**Lab Sessions Day 3**

**Exercise 1**

**Linear regression** #Make two vector X and y

X=np.array([1,2,4,3,5]) y=np.array([1,3,3,2,5])

#With simple linear regression we want to model our data as follows: #y = B0 + B1 \* x

#We can start off by estimating the value for B1 as: #B1 = sum((Xi-mean(X)) \* (yi-mean(y))) / sum((Xi- mean(X))^2)

X\_mean=X-X.mean() print(X\_mean) sqr\_X\_mean=X\_mean\*X\_mean y\_mean=y-y.mean() print(y\_mean) Sqr\_X\_mean\_y\_mean=X\_mean\*y\_mean print(Sqr\_X\_mean\_y\_mean) Sum\_Sqr\_X\_mean\_y\_mean=Sqr\_X\_mean\_y\_mean.sum() print(Sum\_Sqr\_X\_mean\_y\_mean)

B1= Sum\_Sqr\_X\_mean\_y\_mean/sqr\_X\_mean.sum() print(B1)

#We can calculate B0 using B1 and some statistics from our dataset, as follows: #B0 = mean(y) – B1 \* mean(X)

B0 = y.mean()-(B1\*X.mean()) print(B0)

#Making Predictions (y\_hat is a predicted y) y\_hat=B0+B1\*X

y\_hat=B0+B1\*X print(X,y,y\_hat)

#Evaluation RMSE = sqrt( sum( (y\_hat\_i – yi)^2 )/n )

n=np.size(X) error=y\_hat - y print(error) error\_sqr=error\*error print(error) RMSE = np.sqrt( error\_sqr.sum()/n) print(RMSE)

**Exercise 2**

**Logistic regression**

from sklearn.linear\_model import LogisticRegression from sklearn.datasets import load\_breast\_cancer from sklearn.model\_selection import train\_test\_split

cancer=load\_breast\_cancer() X\_train,X\_test,y\_train,y\_test=train\_test\_split(cancer.data,cancer.target,stratify=cancer.target,ra ndom\_state=42)

######default C=1##### lgr=LogisticRegression().fit(X\_train,y\_train) print("training set score: %f" % lgr.score(X\_train, y\_train)) print('\n'"test set score: %f" % lgr.score(X\_test, y\_test))

######increase C to 100##### lgr100=LogisticRegression(C=100).fit(X\_train,y\_train) print('\n'"training set score of lgr100: %f" % lgr100.score(X\_train, y\_train)) print('\n'"test set score of lgr100: %f" % lgr100.score(X\_test, y\_test))

Change C value and compare the performance metric ######decrease C to 0.01##### lgr001=LogisticRegression(C=0.01).fit(X\_train,y\_train) print('\n'"training set score of lgr001: %f" % lgr001.score(X\_train, y\_train)) print('\n'"test set score of lgr001: %f" % lgr001.score(X\_test, y\_test))

import matplotlib.pyplot as plt

plt.plot(lgr.coef\_.T,'o',label='C=1') plt.plot(lgr100.coef\_.T,'+',label='C=100') plt.plot(lgr001.coef\_.T,'-',label='C=0.01') plt.xticks(range(cancer.data.shape[1]),cancer.feature\_names,rotation=90) plt.ylim(-5,5) plt.legend() plt.show()

###If we desire a more interpretable model, using L1 regularization might help ###As LogisticRegression applies an L2 regularization by default, the result ###looks similar to Ridge in Figure ridge\_coefficients. Stronger regularization ###pushes coefficients more and more towards zero, though coefficients never ###become exactly zero.

import numpy as np import math n=np.arange(-2,3) print(n) r=pow(float(10),n) print(r) for C in r:

lr\_l1=LogisticRegression(C=C,penalty="l1").fit(X\_train,y\_train) print('\n'"Training Accuracy of L1 LogRess with C=%f:%f"%(C,lr\_l1.score(X\_train,y\_train))) print('\n'"Test Accuracy of L1 LogRegss with C=%f: %f"%(C,lr\_l1.score(X\_test,y\_test))) plt.plot(lr\_l1.coef\_.T,'o',label="C=%f"%C) plt.xticks(range(cancer.data.shape[1]),cancer.feature\_names,rotation=90) plt.ylim(-5,5) plt.legend(loc='best') plt.show()