Artificial Intelligence II Exercise 1

Michael Mitsios (cs2200011)

Nov 2021

PART 1

Calculation

$$MSE(w) = \frac{1}{m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})^2 (1)$$

with

$$h_w(x^{(i)}) = b + w \cdot x^{(i)}$$

b is the bias and w the weights. x is a vector with the features of an input.

We can assume that the bias of the equation is combined with the weights and in order to do this we add a new column in the start of the x vector that has all 1. The final form of $h_w(x^{(i)})$ is

$$h_w(x^{(i)}) = w_{new} \cdot x_{new}^{(i)^T} (2)$$

where

$$w_{new} = [w_0, w_1 \dots w_n]$$

$$x_{new}^{(i)} = \left[x_0^{(i)} = 1, x_1^{(i)} \dots x_n^{(i)}\right]$$

were n is the number of features that we have. So to find the gradient of $\nabla_w MSE(w)$we have to calculate the equation below.

$$\nabla_w MSE(w) = \begin{bmatrix} \frac{\partial \frac{1}{m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})^2}{\partial w_0} \\ \frac{\partial \frac{1}{m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})^2}{\partial w_1} \\ \vdots \\ \frac{\partial \frac{1}{m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})^2}{\partial w_n} \end{bmatrix} (3)$$

Let's for out ease say that $J(w) = \frac{1}{m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})^2$ which is equal to due to (2)

$$\frac{1}{m}\sum_{i=1}^{m}(w_{new} \cdot x_{new}^{(i)^T} - y^{(i)})^2 =$$

$$\frac{1}{m}\sum_{i=1}^{m}(w_0 x_0^{(i)} + w_1 x_1^{(i)} + \ldots + w_n x_n^{(i)} - y^{(i)})^2$$

If we compress the calculation inside the parenthesis into a variable $e_j(w) = w_0 x_0^{(i)} + w_1 x_1^{(i)} + \ldots + w_n x_n^{(i)} - y^{(i)}$

we can calculate easier the derivative: $\frac{\partial \frac{1}{m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})^2}{\partial w_0}$

So we have that

$$\frac{\partial \frac{1}{m}\sum_{i=1}^{m}(e_j(w))^2}{\partial w_0} =$$

We make use of the sum rule: $\frac{\partial \sum u}{\partial w} = \sum \frac{\partial u}{\partial w}$

$$\frac{1}{m}\sum_{i=1}^{m}\frac{\partial (e_j(w))^2}{\partial w_0} =$$

by using the **chain rule** we get:

$$\frac{1}{m}\sum_{i=1}^{m} 2e_j(w)\frac{\partial e_j(w)}{\partial w_0} (4)$$

The derivative of $\frac{\partial e_j(w)}{\partial w_0}$ is equal to:

$$\frac{\partial e_j(w)}{\partial w_0} = \frac{\partial w_0 x_0^{(i)}}{\partial w_0} + \frac{\partial w_1 x_1^{(i)}}{\partial w_0} + \ldots + \frac{\partial w_n x_n^{(i)}}{\partial w_0} - \frac{\partial y^{(i)}}{\partial w_0} =$$

$$x_0^{(i)}$$

The derivative of $\frac{\partial e_j(w)}{\partial w_1}$ is equal to:

$$\frac{\partial e_j(w)}{\partial w_1} = \frac{\partial w_0 x_0^{(i)}}{\partial w_1} + \frac{\partial w_1 x_1^{(i)}}{\partial w_1} + \ldots + \frac{\partial w_n x_n^{(i)}}{\partial w_1} - \frac{\partial y^{(i)}}{\partial w_1} =$$

$$x_1^{(i)}$$

From the **results** that we get above in combination with **(4)** we can see that (3) becomes:

$$\nabla_w MSE(w) = \begin{bmatrix} \frac{\partial \frac{1}{m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})^2}{\partial w_0} \\ \frac{\partial \frac{1}{m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})^2}{\partial w_1} \\ \vdots \\ \frac{\partial \frac{1}{m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})^2}{\partial w_n} \end{bmatrix} =$$

$$\begin{bmatrix} \frac{2}{m}\sum_{i=1}^{m} e_j(w)x_0^{(i)} \\ \frac{2}{m}\sum_{i=1}^{m} e_j(w)x_1^{(i)} \\ \vdots \\ \frac{2}{m}\sum_{i=1}^{m} e_j(w)x_n^{(i)} \end{bmatrix} =$$

which is equal to the inner product of:

$$\frac{2}{m}\sum_{i=1}^{m} e_j(w) \cdot [x_0^{(i)} x_1^{(i)} \cdots x_n^{(i)}] =$$

$$\frac{2}{m}\sum_{i=1}^{m} (w_{new} x_{new}^{(i)^T} - y^{(i)}) \cdot [x_0^{(i)} x_1^{(i)} \cdots x_n^{(i)}]$$

we can now transform the sum into arrays. So it will take the form of an inner product:

$$\nabla_w MSE(w) = -\frac{2}{m}(y^T - wX^T) \cdot X$$

where
$w = [w_{0,\ldots,w_n}]$
$y = [y^{(1)}, \ldots, y^{(m)}]^T$
and

$$X = \begin{bmatrix} x_0^1 & \cdots & x_n^1 \\ \vdots & \ddots & \vdots \\ x_0^m & \cdots & x_n^m \end{bmatrix}$$

and using these 2 properties of transpose arrays:
$(A - B)^T = A^T - B^T$ and $A^T B = B^T A$
we conclude that

$$\nabla_w MSE(w) = \frac{2}{m}(wX - y)^T \cdot X =$$

$$\nabla_w MSE(w) = \frac{2}{m} X^T \cdot (wX - y)$$

PART 2

First Approach

First time I run the model I **didn't used** any type of **Data Cleaning**. I loaded the DataSet(Train_df) and ValidationSet(val_df) as it is. In order to create the features of my model I Vectorized the words using three different

types of vectorizers (CountVectorizer, HashingVectorizer, TfidfVectorizer). The best results came from **TfidfVectorizer** and this was the one I used. I fit the vectorizer only using the tweets of TrainSet and then I transformed both train and validation set according to the previous fitted vectorizer. I didn't used any type of tokenizing because it wasn't nessecary in this case. Vectorizer do the work for us.

In order to use softmax regression I used the LogisticRegression model for many classes. I run it with GridSearch in order to find the best possible combination of parameters. The one I chose was {'C': 1, 'max_iter': 100, 'penalty': 'l2', 'solver': 'newton-cg'} and gave to me the best accuracy score.

Metrics	**Scores**
Accuracy	0.7287467134092901
F1_score	0.6775144075976213
Precision	0.7113942565393878
Recall	0.6611002547255244

Table 1: **Scores Without cleaning**

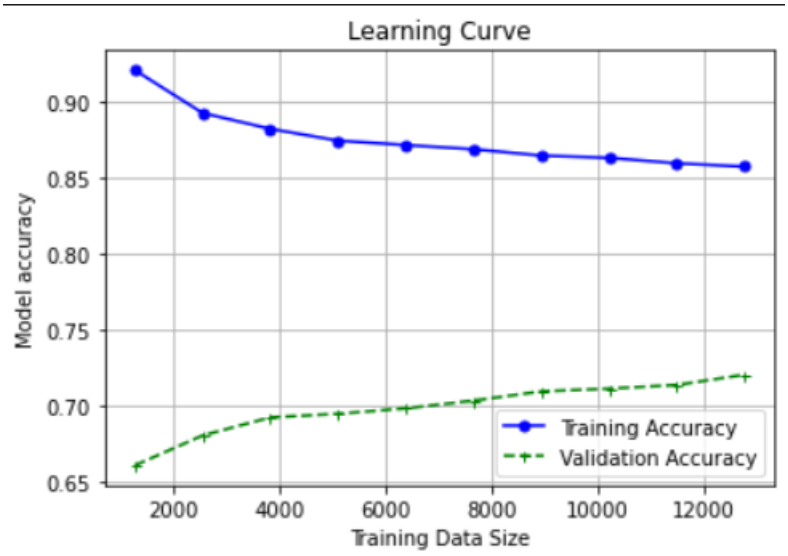

Figure 1: Learning Curve using **learning_curve**

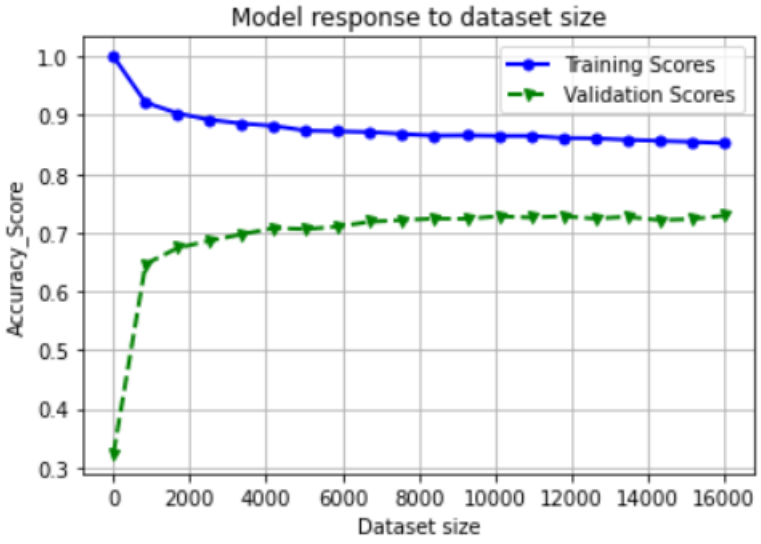

Figure 2: My Learning Curve using the given ValidationSet

Observations about learning Curves: The 2 lines are tending to meet its other if there were more training data probably. For sure the data that we take from the training set is representative for the validation set. The problem that we have here is to lower the gap between the 2 lines. This indicates that our model has high variance! Our model should have both low bias and low variance. In simple words we want the training line to have as high as possible accuracy(low bias) and the validation line to be as closer as possible to the training one (the gap between the lines to be small-low variance). Having said that we can see whether the model can do better, or maybe this is the best that could do to the specific problem.

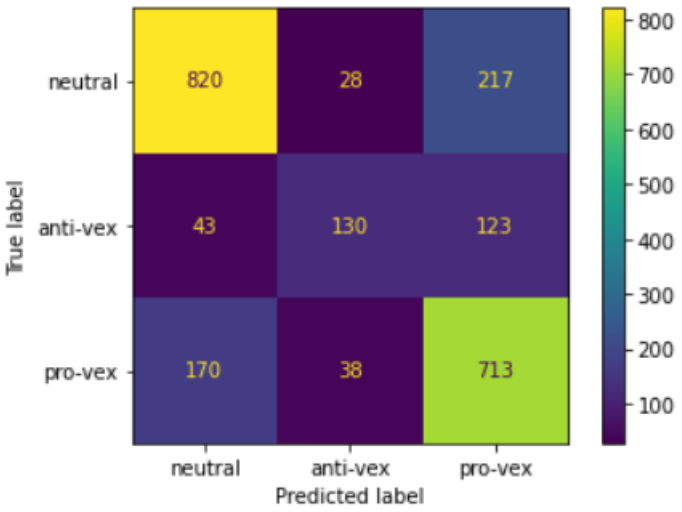

Figure 3: Confusion Matrix of the results

The Confusion matrix for the predictions and the true values is shown above. **The diagonal is the correct classified results**. As we can see our model does not hit very high Recalls but it does better according to Precision. This means that from the total positive results the correct ones are in a good percentage. But because the Recall is lower this means that the percentage of the truely correct results that should have returned is not very high.

Data Cleaning

Because the tweets contain a lot of words that may not help us to classify them (stopwords) we want to emit them. In addition to these words we are going to omit the punctuation of each tweet (any type of symbols may not help us). The next step that may help us to have a better generalization of the model is to remove the different links that are appeared in the tweets. The last step for the best possible generalization is to use stemming in our dataset. In this way we can organize-group words with similar meaning together. The results are below:

Metrics	**Scores**
Accuracy	0.7177914110429447
F1_score	0.6609783008796094
Precision	0.6980334658473129
Recall	0.6439696842321058

Figure 4: **Data Cleaning-Scores**

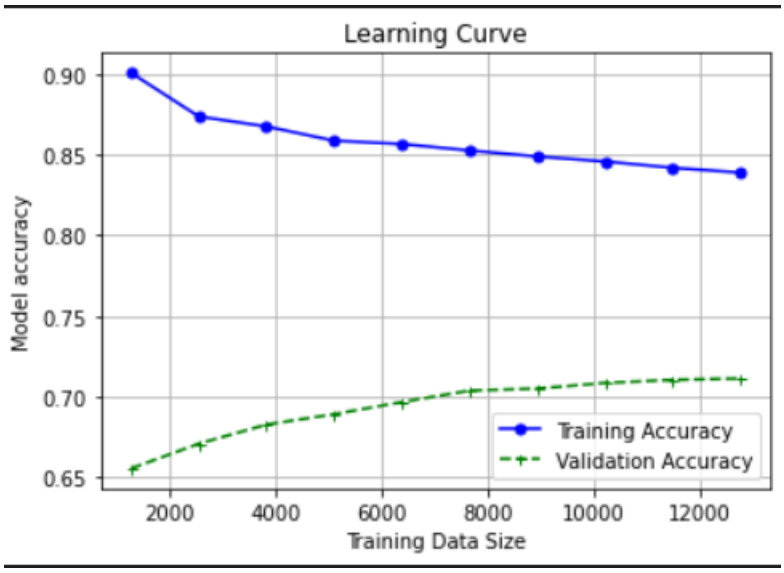

Figure 5: **Data Cleaning-Learning Curve using learning_curve**

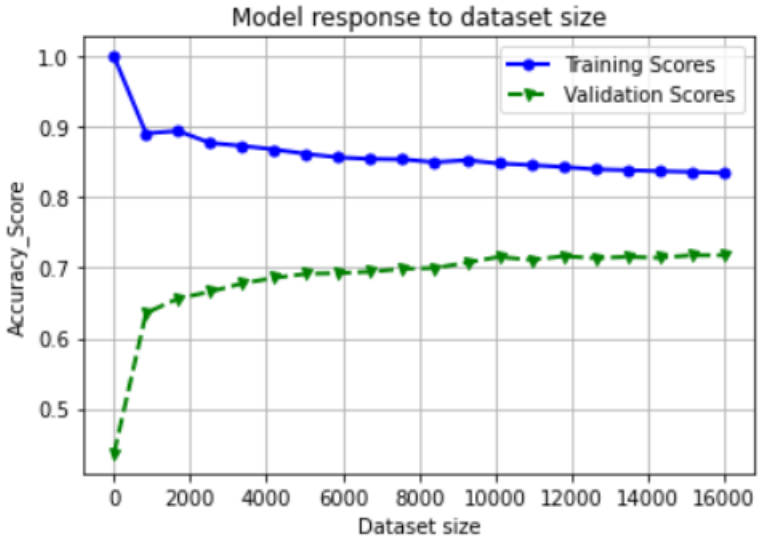

Figure 6: **Data Cleaning-My Learning Curve using the given ValidationSet**

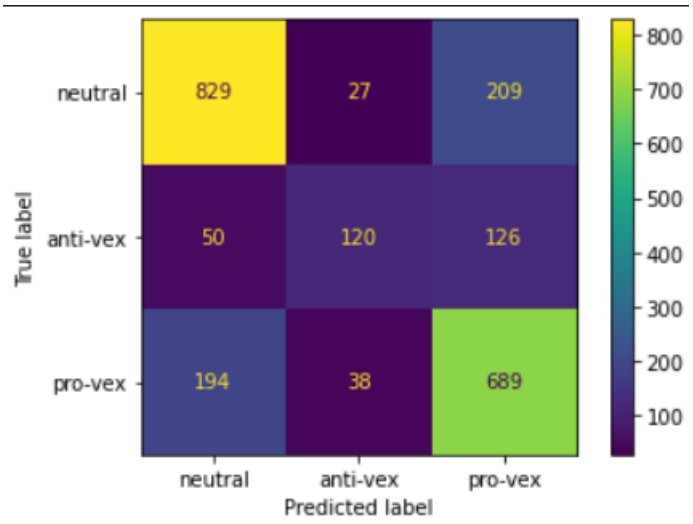

Figure 7: **Data Cleaning-Confusion Matrix of the results**

As we can see from the results above the model after the data cleaning has worse results than before. A little wierd but it can happen. In our case we omit many words that on the one hand could help our model to generalize and avoid overfitting, but on the other it seems that our model needed many of the features-"words" from before. The change we made, helped the model to find more neutral tweet, but it also classified correctly less anti-vex and pro-vex. Overall this change doesn't helped us a lot.

Including negative words

One step further is that if we omit the english-stopwords from the tweets we remove many words that describe the negative effect, such as **["haven't", "shouldn't", "isn't","wasn't"].** These words that indicates the "opposite" may helped the model to identify the pros and the antis easier. This is why I tried to includes those words and not remove them from the tweets.

Metrics	Scores
Accuracy	0.717353198948291
F1_score	0.6595640278702376
Precision	0.6961374285968747
Recall	0.6427946217080044

Figure 8: **Including negative words-Scores**

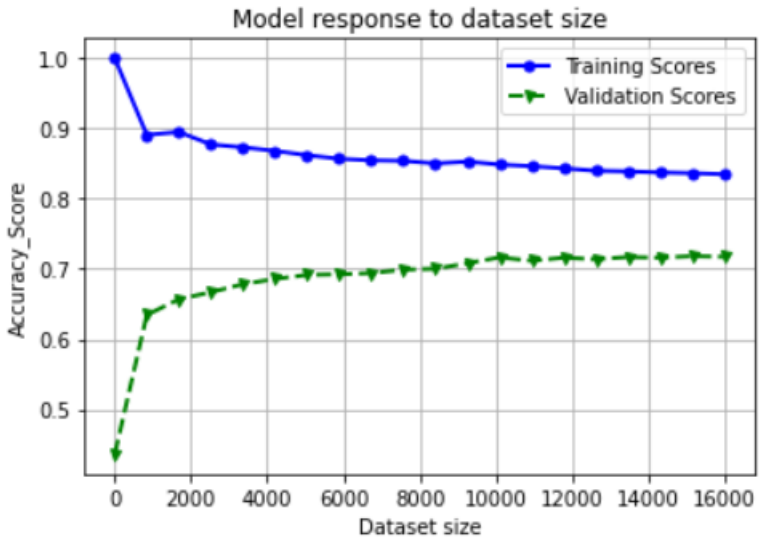

Figure 9: **Including negative words-My Learning Curve using the given ValidationSet**

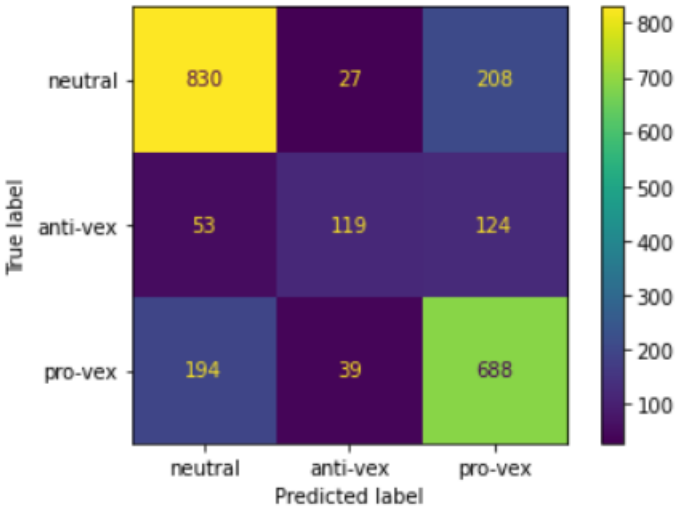

Figure 10: **Including negative words-Confusion Matrix of the results**

Even if it was an ok idea it didn't helped at all. The best results came with the original form of the data.

Last format

One last idea in order to deal with the overfitting of my model was to reduse the features that I had. In order to achieve this I used in **TfidfVectorizer** the parameters **min_df** and **max_df** . By following this logic I could produce a more stable model and lower the Variance of it. **ALTHOUGH** the model is having now higher bias.. As it was probably expected from the well-known trade-off!! After trying many values I could achieve the results below using the **min_df=0.0005,max_df=0.8**

Results for **NO-CLEANING** data:

Metrics	Scores
Accuracy	0.7261174408413672
F1_score	0.6735708797339278
Precision	0.7079955881508152
Recall	0.6564215727403321

Figure 11: **Last format-Scores**

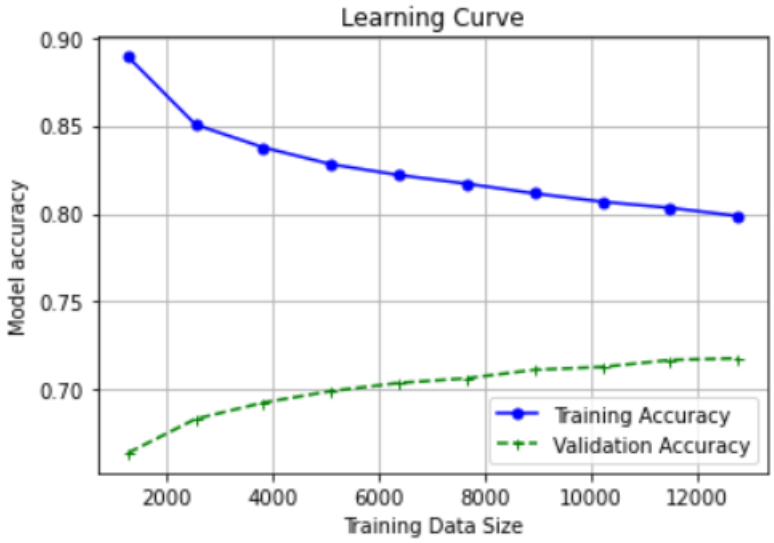

Figure 12: **Last format-Learning Curve using learning_curve**

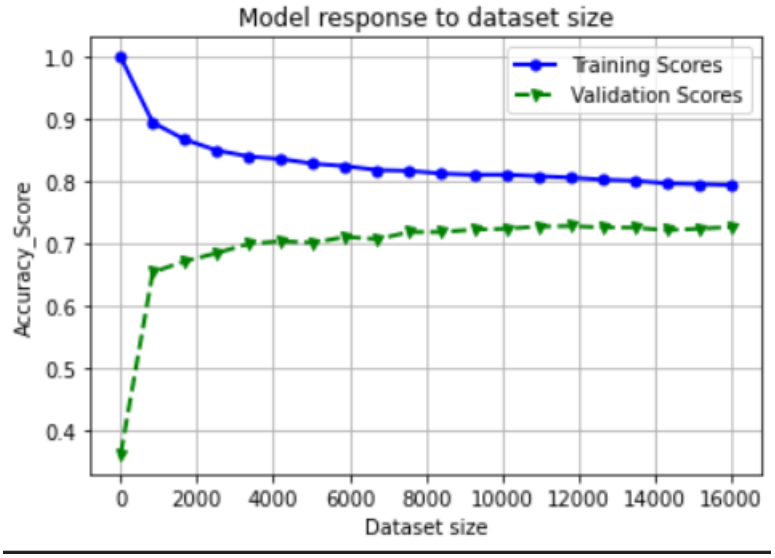

Figure 13: **Last format-My Learning Curve using the given Validation-Set**

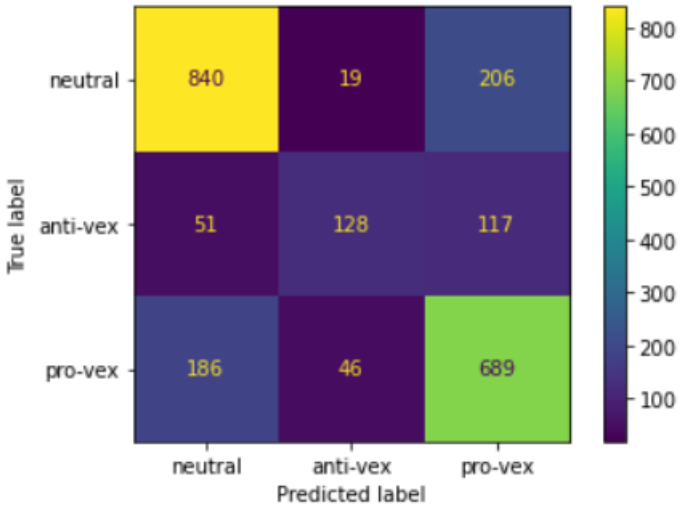

Figure 14: **Last format-Confusion Matrix of the results**

Conclusion: The model cannot do better than 73% . Even if it could it wouldn't be for a big value. Logistic regression has a ceiling and maybe in this particular problem there are models that could fit better. By trying many techniques I tried to generalize the model and make it have lower variance(escape

from overfitting). Using different ways didn't lead to a model with higher accuracy. But by trying to make the model more stable and lower the variance the bias went up! After all we have the variance-bias trayde-off. The parameters of logistic regression were picked with the help of Grid Search.

The current state of the code has the data uncleaned(gave the best scores) in combination with the TfidfVectorizer with the parameters min_df=0.0005,max_df=0.8. If I try to get lower variance it will make my bias higher and the other way around. These combinations I think make the model as balanced as possible(variance-bias trade-off).