Musterloesung Serie 1, Aufg. 6, logistische Regression

In [1]:
```
# libraries pandas, matplotlib und numpy
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Daten einlesen
data = pd.read_csv("voice.csv")

# Preview the first 5 lines of the loaded data
data.head()
```

Out[1]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	sfm	...	centroid	meanfun
0	0.059781	0.064241	0.032027	0.015071	0.090193	0.075122	12.863462	274.402906	0.893369	0.491918	...	0.059781	0.084279
1	0.066009	0.067310	0.040229	0.019414	0.092666	0.073252	22.423285	634.613855	0.892193	0.513724	...	0.066009	0.107937
2	0.077316	0.083829	0.036718	0.008701	0.131908	0.123207	30.757155	1024.927705	0.846389	0.478905	...	0.077316	0.098706
3	0.151228	0.072111	0.158011	0.096582	0.207955	0.111374	1.232831	4.177296	0.963322	0.727232	...	0.151228	0.088965
4	0.135120	0.079146	0.124656	0.078720	0.206045	0.127325	1.101174	4.333713	0.971955	0.783568	...	0.135120	0.106398

5 rows × 21 columns

Wir wollen herausfinden, welche der Eigenschaften der Stimme (meanfreq, median, meanfun) am besten voraussagen kann, ob die Person weiblich oder maennlich ist. Dazu muessen wir jeweils eine logistische Funktion fitten.

In [2]:
```
#load libraries for regression
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import seaborn as sn
```

In diesem Fall sind alle Daten brauchbar, d.h. man muss nicht zuerst bereinigen (wenn man das nicht wuesste, muesste man es aber testen).

In [3]:
```
#dummy coding fuer male/female
x  = data['meanfreq'].values.reshape(-1,1)
dummies = pd.get_dummies(data, columns=['label'])
y = dummies['label_male'].values.reshape(-1,1)
dummies.head()
```

Out[3]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	sfm	...	meanfun	minfun
0	0.059781	0.064241	0.032027	0.015071	0.090193	0.075122	12.863462	274.402906	0.893369	0.491918	...	0.084279	0.015702
1	0.066009	0.067310	0.040229	0.019414	0.092666	0.073252	22.423285	634.613855	0.892193	0.513724	...	0.107937	0.015826
2	0.077316	0.083829	0.036718	0.008701	0.131908	0.123207	30.757155	1024.927705	0.846389	0.478905	...	0.098706	0.015656
3	0.151228	0.072111	0.158011	0.096582	0.207955	0.111374	1.232831	4.177296	0.963322	0.727232	...	0.088965	0.017798
4	0.135120	0.079146	0.124656	0.078720	0.206045	0.127325	1.101174	4.333713	0.971955	0.783568	...	0.106398	0.016931

5 rows × 22 columns

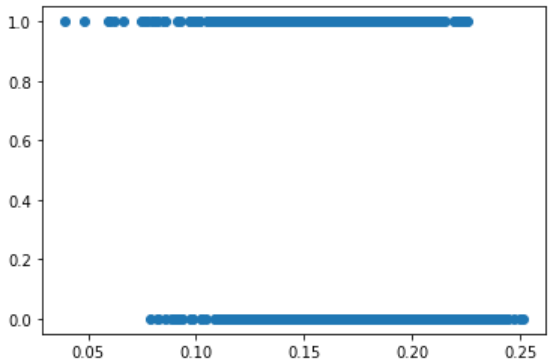
In [4]:
```
#not sure why pandas creates 2D arrays, but need 1D otherwise logistic regression would crash
x=x[:,0]
y=y[:,0]
```

In [5]:
```
#Anzahl Eintrage in unserem Datensatz
x.shape
```

Out[5]:
```
(3168,)
```

In [6]:
```
#scatter plot
plt.plot(x,y,'o')
plt.show()

#Codierung: 1=Mann, 0=Frau
```



Man sieht, dass Maenner generell eine tiefere meanfreq (mittlere Frequenz der Stimme haben). Auch sieht man einen grossen Ueberlapp, sodass wir uns schon denken koennen, dass ein logistisches Modell Muehe haben koennte.

In [7]:
```
#train the model

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x.reshape(-1,1), y, test_size = 0.2, random_st
ate = 0)
# wir trainieren auf 80% der Daten und testen mit 20% der Daten.
```

In [8]:
```
# die log. Regression in Python macht als Default Regularisierung. Dies wollen wir aber hier nich
t.
logreg = LogisticRegression(penalty='none') #funktioniert nicht mit python 2.x
#logreg = LogisticRegression(C=10000) #koennte evtl mit python 2.x gehen

# Fit des Modells (training data)
logreg.fit(X_train,y_train)

# Vorhersage von y fuer die Testdaten
y_pred=logreg.predict(X_test)
y_pred2 = logreg.predict_proba(X_test)[:,1]  #zwischen 0 und 1

print(logreg.coef_, logreg.intercept_)
```

```
[[-25.8863948]] [4.67301218]
```

In [9]:
```
#Anzahl Eintrage fuer Training
y_train.shape
```

Out[9]:
```
(2534,)
```

In [10]:
```
#Anzahl Eintrage fuers Testen
y_test.shape
```

Out[10]:
```
(634,)
```

In [11]:
```
#confusion matrix
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

Out[11]:
```
array([[223,  78],
       [144, 189]])
```

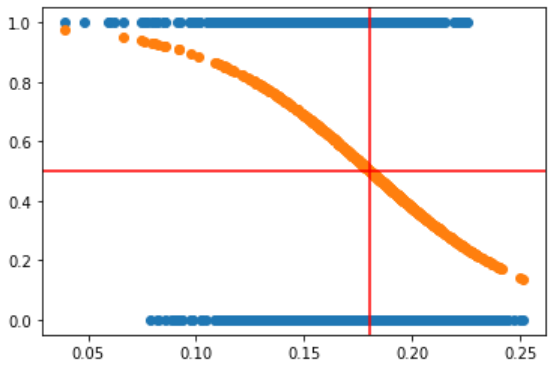
Die falsch-positiven und falsch-negativen Klassifikationen betragen 144+78=222, im Vergleich zu den richtig klassifizierten Punkten (223+189=412). Somit wurden 222/(222+412) = 35% falsch klassifiziert. Dies ist nicht besonders gut.

Wir schauen, ob die anderen Variablen besser sind, um male/female zu unterscheiden.

In [12]:
```
#Berechnung der Klassifikationsgrenze (bei 0.5)
#0.5 = 1./(1.+np.exp(-(x*logreg.coef_[0]+ logreg.intercept_[0])))
# (1.+np.exp(-(x*logreg.coef_[0]+ logreg.intercept_[0]))) = 1./0.5
# np.exp(-(x*logreg.coef_[0]+ logreg.intercept_[0]))) = 1./0.5 - 1   #->log
# -(x*logreg.coef_[0]+ logreg.intercept_[0]))) = np.log(1./0.5 - 1)
grenze = (np.log(1./0.5 - 1) - logreg.intercept_[0] ) / logreg.coef_[0]
```

In [13]:
```
#plotting
#scatter plot
plt.plot(x,y,'o')
plt.plot(X_test,y_pred2,'o')
plt.axhline(y = 0.5, color = 'r', linestyle = '-')
plt.axvline(x = grenze, color = 'r', linestyle = '-') # Berechneter Wert bei y=0.5
plt.show()

# hier sieht man die fehlende Trennung der Datenpunkte in x und dass einige falsch klassifiziert w
erden.
```

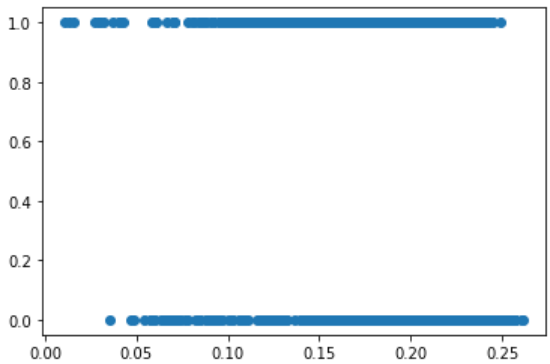


Wir testen nun die anderen Variablen

In [14]:
```
x  = data['median'].values.reshape(-1,1)
x=x[:,0]
```

In [15]:
```
#scatter plot
plt.plot(x,y,'o')
plt.show()

#Codierung: 1=Mann, 0=Frau
```



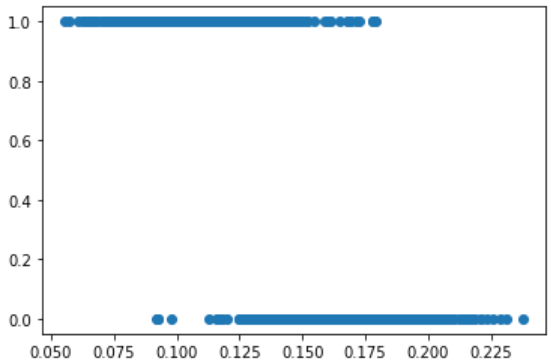
Dies sieht schlecht aus mit der Trennung der Daten in x-Richtung, deswegen verzichten wir auf den Fit.

Der naechste Test ist mit meanfun

In [16]:
```
x  = data['meanfun'].values.reshape(-1,1)
x=x[:,0]
```

In [17]:
```
#scatter plot
plt.plot(x,y,'o')
plt.show()

#Codierung: 1=Mann, 0=Frau
```



Dies sieht besser aus, da der Ueberlapp kleiner ist.

In [18]:
```
#train the model

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x.reshape(-1,1), y, test_size = 0.2, random_st
ate = 0)
```

In [19]:
```
# die log. Regression in Python macht als Default Regularisierung. Dies wollen wir aber hier nich
t.
logreg = LogisticRegression(penalty='none') #funktioniert nicht mit python 2.x

# fit the model with data
logreg.fit(X_train,y_train)

#
y_pred  = logreg.predict(X_test)
y_pred2 = logreg.predict_proba(X_test)[:,1]

print(logreg.coef_, logreg.intercept_)
```

```
[[-189.17086626]] [26.78272021]
```

In [20]:
```
#Berechnung der Klassifikationsgrenze (bei 0.5)
#0.5 = 1./(1.+np.exp(-(x*logreg.coef_[0]+ logreg.intercept_[0])))
# (1.+np.exp(-(x*logreg.coef_[0]+ logreg.intercept_[0]))) = 1./0.5
# np.exp(-(x*logreg.coef_[0]+ logreg.intercept_[0]))) = 1./0.5 - 1   #->log
# -(x*logreg.coef_[0]+ logreg.intercept_[0]))) = np.log(1./0.5 - 1)
grenze = (np.log(1./0.5 - 1) - logreg.intercept_[0] ) / logreg.coef_[0]
grenze
```

Out[20]:
```
array([0.14157952])
```

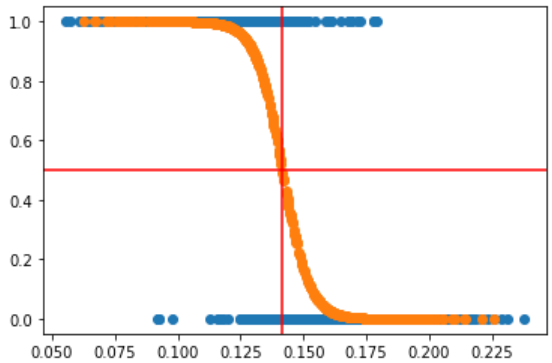
In [21]:
```
#confusion matrix
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

Out[21]:
```
array([[283,  18],
       [ 11, 322]])
```

Dies sieht ziemlich gut aus. Die falsch-positiven und falsch-negativen Klassifikationen betragen 11+18=29, im Vergleich zu den richtig klassifizierten Punkten (283+322). Somit wurden 29/(605+29) = 4.6% falsch klassifiziert. Dies bezieht sich uebrigens nur auf das Testset.

In [22]:
```
#plotting
#scatter plot
plt.plot(x,y,'o')
plt.plot(X_test,y_pred2,'o')
plt.axhline(y = 0.5, color = 'r', linestyle = '-')
plt.axvline(x = grenze, color = 'r', linestyle = '-') # Berechneter Wert bei y=0.5
plt.show()

#Auch hier sieht man die nicht-perfekte Trennung der Datenpunkte in x, aber das Modell ist ziemlic
h gut.
```



Zusammenfassend koennen wir sagen, dass 'meanfun' gut fuer die Klassifikation funktioniert, aber man ca. 5% der Leute falsch klassifizieren wird. Fuer eine reale Anwendung wuerde man wahrscheinlich 2 Variablen verwenden (also meanfun und noch eine andere) um den Prozentsatz der falschen Klassifikationen noch zu minimieren.

In [ ]: