

Task 1: Gaussian classifier: visualization & parameter estimation (10 points)

The goal of this task is to explore the given (artificial) data before diving into the classification function. To do this, we will use `matplotlib` to plot the data set and `numpy` to estimate the means & covariance matrices of the classes as well as the probability of encountering a positive/negative example.

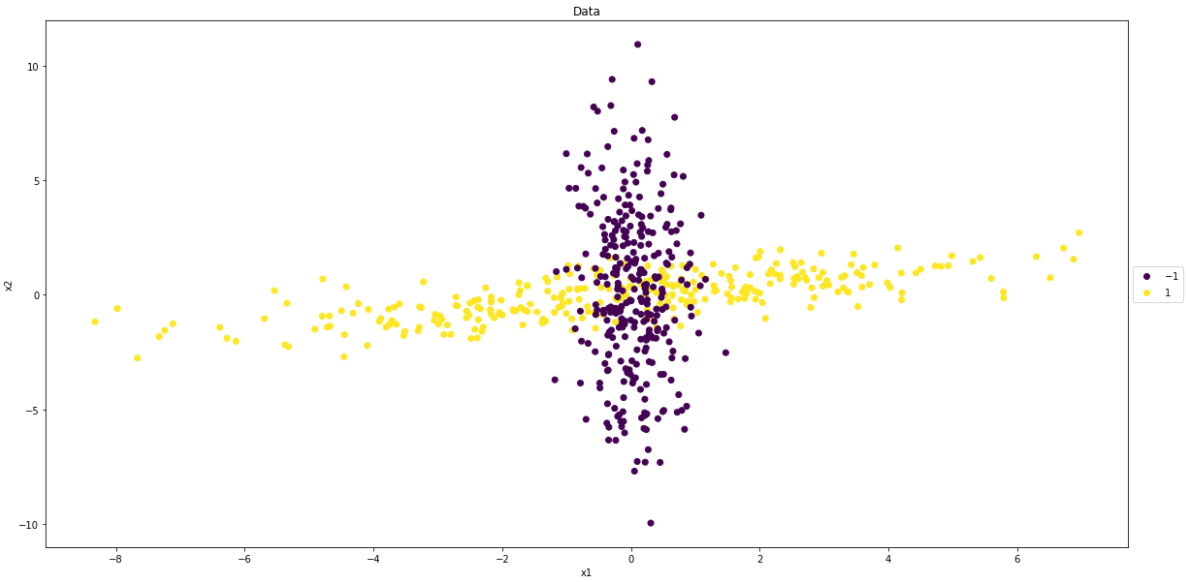
- **Task 1.1**: Visualize the data stored in `normal.csv` with two different colors using a scatter plot and store it in the given variable. Always label the axes of all your plots.
- **Task 1.2**: We assume that the data is distributed according to a two-dimensional multivariate normal distribution:

- Write a function that estimates the means and covariance matrices of each class as well as the distributions $p(y=+1)$ and $p(y=-1)$
- Return a tuple containing the results (the resulting list should be of length 6). The datatype for `covXpos`, `covXneg`, `meanXpos` and `meanXneg` should be a numpy array, for $p(y=+1)$ and $p(y=-1)$ it should be float.

1.1. Code & question (4 points):

In [31]:
```
import sklearn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from mpl_toolkits.mplot3d import Axes3D
import random
%matplotlib inline
```

In [32]:
```
# read data, split into X (features) and y (labels)
Z = np.genfromtxt('normal.csv', delimiter=',')
X, y = Z[:,:-1], Z[:,-1]

# your code for the visualization

# example: plt.figure...
# plt.plot...

plt.rcParams['figure.figsize'] = (20, 10)
fig1 = plt.figure()
scatter = plt.scatter(X[:, 0],X[:, 1], c=y)
plt.xlabel("x1")
plt.ylabel("x2")

plt.title("Data")
plt.legend(handles=scatter.legend_elements()[0], labels =scatter.legend_elements()
plt.show()
```



Answer the following yes/no questions concerning the distribution of the data:

a) *Would a linear regression method be reasonable for this task?*

b) Would a linear classifier roughly achieve a better performance than 33% misclassification?

To answer the question, assign "True" or "False" boolean values to variables in the next cell. A non-correctly answered question yields negative points and no answer (i.e. answer "None") gives 0 points for a question. More details on grading can be found in the FAQ sheet.

Note: Do not reuse these variable names. They are used for testing.

In [33]:
```
# examples for you
example_of_true_variable = True
example_of_false_variable = False

# your answers go here ↓↓↓
a_=False
b_=True
```

1.2. Code (6 points):

In [34]:
```
"""
Function that estimates the means and covariance matrices from the given data as we
a positive/negative example respectively
@param X_, np ndarray, data matrix
@param y_, np ndarray, data vector
"""
def est_mean_cov(X_,y_):
    # replace the following line with your lines of code
    covXpos = np.cov(X_[y_==1].T)
    meanXpos = np.mean(X_[y_==1], axis=0)
    p_ypos = len(X_[y_==1])/len(X_)
    covXneg = np.cov(X_[y_==-1].T)
    meanXneg = np.mean(X_[y_==-1], axis=0)
    p_yneg = len(X_[y_==-1])/len(X_)

    return covXpos, meanXpos, p_ypos, covXneg, meanXneg, p_yneg

covXpos, meanXpos, p_ypos, covXneg, meanXneg, p_yneg = est_mean_cov(X,y)

# print corresponding values
print("Positive class (blue):\n")
print("Covariance:")
print(pd.DataFrame(covXpos,columns=["x1","x2"],index=["x1","x2"]),"\n")
print("Mean = ", meanXpos, "\n")
print("p(y=+1) =", p_ypos, "\n\n")
print("Negative class (orange):\n")
print("Covariance:")
print(pd.DataFrame(covXneg,columns=["x1","x2"],index=["x1","x2"]),"\n")
print("Mean =", meanXneg, "\n")
print("p(y=-1) =", p_yneg, "\n")
```

```
Positive class (blue):

Covariance:
          x1        x2
x1  8.442267  2.017815
x2  2.017815  0.953326

Mean =  [-0.14820707 -0.03790113]

p(y=+1) = 0.5


Negative class (orange):

Covariance:
          x1         x2
x1  0.198607  -0.161064
x2 -0.161064  12.553366

Mean = [0.0340324  0.16727881]

p(y=-1) = 0.5
```

Task 2: Gaussian classifier: compute classifier & visualization (20 points)

Now that we (hopefully) get a good idea of the data, we want to implement a classifier and show its effects using a plot.

- **Task 2.1**: Compute an optimal classification function g in `calc_func_g()` (see slide "Explicit example: Gaussian classifier: Part 2" from lecture Unit2.pdf). To do this, you should:
  - Calculate the values of the corresponding parameters $\mathbf{A}$, $\mathbf{b}$ and c in the provided functions.
  - Store the results in the given parameters **par_A** (np.array), **par_b** (np.array), **par_c** (float), and **func_g** (np.array).
  - Print the values of $\mathbf{A}$, $\mathbf{b}$ and c that you have calculated with their respective shapes.
  - Note: You can reuse the results from the previous exercise here.
- **Task 2.2**: Visualize the classification function and the decision boundaries including the original points from Task 1.1. in **one** two-dimensional plot.

2.1 Code (10 points):

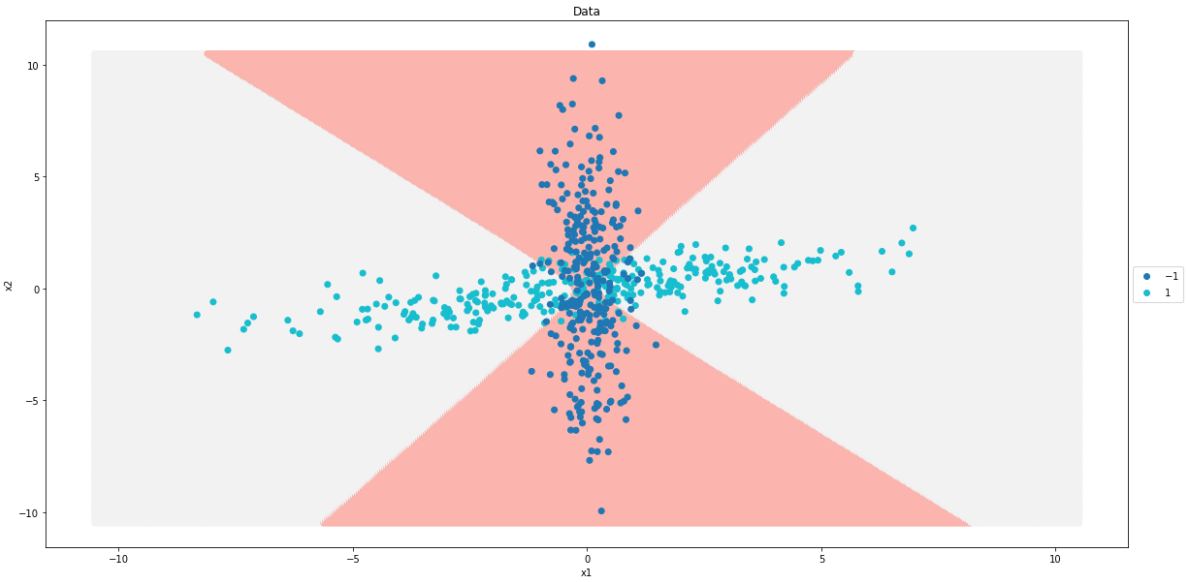
In [35]:
```
"""
These functions should contain the calculations for the respective parameters and r
@param covXpos, np ndarray, covariance matrix of positive examples
@param meanXpos, np ndarray, mean of positive examples
@param covXneg, np ndarray, covariance matrix of negativ examples
@param meanXneg, np ndarray, mean of negative examples
@param p_ypos, float, probability of encountering a positive example
@param p_yneg, float, probability of encountering a negative example
Hint: You may want to check out np.linalg.inv
```

```
"""
def calc_par_A(covXpos, meanXpos, covXneg, meanXneg, p_ypos, p_yneg):

    # replace the following line with your lines of code
    return np.linalg.inv(covXpos)-np.linalg.inv(covXneg)


def calc_par_b(covXpos, meanXpos, covXneg, meanXneg, p_ypos, p_yneg):

    # replace the following line with your lines of code
    return np.dot(np.linalg.inv(covXpos),meanXpos)-np.dot(np.linalg.inv(covXneg),me

def calc_par_c(covXpos, meanXpos, covXneg, meanXneg, p_ypos, p_yneg):

    # replace the following line with your lines of code
    return -1/2*meanXpos.T@np.linalg.inv(covXpos) @ meanXpos+ 1/2*meanXneg.T@np.lin


"""
Combine the previously calculated parameters to the optimal classification functior
@param points, np.array, the points that the function g should be applied to
"""
def calc_func_g(par_A, par_b, par_c, points):

    # replace the following line with your lines of code
    pred=[]
    for row in points:
        pred.append(np.sign(-1/2 * row.T @ par_A @ row + par_b.T @row + par_c))
    return pred

# some code that should help you
X1, X2 = np.mgrid[-10.5:10.5:500j, -10.5:10.5:500j]
points = np.c_[X1.ravel(), X2.ravel()]

par_A = calc_par_A(covXpos, meanXpos, covXneg, meanXneg, p_ypos, p_yneg)
par_b = calc_par_b(covXpos, meanXpos, covXneg, meanXneg, p_ypos, p_yneg)
par_c = calc_par_c(covXpos, meanXpos, covXneg, meanXneg, p_ypos, p_yneg)
func_g = calc_func_g(par_A, par_b, par_c, points)

# print the values shapes of par_A, par_b, and par_c here
print(f"Par_A: value= {par_A}, shape= {par_A.shape}")
print(f"Par_b: value= {par_b}, shape= {par_b.shape}")
print(f"Par_c: value= {par_c}, shape= {par_c.shape}")
```

```
Par_A: value= [[-4.84827549 -0.57269559]
 [-0.57269559  2.04245844]], shape= (2, 2)
Par_b: value= [-0.2003753  -0.02094713], shape= (2,)
Par_c: value= -0.23553240713102885, shape= ()
```

2.2 Code & question (10 points):

In [36]:
```
# your code for the visualization

plt.rcParams['figure.figsize'] = (20, 10)
fig2 = plt.figure()
plt.scatter(points[:, 0],points[:, 1], c=func_g, cmap='Pastel1')
scatter2=plt.scatter(X[:, 0],X[:, 1], c=y, cmap='tab10')
plt.xlabel("x1")
plt.ylabel("x2")

plt.title("Data")
plt.legend(handles=scatter2.legend_elements()[0], labels =scatter2.legend_elements
plt.show()
```



Answer the following questions about the plot you just created:

c) Did the classifier perform well on the task i.e. do the decision boundaries seem to match the classes as plotted in Task 1.1?
d) Are datapoints that lie in the middle, overlapping region of the two classes more prone to being misclassified?

To answer the question, assign "True" or "False" boolean values to variables in the next cell. A non-correctly answered question yields negative points and no answer (i.e. answer "None") gives 0 points for a question. More details on grading can be found in the FAQ sheet.
Note: Do not reuse these variable names. They are used for testing.

In [37]:
```
# your answers go here ↓↓↓
c_=True
d_=True
```

Task 3: Details for bias-variance decomposition for quadratic loss (15 points)

An explicit formula of the bias variance decomposition for the quadratic loss was mentioned in the lecture. In this task, you will be asked to fill in some details that haven't been discussed there. To this end, let us introduce some notation:

Z_l denotes a data matrix of l elements from our data set $Z =(X,\mathbf{y})$ with X the feature matrix and $\mathbf{y}$ the label vector. $g(\mathbf{x}_0;\mathbf{w}(Z_l)))$ denotes the model, with a parameter vector $\mathbf{w}(Z_l)$ originating from Z_l, and y is the label corresponding to a feature vector $\mathbf{x}_0$.

Our object of interest is the expected prediction error (EPE) for $\mathbf{x}_0\in X$ in case of the quadratic loss, i.e.:

$$\mathrm{EPE}(\mathbf{x}_0) = \mathrm{E}_{y\mid \mathbf{x}_0,Z_l}\big(L_{\mathbf{q}}(y,g(\mathbf{x}_0;\mathbf{w}(Z_l)))\big) = \mathrm{E}_{y\mid \mathbf{x}_0,Z_l}\big((y-g(\mathbf{x}_0;\mathbf{w}(Z_l)))^2\big)$$

We assume that $y\!\mid\!\mathbf{x}_0$ and the selection of training samples Z_l are independent which results in the following reformulation of the total expected prediction error:

$$\mathrm{EPE}(\mathbf{x}_0) = \mathrm{E}_{y\mid \mathbf{x}_0}\Big(\mathrm{E}_{Z_l}\big((y-g(\mathbf{x}_0;\mathbf{w}(Z_l)))^2\big)\Big)\quad \text{(1)}$$

Show that we can obtain the following bias-variance decomposition:

$$\begin{aligned} \mathrm{EPE}(\mathbf{x}_0)=&\,\operatorname{Var}(y\!\mid\!\mathbf{x}_0)\\ &+\Big(\mathrm{E}_{y\mid\mathbf{x}_0}(y)-E_{Z_l}\big(g(\mathbf{x}_0;\mathbf{w}(Z_l))\big)\Big)^2\\ &+\mathrm{E}_{Z_l}\Big(\big(g(\mathbf{x}_0;\mathbf{w}(Z_l))-E_{Z_l}(g(\mathbf{x}_0;\mathbf{w}(Z_l)))\big)^2\Big) \end{aligned}$$

For your calculation please use the given notation. Follow the steps indicated below.

3.1 Calculation (5 points): Expand the Expected Prediction Error.

Expand $\mathrm{EPE}(\mathbf{x}_0)$, i.e. eq. (1) above, and write it as three separate terms.

$$\mathrm{EPE}(\mathbf{x}_0) = \mathrm{E}_{y\mid \mathbf{x}_0}\Big(\mathrm{E}_{Z_l}\big((y-g(\mathbf{x}_0;\mathbf{w}(Z_l)))^2\big)\Big)$$

$$\mathrm{EPE}(\mathbf{x}_0) =\mathrm{E}_{y\mid \mathbf{x}_0}\Big(y^2-2y\mathrm{E}_{Z_l}\big(g(\mathbf{x}_0;\mathbf{w}(Z_l))\big)+\mathrm{E}_{Z_l}\big(g(\mathbf{x}_0;\mathbf{w}(Z_l))^2\big)\Big)$$

$$\mathrm{EPE}(\mathbf{x}_0) =\mathrm{E}_{y\mid\mathbf{x}_0}\Big(y^2\Big)-\mathrm{E}_{y\mid\mathbf{x}_0}\Big(y\Big)^2 +\mathrm{E}_{y\mid\mathbf{x}_0}\Big(y\Big)^2 -2\mathrm{E}_{y\mid\mathbf{x}_0}\Big(y\Big)\mathrm{E}_{Z_l}\Big(g(\mathbf{x}_0;\mathbf{w}(Z_l))\Big)+\mathrm{E}_{Z_l}\Big(g(\mathbf{x}_0;\mathbf{w}(Z_l))^2\big)\Big)$$

$$\mathrm{EPE}(\mathbf{x}_0) = \mathrm{Var}_{y\mid\mathbf{x}_0}\Big(y\Big)+\mathrm{E}_{y\mid\mathbf{x}_0}\Big(y\Big)^2 -2\mathrm{E}_{y\mid\mathbf{x}_0}\Big(y\Big)\mathrm{E}_{Z_l}\Big(g(\mathbf{x}_0;\mathbf{w}(Z_l))\Big)+ \mathrm{E}_{Z_l}\Big(g(\mathbf{x}_0;\mathbf{w}(Z_l))\big)\Big)^2+\mathrm{E}_{Z_l}\Big(g(\mathbf{x}_0;\mathbf{w}(Z_l))^2\Big) - \mathrm{E}_{Z_l}\Big(g(\mathbf{x}_0;\mathbf{w}(Z_l))\Big)^2$$

$$\mathrm{EPE}(\mathbf{x}_0) = \mathrm{Var}_{y\mid\mathbf{x}_0}\Big(y\Big)+\Big(\mathrm{E}_{y\mid\mathbf{x}_0}\big(y\big) - \mathrm{E}_{Z_l}\big(g(\mathbf{x}_0;\mathbf{w}(Z_l))\big)\Big)^2 +\mathrm{E}_{Z_l}\Big(\big(g(\mathbf{x}_0;\mathbf{w}(Z_l)) - \mathrm{E}_{Z_l}\big(g(\mathbf{x}_0;\mathbf{w}(Z_l))\big)^2\Big)$$

3.2 Calculation (2 points): Rewrite $\operatorname{Var}(y\!\mid\!\mathbf{x}_0)$ using expected values.

Write the variance in terms of expectation values.

$\mathrm{Var}_{y\mid\mathbf{x}_0}\Big(y\Big) =
\mathrm{E}_{y\mid\mathbf{x}_0}\Big(y^2\Big)-
\mathrm{E}_{y\mid\mathbf{x}_0}\Big(y\Big)^2$

3.3. Calculation (3 points): Expand the squared bias.

Expand the squared bias and write it in three separate terms.

$ \mathrm{Bias}^2 = \Big(\mathrm{E}_{y\mid\mathbf{x}_0}\big(y\big) -
\mathrm{E}_{Z_l}\big(g(\mathbf{x}_0;\mathbf{w}(Z_l))\big)\Big)^2 $

$ \Big(\mathrm{E}_{y\mid\mathbf{x}_0}\big(y\big) -
\mathrm{E}_{Z_l}\big(g(\mathbf{x}_0;\mathbf{w}(Z_l))\big)\Big)^2
=\mathrm{E}_{y\mid\mathbf{x}_0}\Big(y\Big)^2
-2\mathrm{E}_{y\mid\mathbf{x}_0}\Big(y\Big)\mathrm{E}_{Z_l}\Big(g(\mathbf{x}_0;\mathbf{w}
(Z_l))\Big)+ \mathrm{E}_{Z_l}\Big(g(\mathbf{x}_0;\mathbf{w}(Z_l))\big)\Big)^2$

3.4 Calculation (5 points): Expand the variance of the model.

Write the model variance in (first three, then simplified to two) separate terms.

(Eventually, you can see that adding up 3.2., 3.3., and 3.4., some terms cancel and exactly 3.1. remains.)

$\mathrm{Var}_{Z_l}\Big(g(\mathbf{x}_0;\mathbf{w}(Z_l))\Big) =
\mathrm{E}_{Z_l}\Big(\big(g(\mathbf{x}_0;\mathbf{w}(Z_l)) -
\mathrm{E}_{Z_l}\big(g(\mathbf{x}_0;\mathbf{w}(Z_l))\big)\big)^2\Big)$

$\mathrm{Var}_{Z_l}\Big(g(\mathbf{x}_0;\mathbf{w}(Z_l))\Big) =
\mathrm{E}_{Z_l}\Big(g(\mathbf{x}_0;\mathbf{w}(Z_l))^2 - 2g(\mathbf{x}_0;\mathbf{w}
(Z_l))\mathrm{E}_{Z_l}\big(g(\mathbf{x}_0;\mathbf{w}(Z_l))\big)
+\mathrm{E}_{Z_l}\big(g(\mathbf{x}_0;\mathbf{w}(Z_l))\big)^2\Big)$

$ \mathrm{Var}_{Z_l}\Big(g(\mathbf{x}_0;\mathbf{w}(Z_l))\Big) =
\mathrm{E}_{Z_l}\Big(g(\mathbf{x}_0;\mathbf{w}(Z_l))^2\Big) -
2\mathrm{E}_{Z_l}\Big(g(\mathbf{x}_0;\mathbf{w}
(Z_l))\Big)\mathrm{E}_{Z_l}\Big(g(\mathbf{x}_0;\mathbf{w}(Z_l))\Big)
+\mathrm{E}_{Z_l}\Big(g(\mathbf{x}_0;\mathbf{w}(Z_l))\Big)^2$

$ \mathrm{Var}_{Z_l}\Big(g(\mathbf{x}_0;\mathbf{w}(Z_l))\Big) =
\mathrm{E}_{Z_l}\Big(g(\mathbf{x}_0;\mathbf{w}(Z_l))^2\Big)
-2+\mathrm{E}_{Z_l}\Big(g(\mathbf{x}_0;\mathbf{w}(Z_l))\Big)^2
+\mathrm{E}_{Z_l}\Big(g(\mathbf{x}_0;\mathbf{w}(Z_l))\Big)^2$

$\mathrm{Var}_{Z_l}\Big(g(\mathbf{x}_0;\mathbf{w}(Z_l))\Big) =
\mathrm{E}_{Z_l}\Big(g(\mathbf{x}_0;\mathbf{w}(Z_l))^2\Big) -
\mathrm{E}_{Z_l}\Big(g(\mathbf{x}_0;\mathbf{w}(Z_l))\Big)^2$

Task 4: Bias-variance decomposition for regression (40 points)

4.1 Question (10 points):

Consider the following one-dimensional regression task: inputs x are sampled from the uniform distribution in $[−1, 3] \subset \mathbb{R}$ and targets y are given as

\begin{align*} f(x) &= 0.6\,x^4 + 2\,x^3 - 8\,x^2 \\ y &= f(x) + \varepsilon, \end{align*}
where ε is independent normally distributed noise with $\mu=0$ and $\sigma^2 = 0.09$.

What are $E(y\!\mid\!x_0)$ and the unavoidable error $\operatorname{Var}(y\!\mid\!x_0)$ for a fixed x_0 in this setting?

e_) $E(y\!\mid\!x_0) = 0.6\,\sigma^4 + 2\,\sigma^3 - 8\,\sigma^2 \text{ and } \operatorname{Var}(y\!\mid\!x_0) = x_0^2$.
f_) $E(y\!\mid\!x_0) = 0.6\,x_0^4 + 2\,x_0^3 - 8\,x_0^2 \text{ and }\operatorname{Var}(y\!\mid\!x_0) = \sigma^2$.
g_) $E(y\!\mid\!x_0) = 0.6\,\sigma^4 + 2\,\sigma^3 - 8\,\sigma^2 \text{ and } \operatorname{Var}(y\!\mid\!x_0) = \sigma^2$.
h_) $E(y\!\mid\!x_0) = 0.6\,x_0^4 + 2\,x_0^3 - 8\,x_0^2 \text{ and } \operatorname{Var}(y\!\mid\!x_0) = 0.6\,x_0^4 + 2\,x_0^3 - 8\,x_0^2+\sigma^2$.

To answer the question, assign "True" or "False" boolean values to variables in the next cell. A non-correctly answered question yields negative points and no answer (i.e. answer “None”) gives 0 points for a question. More details on grading can be found in the FAQ sheet.
Note: Do not reuse these variable names. They are used for testing.

In [38]:
```
# your answers go here ↓↓↓
e_=False
f_=True
g_=False
h_=False
```

We intend to perform polynomial regression to illustrate the bias-variance decomposition for the regression task described before. To this end, perform the following steps:

- **Task 4.2**:
  - Implement the function `create_train_X` which should return $k=200$ training sets with $l=20$ samples in the form of a numpy array.
  - Implement the function `create_train_y` according to the function described at the beginning of this task.
  - Below, we provide the code for a function that trains a polynomial regression model with degree m on a given training set and returns the prediction for a given test set. Use this to implement the function `bias_var` that estimates for each degree $m=1,...,11$ the squared bias and the variance from the predictions for each of the $k=200$ training sets at $x_0=1.8$ and stores them in the lists

sqbias and variance (which are already initiated as empty lists). Each of these two lists should then only contain 11 elements.

- **Task 4.3**:
  - Utilize the function `pol_reg_pred` to produce *one* plot that simultaneously visualizes the training data as dots (plot only the *first* instance of the k training sets, i.e. the 20 points from the first set) and the corresponding models for $m=1,3,11$. Don't forget to label the axes. Note: Make sure to produce the plot in the correct (second) cell.
  - Finally, visualize your results in *one* plot where the dependence of the variance and squared bias versus m is shown. Again, the axes should be labeled appropriately.

4.2 Code (15 points):

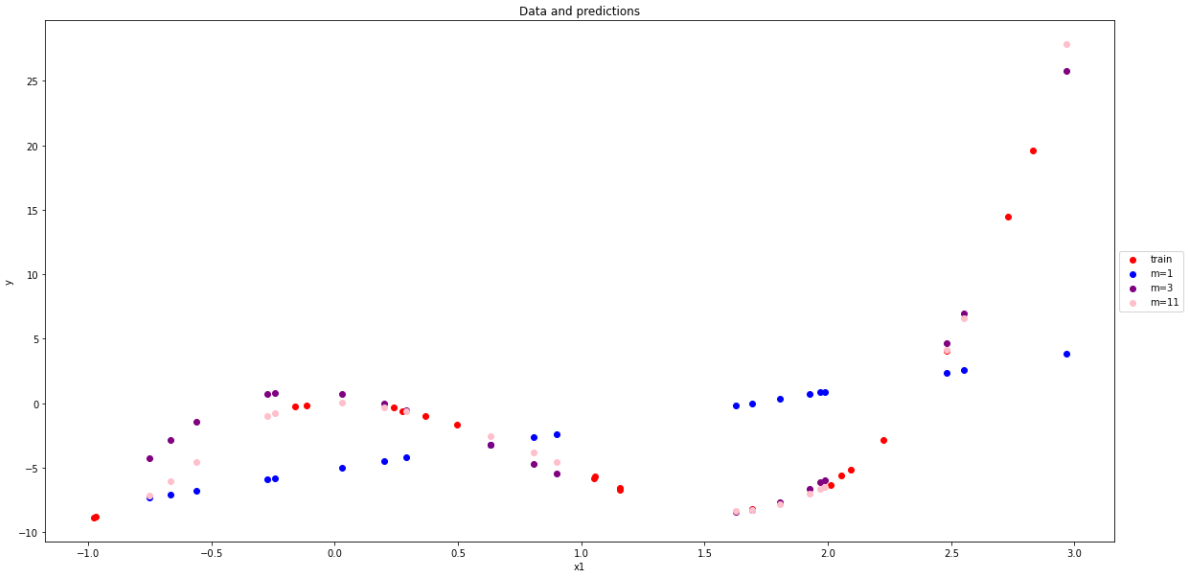
In [39]:
```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

# some code that should help you
# do not change the seed
np.random.seed(14)

def pol_reg_pred(X_train,y_train,X_test,m):
    poly_reg = PolynomialFeatures(m)
    X_poly_train = poly_reg.fit_transform(X_train.reshape(-1, 1))
    X_poly_test= poly_reg.fit_transform(X_test.reshape(-1, 1))
    lin_reg = LinearRegression()
    lin_reg.fit(X_poly_train, y_train)
    y_pred = lin_reg.predict(X_poly_test)
    return y_pred

def f(x):
    return 0.6 * x**4 + 2 * x**3 - 8 * x**2

def create_train_X(k,l):

    # replace the following line with your lines of code
    Xtrain_samples=[]
    for i in range(1,k+1):
        Xtrain_samples.append(np.random.uniform(-1,3,  size=l).tolist())
    return np.array(Xtrain_samples)

def create_train_y(k,l,X_train):

    # replace the following line with your lines of code
    ytrain_samples =[]
    for i in range(1,k+1):
        ytrain_samples.append(np.random.normal(0,0.09,  size=l).tolist())
    np.array(ytrain_samples)
    return ytrain_samples + f(X_train)

k = 200
l = 20
M = 11
X_train = create_train_X(k,l)
y_train = create_train_y(k,l,X_train)

def bias_var(X_train,y_train):
    x0 = np.array([1.8])
    sqbias = []
    variance = []
```

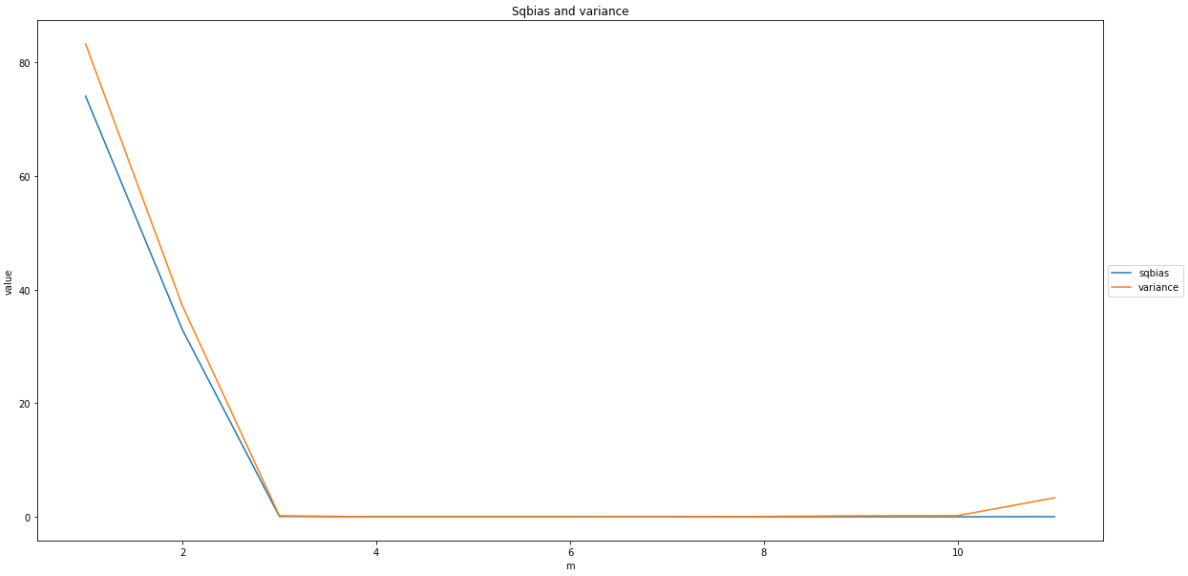
```
    # replace the following line with your lines of code
    for i in range(1,M+1):
        pred =[]
        for j in range(0,len(X_train)):
            pred.append(pol_reg_pred(X_train[j],y_train[j],x0,i))
        variance.append(np.sum((pred-f(x0))**2)/200)
        pred = np.mean(pred)
        sqbias.append((pred-f(x0))**2)


    return (sqbias,variance)

sqbias, variance = bias_var(X_train,y_train)
print(X_train.shape,y_train.shape)
print(sqbias)
print(variance)
```

```
(200, 20) (200, 20)
[array([74.03235701]), array([32.89593601]), array([0.06724866]), array([1.3878766
2e-05]), array([3.96819464e-05]), array([0.00011991]), array([0.00045402]), array
([5.87172525e-05]), array([0.00059329]), array([1.61673804e-06]), array([0.0198052
7])]
[83.21758168743295, 37.092733228866166, 0.18952190840395441, 0.001685498145524884
1, 0.002309966313801731, 0.004518910658752626, 0.019801599877171658, 0.06450500412
076084, 0.18387122830700495, 0.21264983987255262, 3.340291772156275]
```

4.3 Code (10 points):

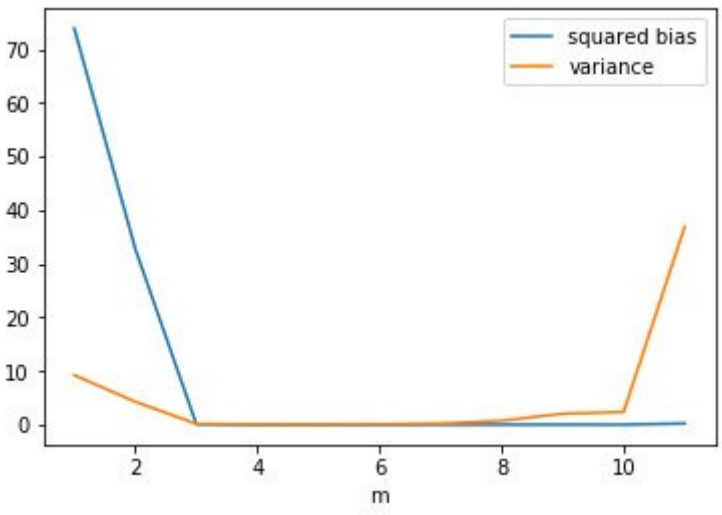
In [40]:
```
# your code for the visualization (remember, you need to create two plots for this
plt.rcParams['figure.figsize'] = (20, 10)
fig3 = plt.figure()
plt.scatter(X_train[0],y_train[0], c='red', label='train')
scatter2=plt.scatter(X_train[1],pol_reg_pred(X_train[0],y_train[0],X_train[1],1), 
scatter3=plt.scatter(X_train[1],pol_reg_pred(X_train[0],y_train[0],X_train[1],3), 
scatter4=plt.scatter(X_train[1],pol_reg_pred(X_train[0],y_train[0],X_train[1],11),

plt.xlabel("x1")
plt.ylabel("y")

plt.title("Data and predictions")
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()

m = 11
k=range(1,m+1)

fig4 = plt.figure()
plt.plot(k,sqbias, label = "sqbias")
plt.plot(k, variance, label = "variance")

plt.xlabel("m")
plt.ylabel("value")

plt.title("Sqbias and variance")
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```





4.4 Question (5 points):

If you did the previous task correctly, the resulting plot should look like this:



What observations can you make from this plot? Tick the correct boxes (several may be correct):

i) *The variance is lowest for models which are too simple, i.e. $m<3$.*
j) For appropriate complexity, i.e. $3 \leq m < 7$, both model variance and bias are low, which indicates good generalization abilities.
k) *As the model becomes too complex, i.e. $m \geq 7$, the variance increases again while the bias still decreases. This is an indication for underfitting.*
l) For models with $m \geq 7$, the variance is high (i.e. significantly larger than 0) because the independent noise has zero mean and high individual biases cancel in expectation.
m_) For models with $m \geq 7$, the bias is still low (i.e. close to 0) because the independent noise has zero mean and high individual biases cancel in expectation.

To answer the question, assign "True" or "False" boolean values to variables in the next cell. A non-correctly answered question yields negative points and no answer (i.e. answer "None") gives 0 points for a question. More details on grading can be found in the FAQ sheet.
Note: Do not reuse these variable names. They are used for testing.

In [41]:
```
# your answers go here ↓↓↓
i_=False
j_=True
k_=False
l_=True
m_=True
```

Task 5: Evaluation metrics for imbalanced data sets (15 points)

Consider a classifier with discriminant function $\bar g$. For a given labeled data set, the following results are obtained: $$\begin{array}{|r|r|} \hline y & \bar g(x)\\ \hline \hline +1 & 0.93 \\ +1 & 0.51 \\ +1 & 0.48 \\ -1 & 0.13 \\ +1 & 0.02 \\ -1 & -0.11 \\ -1 & -0.25 \\ -1 & -0.37 \\ +1 & -0.41 \\ -1 & -1.68 \\ +1 & -2.23 \\ \hline \end{array}$$

- **Task 5.1**: Compute the confusion matrix using the usual zero threshold. Complete the given function to calculate the following evaluation measures: ACC, TPR, TNR, FPR, FNR, PREC, and F_1, and store the exact results in the respective variables.

5.1 Calculation (8 points):

- **Reminder:** Confusion Matrix structure:

$$ \begin{array}{|c|c|c|} \hline &\bar g(x)=+1 & \bar g(x)=-1\\ \hline y=+1 & \text{TP} & \text{FN} \\ \hline y=-1 & \text{FP} & \text{TN} \\ \hline \end{array} $$

In [42]:
```
# confusion matrix
_TP = 0
_TN = 0
_FP = 0
_FN = 0

# evaluation measures
def evaluate_measures(TP,TN,FP,FN):
```

```
    # replace the following line with your lines of code
    raise NotImplementedError("You have not implemented this function")

    return (ACC,TPR,TNR,FPR,FNR,PREC,BACC,F1)

_ACC,_TPR,_TNR,_FPR,_FNR,_PREC,_BACC,_F1 = evaluate_measures(_TP,_TN,_FP,_FN)

print("ACC: {:.3f}\nTPR: {:.3f}\nTNR: {:.3f}\nFPR: {:.3f}\nFNR: {:.3f}\nPREC: {:.3
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
Input In [42], in <cell line: 15>()
     11     raise NotImplementedError("You have not implemented this function")
     13     return (ACC,TPR,TNR,FPR,FNR,PREC,BACC,F1)
---> 15 _ACC,_TPR,_TNR,_FPR,_FNR,_PREC,_BACC,_F1 = evaluate_measures(_TP,_TN,_FP,_
FN)
     17 print("ACC: {:.3f}\nTPR: {:.3f}\nTNR: {:.3f}\nFPR: {:.3f}\nFNR: {:.3f}\nPR
EC: {:.3f}\nBACC: {:.3f}\nF1: {:.3f}".format(_ACC,_TPR,_TNR,_FPR,_FNR,_PREC,_BACC,
_F1))

Input In [42], in evaluate_measures(TP, TN, FP, FN)
      8 def evaluate_measures(TP,TN,FP,FN):
      9
     10     # replace the following line with your lines of code
---> 11     raise NotImplementedError("You have not implemented this function")
     13     return (ACC,TPR,TNR,FPR,FNR,PREC,BACC,F1)

NotImplementedError: You have not implemented this function
```

Let's say we have a population of 1000 people and we know that 50 are infected with the corona virus.

- **Task 5.2**: Assume that the population is tested with an assay that has a certain specificity and sensitivity. What is the probability that a person is *not* infected if they are diagnosed as ill by the test? Write a function that returns the desired value. Then check your calculation using specificity of 97% and sensitivity of 98%.

Note: Round your result to 3 decimal points, i.e. 0.987 if its 98.7%

5.2 Calculation (7 points):

In [ ]:
```
"""
This function should return the desired probability.
@param spec, float, specificity
@param sens, float, sensitivity
@param pop, int, population
@param inf, int, infected
"""
def calc_prob(spec,sens,pop,inf):
    population = pop
    infected = inf

    # replace the following line with your lines of code
    raise NotImplementedError("You have not implemented this function")

_result = calc_prob(0.97,0.98,1000,50)
print("The probability that a person who is tested positive is in fact not infecte
```

In [ ]:
```
# executability check
est_mean_cov(X,y)
```

```
calc_par_A(np.eye(2),np.ones(shape=2),np.eye(2),np.ones(shape=2),1,1)
calc_par_b(np.eye(2),np.ones(shape=2),np.eye(2),np.ones(shape=2),1,1)
calc_par_c(np.eye(2),np.ones(shape=2),np.eye(2),np.ones(shape=2),1,1)
calc_func_g(np.ones(shape=(2,2)),np.ones(2),1,np.ones(shape=(250000,2)))
create_train_X(1,1)
create_train_y(1,1,np.ones((200,20)))
bias_var(np.ones((200,20)),np.ones((200,20)))
evaluate_measures(1,1,1,1)
calc_prob(1,1,1,1)
print("Executable")
```