Gabriel BUGINGA

# Machine Learning
# CPS 863

Lista 5

November 7, 2019

| | |
|---|---|
| Date Performed: | November 7, 2019 |
| Institution: | PESC/COPPE/UFRJ |
| Instructors: | Edmundo de Souza e Silva |
| | Rosa M. M. Leão |
| | Daniel Sadoc Menasché |

## Problem 1

**1**: For $M_1$ we used $\phi_1 = [1, x]$, for $M_2$ they were $\phi_2 = \left[1, x, x^2, x^3, x^4\right]$. At the "Code in Python"s code section we show the code used, figure 1 plots the results; .The coefficients obtained were:

$$For \quad \hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_1^2 + \ldots + \hat{\beta}_d x_i^d + \epsilon$$
$$coef(M_1) = (158836.1518, 2.0999)$$
$$coef(M_2) = (163873.6246, 4.98775578 \cdot 10^{-9}, 1.58489864 \cdot 10^{-4}, -1.79847628 \cdot 10^{9}, 5.05391918 \cdot 10^{15})$$

**2**: For the data points $D_x = 334, 438, 520, 605, 672, 767$ and $D_y = 39300, 60000, 68500, 86000, 113000, 133000$ we obtained:

$$MSE_{M_1} = 6.8741 \cdot 10^9$$
$$MSE_{M_2} = 7.5085 \cdot 10^9$$
$$\frac{MSE_{M_1}}{MSE_{M_1}} = 0.9155$$

**3**:Directly applying the above models for 848 and 912 sq feet: $M_1 \Rightarrow \widehat{y_1} = 160,616.9281$ and $\widehat{y_2} = 160,751.3263$ and $M_2 \Rightarrow \widehat{y_1} = 163,986.5012$ and $\widehat{y_2} = 164,004.0869$.

**4**:For the true labels $y_1 = 155900$, $y_2 = 156000$:

$$MSE_{M_1} = 2.2412 \cdot 10^7$$
$$MSE_{M_2} = 6.4728 \cdot 10^7$$
$$\frac{MSE_{M_1}}{MSE_{M_2}} = 0.3463$$

It's clear from Figure 1 that both models greatly suffer from outliers, so the mean squared errors are still big. Moreover, concerning the ratio difference from before: being points with higher x value, the non-linearities can start to matter more - contributing to a better fit - so the ratio decreased for these points, making the model 2 better at them than at the data points at **2**.

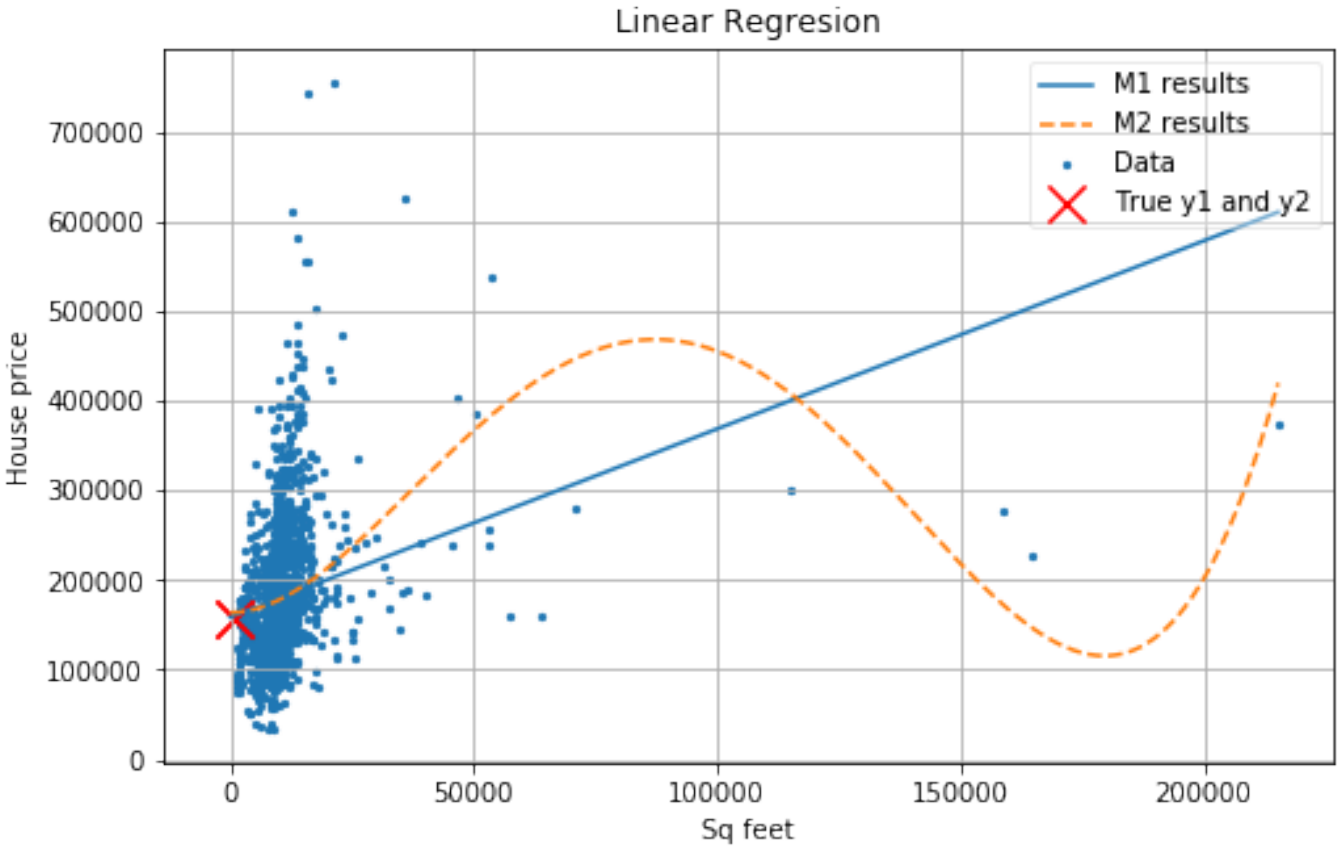


Figure 1: Problem 1's linear regression results

## Problem 2

**1**:First note that logistic regression is a binary classification problem, where in our case $y = 1$ when $price \geq T$ and $y = 0$ when $price < T$. For it, the negative log-likelihood can be readily applied in relation to the parameters $w_1$ and $w_2$ of the model. Making $\tilde{y}_i \in \{-1, +1\}$ instead of $y_i \in \{0, 1\}$ causes the algebraic form to be simpler, check [1] for more details - $\mu(\mathbf{x}) = \mathbb{E}[y|\mathbf{x}] = p(y = 1|\mathbf{x})$.

$$p(y|\mathbf{x}, \mathbf{w}) = \text{Ber}\left(y|\,\text{sigm}\left(\mathbf{w}^T\mathbf{x}\right)\right)$$

$$\begin{aligned}
\text{NLL}(\mathbf{w}) &= -\sum_{i=1}^{N} \log\left[\mu_i^{\mathbb{I}(y_i=1)} \times (1-\mu_i)^{\mathbb{I}(y_i=0)}\right] \\
&= -\sum_{i=1}^{N} \left[y_i \log \mu_i + (1-y_i)\log\left(1-\mu_i\right)\right] \\
&= \sum_{i=1}^{N} \log\left(1+\exp\left(-\tilde{y}_i \mathbf{w}^T\mathbf{x}_i\right)\right)
\end{aligned}$$

**2**:Unfortunately, we can't derive and equal zero - in this case we have to decease the whole of $\text{NLL}(\mathbf{w})$, one form to do this is just apply a form of gradient descent, treating it as a optimization problem.

Even better, this function is convex and possess a unique global minimum, so we just iterate and calculate the convergence.

**3**:For $\mathbf{w}^T\mathbf{x} = [w_0, w_1 x]$, we plot the Sigmoid and the Calibration curve at Figure 2:

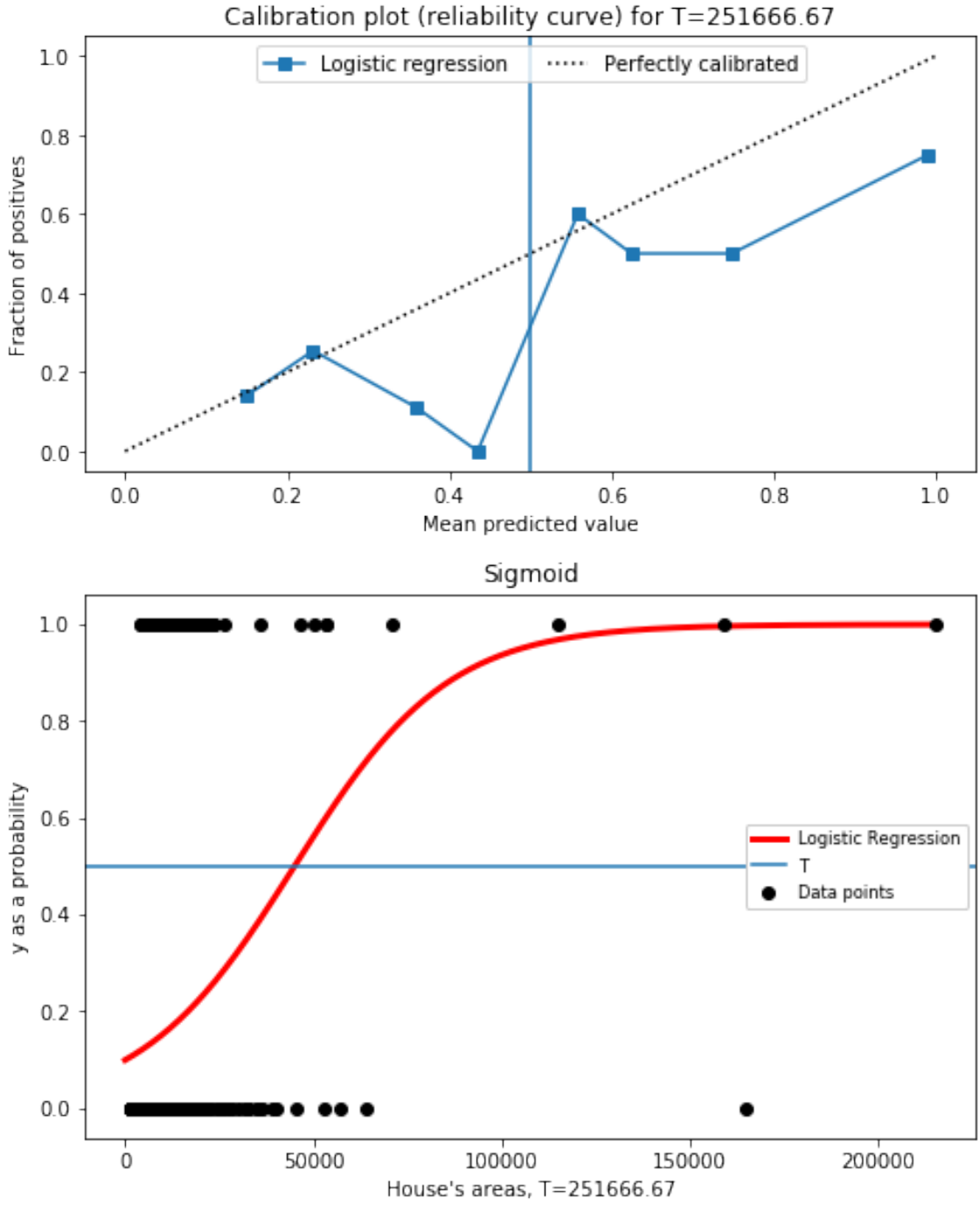


Figure 2: Problem 2's logistic regression results.

**4**:For $area = 100000$ square feet, we have $p(y = 1|\mathbf{100000}, \mathbf{w}) = 93.68\%$.

# Problem 3

**1**: Considering the probability distribution for each feature given a class is a bernolli variable, then we can readily find the parameters by the MLE for naive bayes $\hat{\pi}_c = \frac{N_c}{N}$ and $\hat{\theta}_{jc} = \frac{N_{jc}}{N_c}$.

| $\hat{\theta}_{word,class}$ | | |
|---|---|---|
| word | positive | negative |
| good | 0.030445 | 0.023321 |
| funny | 0.045277 | 0.026119 |
| love | 0.041374 | 0.009328 |
| bad | 0.001561 | 0.057836 |
| dull | 0.001561 | 0.026119 |
| worst | 0.000781 | 0.022388 |

**2**:We don't have all the words and use the MLE parameter for the approximate posterior in order to calculate the posterior predictive density:

$$p(y = c|\mathbf{x}, \mathcal{D}) \propto \hat{\pi}_c \prod_{j=1}^{D} \left(\hat{\theta}_{jc}\right)^{\mathbb{I}(x_j=1)} \left(1 - \hat{\theta}_{jc}\right)^{\mathbb{I}(x_j=0)}$$

The phrases are the following data points: $\mathbf{x} = [0,1,0,0,0,0], [1,0,0,0,1,0], [1,0,0,1,0,0]$, then the approximations are:

$$p(y = positive|[0,1,0,0,0,0], \mathcal{D}) \propto 2.2820 \cdot 10^{-2}$$
$$p(y = negative|[0,1,0,0,0,0], \mathcal{D}) \propto 1.0328 \cdot 10^{-2}$$
$$p(y = positive|[1,0,0,0,1,0], \mathcal{D}) \propto 2.3628 \cdot 10^{-5}$$
$$p(y = negative|[1,0,0,0,1,0], \mathcal{D}) \propto 2.4660 \cdot 10^{-4}$$
$$p(y = positive|[1,0,0,1,0,0], \mathcal{D}) \propto 2.3628 \cdot 10^{-5}$$
$$p(y = negative|[1,0,0,1,0,0], \mathcal{D}) \propto 5.6444 \cdot 10^{-4}$$

The first phrase "This is a very funny , heartwarming film ." got its classification right by aiming for "funny". The second "It 's never dull and always looks good ." got wrongly classified as negative because the model weighted the word "dull" more strongly, the reason for this is that the ratio for the quantity inside the negative review are much greater for "dull" $\frac{28}{2}$ than for "good" $\frac{25}{39}$. The third phrase "A bad movie that happened to good actors ." classified rightly as negative, the rationale is the same, the word "bad" is much more predictive for negative reviews by having 62 and just 2 for positive reviews, while "good" is more mixed: for positive review 39 and negative 25.

**3**:Using new data from table 7 and 8:

| $\hat{\theta}_{word,class}$ | | |
|---|---|---|
| word | positive | negative |
| good | 0.030445 | 0.023321 |
| funny | 0.045277 | 0.026119 |
| love | 0.041374 | 0.009328 |
| bad | 0.001561 | 0.057836 |
| dull | 0.001561 | 0.026119 |
| worst | 0.000781 | 0.022388 |

$$
\begin{aligned}
p(y = positive|[0,1,0,0,0,0],\mathcal{D}) &\propto 4.1570 \cdot 10^{-2} \\
p(y = negative|[0,1,0,0,0,0],\mathcal{D}) &\propto 1.8813 \cdot 10^{-4} \\
p(y = positive|[1,0,0,0,1,0],\mathcal{D}) &\propto 4.3041 \cdot 10^{-5} \\
p(y = negative|[1,0,0,0,1,0],\mathcal{D}) &\propto 4.4922 \cdot 10^{-6} \\
p(y = positive|[1,0,0,1,0,0],\mathcal{D}) &\propto 4.3041 \cdot 10^{-5} \\
p(y = negative|[1,0,0,1,0,0],\mathcal{D}) &\propto 1.0281 \cdot 10^{-5}
\end{aligned}
$$

The $\hat{\theta}_{word,class}$ are all the same, but the prior now is greatly biased toward the positive reviews: $\hat{\pi}_{positive} = 0.9917$ and $\hat{\pi}_{negative} = 0.0083$, this makes all the predictions positive.

# Problem 4

**1**:The mixture of two gaussians has the following form:

$$
p\left(\mathbf{x}_i|\boldsymbol{\theta}\right) = \pi_1 \mathcal{N}\left(\mathbf{x}_i|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1\right) + \pi_2 \mathcal{N}\left(\mathbf{x}_i|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2\right)
$$

**2**:We can't because of the classical problem of maximizing a log of a sum. A closed form is still possible if we obtain the value of the cluster from which the data point was obtained.

**3**: We have for the algorithm the following steps:

$$
\begin{aligned}
&\mathbf{E} - step \\
&\qquad r_{ik} = \frac{\pi_k p\left(\mathbf{x}_i|\boldsymbol{\theta}_k^{(t-1)}\right)}{\sum_{k'} \pi_{k'} p\left(\mathbf{x}_i|\boldsymbol{\theta}_{k'}^{(t-1)}\right)} \\
&\mathbf{M} - step \\
&\qquad \pi_k = \frac{1}{N}\sum_i r_{ik} = \frac{r_k}{N} \\
&\qquad \boldsymbol{\mu}_k = \frac{\sum_i r_{ik}\mathbf{x}_i}{r_k} \\
&\qquad \boldsymbol{\Sigma}_k = \frac{\sum_i r_{ik}\left(\mathbf{x}_i - \boldsymbol{\mu}_k\right)\left(\mathbf{x}_i - \boldsymbol{\mu}_k\right)^T}{r_k} = \frac{\sum_i r_{ik}\mathbf{x}_i\mathbf{x}_i^T}{r_k} - \boldsymbol{\mu}_k\boldsymbol{\mu}_k^T
\end{aligned}
$$

**4**: Interpreting that the mentioned "3 steps" are 3 entire iterations of the EM-algorithm:

$$
\begin{aligned}
&\textbf{EM} - \textit{step}\ 0 \\
&\qquad \pi_0 = [0.5, 0.5] \\
&\qquad \boldsymbol{\mu}_0 = [2.0684, 22.0684] \\
&\qquad \boldsymbol{\Sigma}_0 = [1, 1] \\
&\textbf{EM} - \textit{step}\ 1 \\
&\qquad \pi_1 = [0.4059, 0.5940] \\
&\qquad \boldsymbol{\mu}_1 = [6.4628, 15.8992] \\
&\qquad \boldsymbol{\Sigma}_1 = [7.2184, 5.5884] \\
&\textbf{EM} - \textit{step}\ 2 \\
&\qquad \pi_2 = [0.4059, 0.5940] \\
&\qquad \boldsymbol{\mu}_2 = [6.4628, 15.8992] \\
&\qquad \boldsymbol{\Sigma}_2 = [7.2184, 5.5884] \\
&\textbf{EM} - \textit{step}\ 3 \\
&\qquad \pi_3 = [0.4059, 0.5940] \\
&\qquad \boldsymbol{\mu}_3 = [6.4628, 15.8992] \\
&\qquad \boldsymbol{\Sigma}_3 = [7.2184, 5.5884]
\end{aligned}
$$

It is of note that these values were achieved by using different starting points for the parameter at step 0, moreover the final value systematically converged after only one entire algorithm iteration. Finally Figure 3 shows the results:

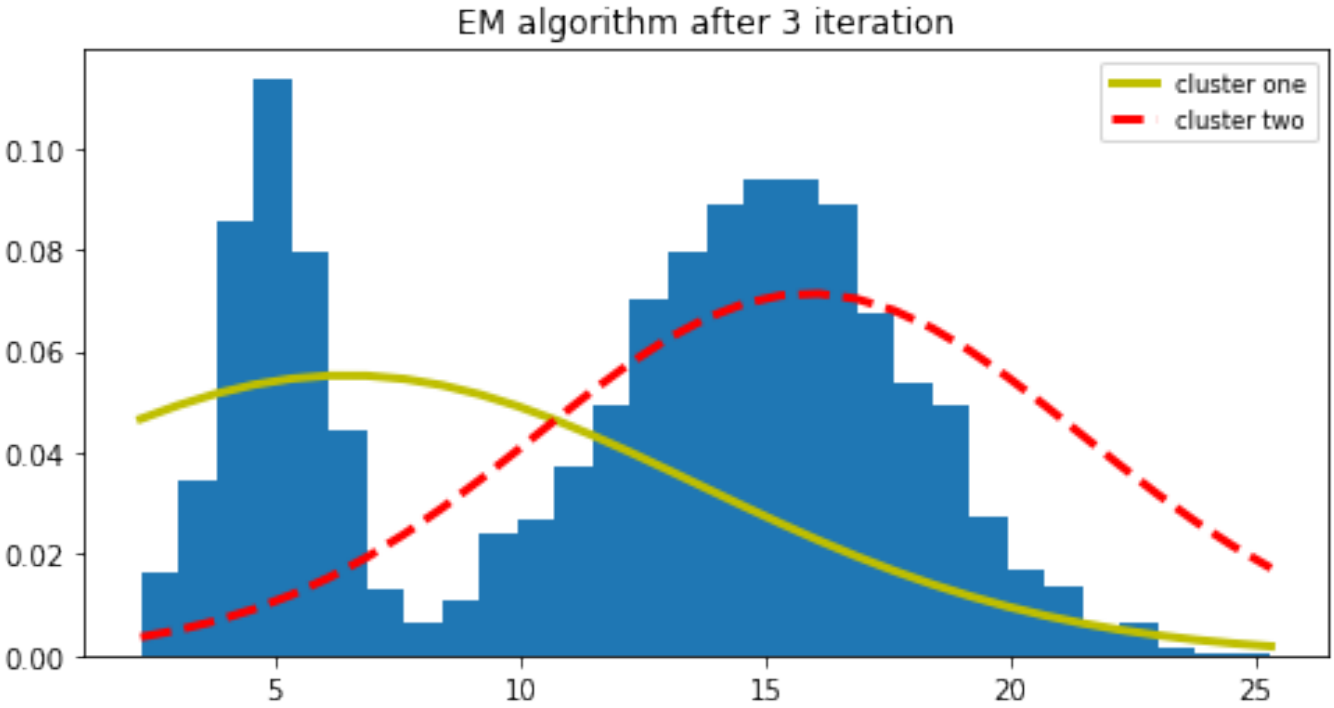


Figure 3: Problem 4's EM algorithm results

## Code in Python

### Problem 1

```
from matplotlib import pyplot as plt
%matplotlib inline
from IPython.core.pylabtools import figsize
from sklearn.calibration import calibration_curve
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.datasets import load_boston
from sklearn import datasets
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
import numpy as np
import pandas as pd
import os
current_path = os.getcwd()
exp_file_2 = '\\datasets\\train.csv'
df = pd.DataFrame()
df = pd.read_csv(current_path + exp_file_2 ,comment = '#')

areas = df.loc[:,'LotArea'].values
areas = areas.reshape(len(areas),1)
prices = df.loc[:,'SalePrice'].values

regression_m1 = LinearRegression()
M1 = regression_m1.fit(areas,prices)

polynomial = PolynomialFeatures(degree=4,include_bias = False)
features_polynomial = polynomial.fit_transform(areas)
regression_m2 = LinearRegression()
M2 = regression_m2.fit(features_polynomial, prices)

x = np.linspace(0,np.amax(areas),1000)
figsize(8, 5)
plt.plot(x,M1.predict(x.reshape(-1,1)),label="M1 results")
plt.plot(x,M2.predict(polynomial.fit_transform(x.reshape(-1,1))),
         label="M2 results",linestyle='--')
plt.scatter(areas,prices,label="Data",marker='o',s=5,alpha=1)

plt.scatter([848,912],[155900,156000],label="True y1 and y2",
            marker='x',s=200,color='r',alpha=1)

plt.xlabel("Sq feet")
plt.ylabel("House price")
leg = plt.legend(loc = "upper right")
leg.get_frame().set_alpha(0.4)

plt.title("Linear Regresion")
plt.grid(True)
print (M1.intercept_,M1.coef_.ravel())
print (M2.intercept_,M2.coef_)

a = np.array([334,438,520,605,672,767]).reshape(-1,1)
```

```
y_true = np.array([39300,60000,68500,86000,113000,133000])

y = M1.predict(a)
print (y)
MSE_M1 = (((y - y_true)**2).sum())/len(y)

y = M2.predict(polynomial.fit_transform(a))
print (y)
MSE_M2 = (((y - y_true)**2).sum())/len(y)

ratio = MSE_M1/MSE_M2
print ("MSE_M1 = %.4e, MSE_M2 = %.4e, ratio %.4f"%(MSE_M1,MSE_M2, ratio))

y1 = M1.predict([[848]])
y2 = M1.predict([[912]])
print ("M1 => y1 = %.4f, y2 = %.4f"%(y1,y2))

y1 = M2.predict(polynomial.fit_transform([[848]]))
y2 = M2.predict(polynomial.fit_transform([[912]]))
print ("M2 => y1 = %.4f, y2 = %.4f"%(y1,y2))

a = np.array([848,912]).reshape(-1,1)
y_true = np.array([155900,156000])

y = M1.predict(a)
print (y)
MSE_M1 = (((y - y_true)**2).sum())/len(y)

y = M2.predict(polynomial.fit_transform(a))
print (y)
MSE_M2 = (((y - y_true)**2).sum())/len(y)

ratio = MSE_M1/MSE_M2
print ("MSE_M1 = %.4e, MSE_M2 = %.4e, ratio %.4f"%(MSE_M1,MSE_M2, ratio))
```

**Problem 2**

```
T = np.amax(prices)/3
scaler = StandardScaler()
target = np.array(prices)
target[prices >= T] = 1
target[prices < T] = 0
model = logistic_regression.fit(areas, target)

plt.figure(figsize=(8, 4))
prob_pos = model.predict_proba(areas)[:,1]
fraction_of_positives, mean_predicted_value = \
    calibration_curve(target, prob_pos, n_bins=10)

plt.plot(mean_predicted_value, fraction_of_positives, "s-",
         label="Logistic regression")
plt.plot([0, 1], [0, 1], "k:", label="Perfectly calibrated")
plt.axvline(x=0.5)
```

```
plt.legend(loc="upper center", ncol=2)

plt.ylim([-0.05, 1.05])
plt.ylabel("Fraction of positives")
plt.xlabel("Mean predicted value")
plt.title("Calibration plot (reliability curve) for T=%.2f"%(T))

from scipy.special import expit
plt.figure(figsize=(8,5))
plt.scatter(areas, target, color='black', zorder=20,label="Data points")

X_test = np.linspace(0,np.amax(areas),1000)
loss = expit(X_test * model.coef_ + model.intercept_).ravel()
plt.plot(X_test, loss, color='red', linewidth=3,label="Logistic Regression")

plt.axhline(y=0.5,label="T")

leg = plt.legend(loc = "right",fontsize = 'small')
leg.get_frame().set_alpha(1)
plt.ylabel("y as a probability")
plt.xlabel("House's areas, T=%.2f"%(T))
plt.title("Sigmoid")

model.predict_proba([[100000]])
```

**Problem 3**

```
n = np.zeros([6,2])
theta = np.zeros([6,2])
n[:,0]=[39,58,53,2,2,1]
n[:,1]=[25,28,10,62,28,24]
n_c=np.array([1281,1072])

for c in np.arange(2):
    for i,_ in enumerate(n[:,c]):
        theta[i,c] = n[i,c]/n_c[c]
pi=n_c/np.sum(n_c)

points=np.array([[0,1,0,0,0,0],[1,0,0,0,1,0],[1,0,0,1,0,0]])
cond= np.array([[pi[0]]*3,[pi[1]]*3])

for cl,_ in enumerate(cond):
    for i,x in enumerate(points):
        for d in np.arange(6):
            factor = x[d]*theta[d,cl]+(1-x[d])*(1-theta[d,cl])
            cond[cl,i] = cond[cl,i]*factor
print (cond)

n = np.zeros([6,2])
theta = np.zeros([6,2])
n[:,0]=[3900,5800,5300,200,200,100]
n[:,1]=[25,28,10,62,28,24]
n_c=np.array([128100,1072])
```

```
for c in np.arange(2):
    for i,_ in enumerate(n[:,c]):
        theta[i,c] = n[i,c]/n_c[c]
pi=n_c/np.sum(n_c)

points=np.array([[0,1,0,0,0,0],[1,0,0,0,1,0],[1,0,0,1,0,0]])
cond= np.array([[pi[0]]*3,[pi[1]]*3])

for cl,_ in enumerate(cond):
    for i,x in enumerate(points):
        for d in np.arange(6):
            factor = x[d]*theta[d,cl]+(1-x[d])*(1-theta[d,cl])
            cond[cl,i] = cond[cl,i]*factor
print (cond)
```

### Problem 4

```
current_path = os.getcwd()
exp_file_2 = '\\datasets\\data_EM.txt'
df = pd.DataFrame()
df = pd.read_csv(current_path + exp_file_2,comment = '#',header=None)
def gaus(x,mu,sigma):
    return 1/(sigma * np.sqrt(2 * np.pi)) * \
    np.exp( - (x - mu)**2 / (2 * sigma**2))

t=3
theta = np.array([[[np.mean(data)-10,1],[np.mean(data)+10,1]]]*t)
pi = np.array([[0.5,0.5]]*t)
r = np.zeros([len(data),2])

for time in np.arange(t):
    #E-Step
    for i in np.arange(len(data)):
        for k in np.arange(2):

            r[i,k]=pi[time,k]*gaus(data[i],theta[time,k,0],theta[time,k,1])
            term = pi[time,0]*gaus(data[i],theta[time,0,0],theta[time,0,1])+\
            pi[time,1]*gaus(data[i],theta[time,1,0],theta[time,1,1])
            r[i,k] = r[i,k]/term
    #M-Step
    for k in np.arange(2):
        pi[time,k]=np.sum(r[:,k])/len(data)

        theta[time,k,0]=np.sum([r[i,k]*data[i] for i in np.arange(len(data))])/np.s

        theta[time,k,1]=np.sum([r[i,k]*data[i]*data[i]\
         for i in np.arange(len(data))])/\
            np.sum(r[:,k]) - theta[time,k,0]*theta[time,k,0]
```

## References


[1] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.