Assignment 2

Elena Pérez-Ródenas Martínez
REDID: 827222533

September 2021

I, Elena Pérez-Ródenas Martínez, guarantee that this homework is my independent work and I have never copied any part from other resources. Also, I acknowledge and agree with the plagiarism penalty specified in the course syllabus.

1. Exercise

1.1.

To apply MLE we have to find the θ such that maximizes $P(x;\theta)$ i.e.

$\theta_m = argmax_\theta P(x;\theta) = argmax_\theta(\frac{n_b+n_g+n_w}{n_b})\theta^{n_b}(1-\theta)^{n_g+n_w} =$
$= argmax_\theta log(\frac{n_b+n_g+n_w}{n_b}) + n_b log\theta + (n_g+n_w)log(1-\theta)$

Being n_b, n_g, n_w the number of black, green and white good watermelons.
If we derive the expression above and equalize it to zero to get the maximum point, we get:

$\frac{n_b}{\theta_m} = \frac{n_g+n_w}{1-\theta_m}$

And if you solve this:

$\theta_m = \frac{n_b}{n_b+n_g+n_w}$

So now we can guess $P(C = Black|Good) = \frac{4}{8} = \frac{1}{2}$
Similarly, we can guess the green and white ones:
$P(C = Green|Good) = \frac{3}{8}$ and $P(C = White|Good) = \frac{1}{8}$

1.2.

In order to decide if a person may or may not buy the computer we will use the Naive Bayes Classifier:
First of all we calculate the following probability:

$P(buy\_comp = yes|X) = \frac{P(X|buy\_comp=yes)*P(buy\_comp=yes)}{P(X)}$

$P(X|buy\_comp = yes) = P(age <= 30|buy\_comp = yes) * P(income = medium|buy\_comp = yes) * P(student = yes|buy\_comp = yes) * P(credit = fair|buy\_comp = yes) = \frac{2}{9} * \frac{4}{9} * \frac{2}{3} * \frac{2}{3} = \frac{32}{729} = 0{,}04389$

$P(buy\_comp = yes) = \frac{9}{14} = 0{,}6428$

$P(X) = P(age <= 30) * P(income = medium) * P(student = yes) * P(credit = fair) = \frac{5}{14} * \frac{6}{14} * \frac{7}{14} * \frac{8}{14} = \frac{15}{343} = 0{,}0437$

$P(buy\_comp = yes|X) = \frac{0{,}04389*0{,}6428}{0{,}0437} = 0{,}6453$

Now we can do the same with not buying the computer:

$P(buy\_comp = no|X) = \frac{P(X|buy\_comp=no)*P(buy\_comp=no)}{P(X)}$

$P(X|buy\_comp = no) = P(age <= 30|buy\_comp = no) * P(income = medium|buy\_comp = no) * P(student = yes|buy\_comp = no) * P(credit = fair|buy\_comp = no) = \frac{3}{5} * \frac{2}{5} * \frac{1}{5} * \frac{2}{5} = \frac{12}{625} = 0{,}0192$

$P(buy\_comp = no) = \frac{5}{14} = 0{,}3571$

$P(buy\_comp = no|X) = \frac{0{,}0192*0{,}3571}{0{,}0437} = 0{,}1568$

So the person **should buy** the computer because 0.6453>0.1568.

1.3.

A solution to prevent this from happening could be working with logarithms and instead of using

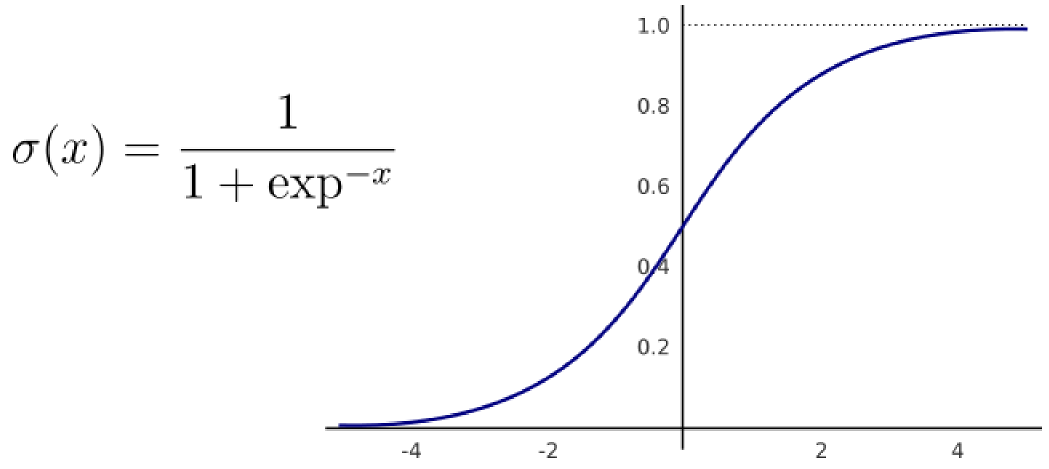
$h_{nb}(x) = argmax_{c \in Y} P(c) \prod_{i=1}^{d} P(x_i|c)$

Use $log(h_{nb}(x)) = log(argmax_{c \in Y} P(c) \prod_{i=1}^{d} P(x_i|c))$

Then, we can avoid underflow errors.

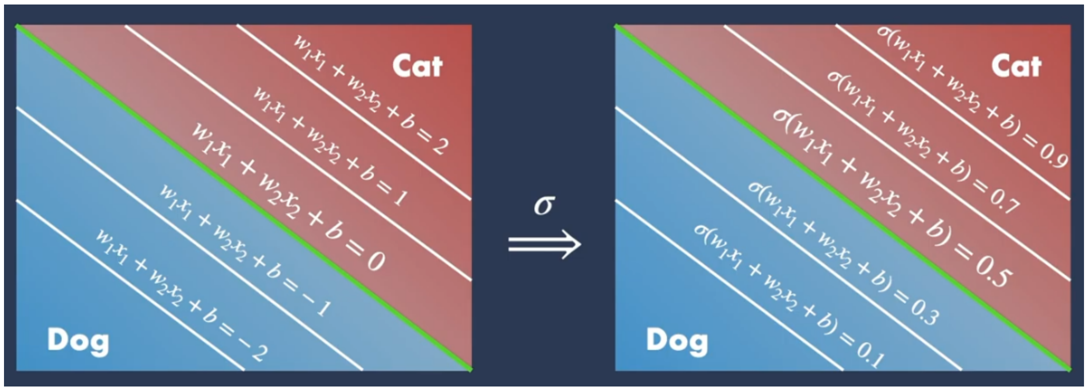
1.4.

Logistic regression predicts whether something is True or False instead of predicting something continuous like size. It's a technique to predict the outcome of binary classification problems. However, it's usually used for classification. For example, if the probability is greater than 50 % the we will classify it as True. Besides, logistic regression can work with either continuous or discrete data.
The Sigmoid function plays the role of an activation function. It takes the weighted sum of the input and outputs the probability of the outcome. Its formula is: $\sigma(x) = \frac{1}{1+e^{-x}}$ and its graphical representation is an s-shaped curve:



For each value of x, the sigmoid function will output a value between 0 and 1. The limits of this function are 0 when $x- > -\infty$, 0.5 when $x- > 0$ and 1 when $x- > +\infty$. That's why 1 and 0 are the upper and lower bound respectively for the sigmoid function.

The **sigmoid function** is used to make predictions. First, we set a decision boundary separating our two options which could be a straight line represented by $w_1x_1 + w_2x_2 + b = 0$. If we want to know the probability of any point in this line we need to apply the sigmoid function. The predition here will always be 0.5. If we draw lines in either sides of that line we can do the same and apply the sigmoid function to predict the probability value as we can see in the following picture: [1]

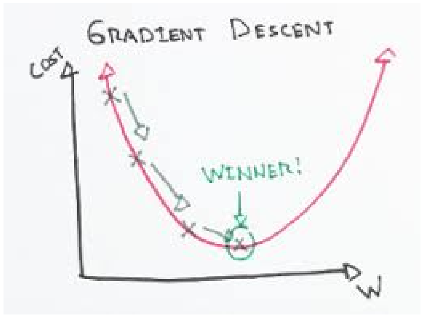

Cat
$w_1x_1 + w_2x_2 + b = 2$
$w_1x_1 + w_2x_2 + b = 1$
$w_1x_1 + w_2x_2 + b = 0$
$w_1x_1 + w_2x_2 + b = -1$
$w_1x_1 + w_2x_2 + b = -2$
Dog
σ
Cat
$\sigma(w_1x_1 + w_2x_2 + b) = 0.9$
$\sigma(w_1x_1 + w_2x_2 + b) = 0.7$
$\sigma(w_1x_1 + w_2x_2 + b) = 0.5$
$\sigma(w_1x_1 + w_2x_2 + b) = 0.3$
$\sigma(w_1x_1 + w_2x_2 + b) = 0.1$
Dog

Cost function is an error representation in Machine Learning. It shows how our model is predicting compared to the original given dataset. The less cost function the more accuracy we obtain. Its formula is

$Cost = \frac{-1}{m} * \sum_{i=1}^{m}[ylog(y_{pred}) + (1-y)log(1-y_{pred})]$ (this is the error for one observation)

$y_{pred} = \sigma(w^T x + b)$ (being σ the sigmoid function)

In order to minimize this cost function we need the **gradient descent** algorithm. The point is to change w until we get to the minimum point. To do this, we take $w = w - \alpha * \frac{\partial cos(t)}{\partial w}$ where α is a small number called learning rate and the derivative represents the slope of the graph.
As we can see in the picture below, if the derivative is negative, the w will increase and the other way around. That's how we get closer to the minimum. We also have to update b with $b = b - \alpha\frac{\partial cos(t)}{\partial b}$. [2]



2. Coding Assignment NBC

First of all we have two files called TR and TT containing the emails of our training and testing dataset respectively. Also, we have a file called spam-mail.tr with the predictions of each email in TR.
After importing every package we need, we will extract the body of the training emails and put them all in a folder called dst. We will use the function provided ExtractBodyFromDir(srcdir,dstdir).
After that, we will create a csv file called training_emails.csv with the text of the email and whether it is spam or not (0 or 1).

```
ExtractBodyFromDir(srcdir, dstdir)

df_tr = pd.read_csv('/home/elipeke/Escritorio/ML/A2/NBClassifier/spam-mail.tr.label')
spams = df_tr['Prediction']

header = ['text','spam']
data = []
```

```
for i in range(1,2499):
    data.append([read_file("".join(['/home/elipeke/Escritorio/ML/A2/NBClassifier/dst/TRAIN_',str(i),
    '.eml'])), spams[i-1]])

with open('training_emails.csv', 'w', encoding='UTF8', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(header)
    writer.writerows(data)
```

As we did in the previous assignment we will read and split the emails into random train and test subsets with a test size of 0.25.

```
df = pd.read_csv('/home/elipeke/Escritorio/ML/A2/NBClassifier/training_emails.csv')
X_train, X_test, y_train, y_test = train_test_split(df.text,df.spam,test_size=0.25)
```

To convert words into a matrix of features we will create a CountVectorizer and use the function fit_transform to fit the X_train values and transform them into a new array.

```
v = CountVectorizer()
X_train_count = v.fit_transform(X_train.values)
```

Now we will use Multinomial Naive Bayes model to classify the emails. We will have count of each word to predict if it is spam or not. (The class MultinomialNB has been implemented and will be explained later in this document.)

```
model = MultinomialNB()
model.fit(X_train_count, y_train)
```

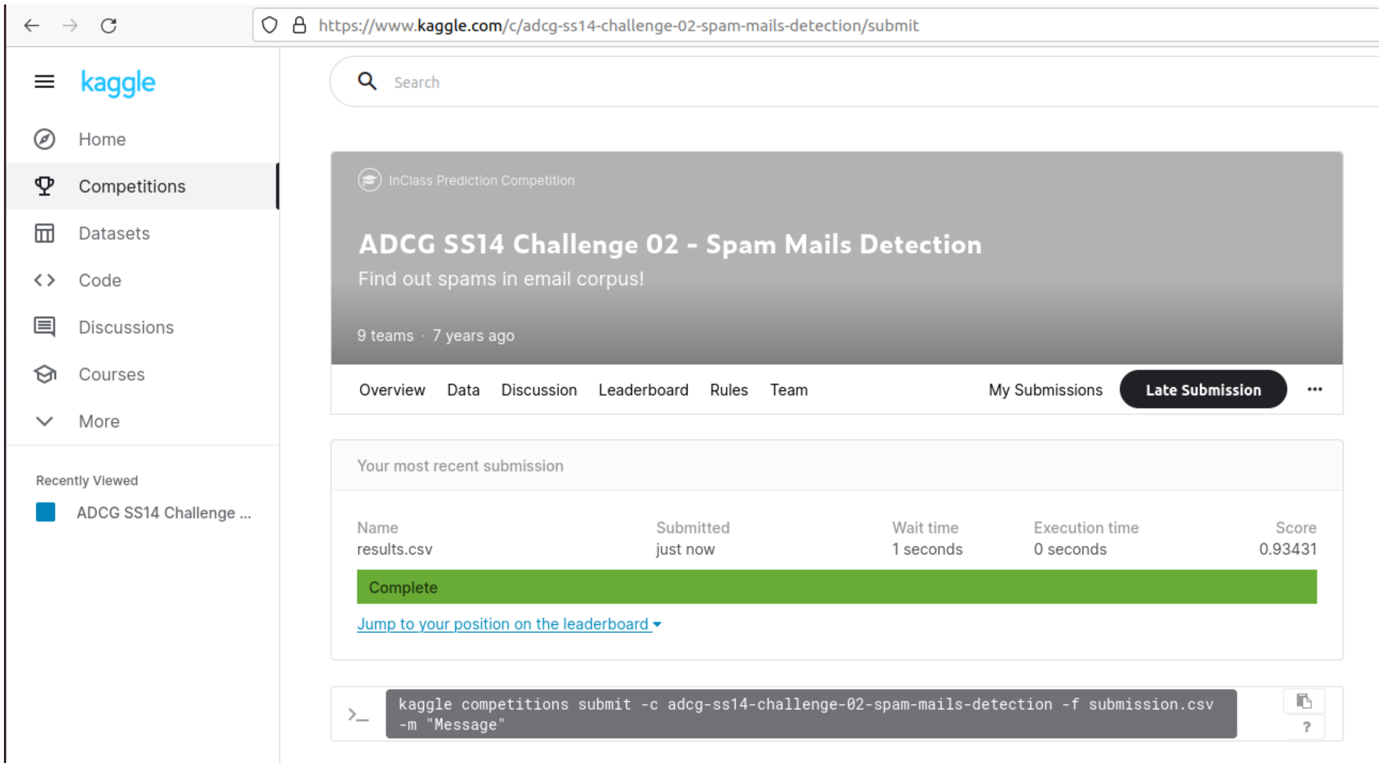
Next we will create an array with the emails of the testing set, transform it and use the model to make the predictions. (Remember that the model was already trained with the training set.)

```
emails = []
for i in range(1,1828):
    emails.append(read_file("".join(['/home/elipeke/Escritorio/ML/A2/NBClassifier/TT/TEST_',str(i),
    '.eml'])))

emails_count = v.transform(emails)
test = model.predict(emails_count)
```

Finally, we will create a final csv file with the final predictions called results.csv.

```
header2 = ['Id','Prediction']
data2 = []
count=1
for i in test:
    data2.append([count,i])
    count=count+1

with open('results.csv', 'w', encoding='UTF8', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(header2)
    writer.writerows(data2)
```

If we open the file we will notice that most of the predictions are the same as the ones on the training set. We can submit our file into the kaggle competition and see that we get a score of 0.93431.



ADCG SS14 Challenge 02 - Spam Mails Detection

Find out spams in email corpus!

9 teams · 7 years ago

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
results.csv	just now	1 seconds	0 seconds	0.93431

Complete

```
kaggle competitions submit -c adcg-ss14-challenge-02-spam-mails-detection -f submission.csv
-m "Message"
```

2.1. Implementation of MultinomialNB()

We will create a class called MultinomialNB() and implement the functions fit and predict.

For the fit function the parameters are the X_train_count with each email and the number of times each word appears in the email and the y_train with the predictions of the training set.

Our classes will be spam and ham, the priors will be the probability of spam or ham. To calculate them we just have to divide the number of spam/ham emails and the total number of emails.

The likelihoods will be the probabilities that a word is in the email knowing that it's spam/ham. Here it's important to add a number so that we never divide by zero.

```
class MultinomialNB():

    def fit(self, X_train, y_train):
        num_emails, num_words = X_train.shape
        self._classes = np.unique(y_train)
        num_classes = len(self._classes)

        self._priors = np.zeros(num_classes)
        self._likelihoods = np.zeros((num_classes, num_words))

        for index, c in enumerate(self._classes):
            class_X_train = X_train[c == y_train]
            self._priors[index] = class_X_train.shape[0] / num_emails
            self._likelihoods[index, :] = ((class_X_train.sum(axis=0)) + 1) / (
                np.sum(class_X_train.sum(axis=0) + 1))
```

Now for the predict function we will predict the class for each x_test returning an array with the results. In order to do that, we will create an array called posteriors with the sum of the class_likelihood and the prior and then get the maximum value.

```
    def predict(self, X_test):
        return [self.predict_each(x_test) for x_test in X_test]

    def predict_each(self, x_test):
        posteriors = []
        for index, c in enumerate(self._classes):
            class_likelihoods = x_test * np.transpose(np.log(self._likelihood[index, :]))
            class_posteriors = np.sum(class_likelihood) + np.log(self._priors[index])
            posteriors.append(class_posteriors)

        return self._classes[np.argmax(posteriors)]
```

3. Referencias

[1] https://www.youtube.com/watch?v=TPqr8t919YM.

[2] https://www.youtube.com/watch?v=t6MVuMavbBY.