CS4287 Project 1: **Multi Layer Perceptron**

Nutsa Chichilidze 18131956 - Ranya El-Hwigi 18227449

The Data Set **3**

Data Preprocessing **5**
Data Cleaning 5

The Network **6**
TensorFlow 6
Keras Sequential Model 6

The Loss Function **7**
Categorical Cross Entropy 7

The Optimiser **7**

Cross Fold Validation **8**

Results **8**

Evaluation of Results **9**
Accuracy and Loss 9
Predictions 10
Other metrics 10

Impact of Varying Hyperparameters **11**

References **11**

The Data Set

The data set we choose to work with is “Tweet Sentiment Extraction” [7] which we found on kaggle. It consists of 31,015 tweets which are divided between 27,481 tweets for training and 3,534 tweets for testing. The data is made up of textual information.

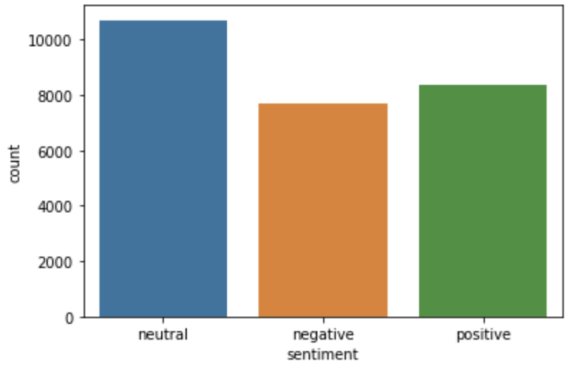
textID	text	sentiment
f87dea47db	Last session of the day http://twitpic.com/67ezh	neutral
96d74cb729	Shanghai is also really exciting (precisely -- skyscrapers galore). Good tweeps in China: (SH) (B...	positive
eee518ae67	Recession hit Veronique Branquinho, she has to quit her company, such a shame!	negative
01082688c6	happy bday!	positive
33987a8ee5	http://twitpic.com/4w75p - I like it!!	positive
726e501993	that`s great!! weee!! visitors!	positive
261932614e	I THINK EVERYONE HATES ME ON HERE lol	negative

Our aim is to create a classification model that can determine which tweets have a positive sentiment, which are neutral and which have a negative sentiment.

To the left is an extraction of the data that we will be using for our project. It’s contained in a csv file consisting of 3 columns. “textID” which is a unique ID for each piece of text, “text” which is the tweet itself, and “sentiment” which is the general sentiment of the tweet that we’ll be trying to classify from “text”.

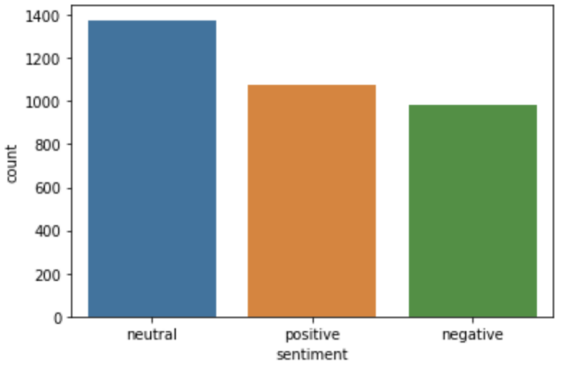
Biases with the data are that it doesn’t account for sarcasm in any of the tweets. However, we believe to detect sarcasm in tweets would require another model entirely. It also doesn’t account for the use of emojis and abbreviations to reflect sentiment.

We plotted our data to in many ways visualize the difference and association between the sentiments. For our visualizations we made use of the seaborn [8] library.



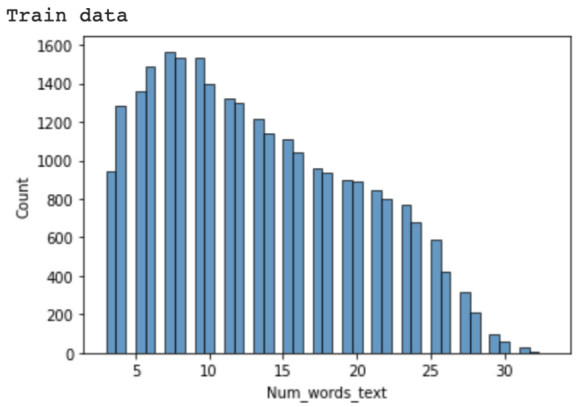
We found that our training data contains 10,704 neutral sentiment tweets, 8,375 positive sentiment tweets, and 7,673 negative sentiment tweets.

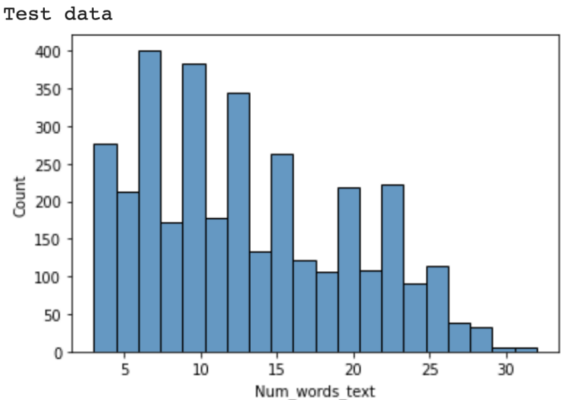
We also found that our test data contains 1,376 neutral sentiment tweets, 1,075 positive sentiment tweets and 983 negative sentiment tweets.



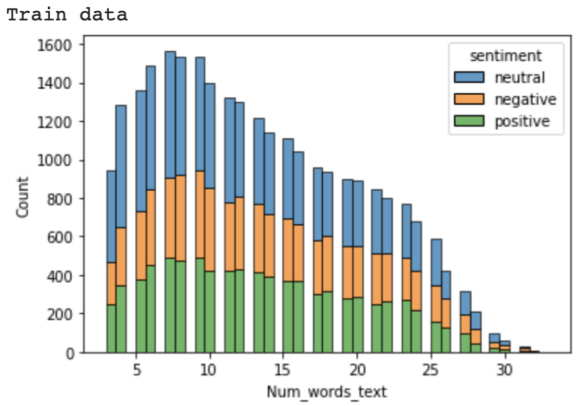
We were happy to see that the data is almost evenly distributed between the 3 sentiments in both train and test, giving our model a better chance at correctly classifying the sentiments of the tweets.

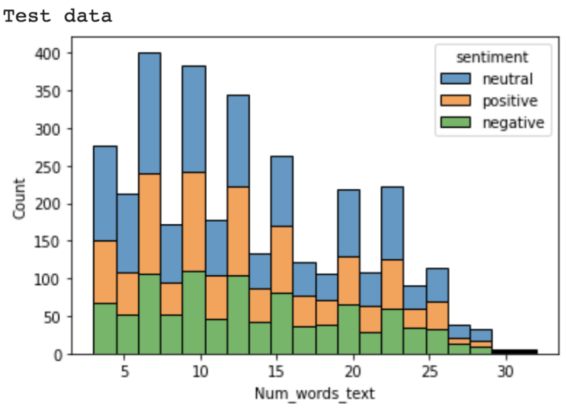
Next we moved on to visualizing the number of words our multi-layer perceptron would be working with in each tweet in order to classify them.



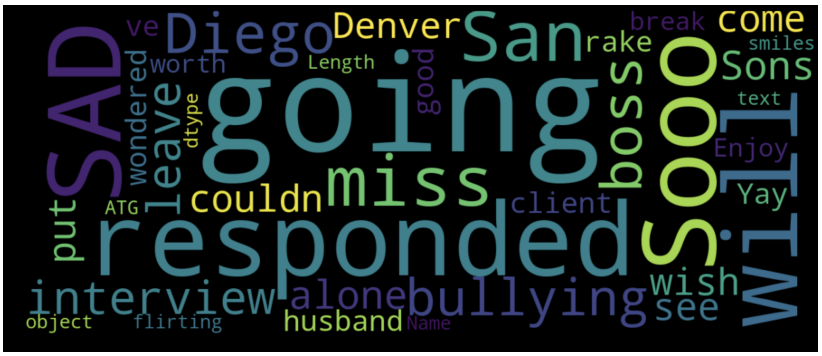


We found that the data for both the train and test sets provided are right skewed with most of the tweets containing between 5 and 10 words. Next we decided to find out how the number of words per tweet affects the sentiment of the tweet.





As we can see from the distributions the number of words in the data is not associated with the sentiment of the tweet. The number of words in the tweets classified as either of the three sentiments are all right skewed. We were happy to see this as it illustrated that the data doesn't include biases regarding the number of words determining the sentiment.



Finally, we were interested in seeing the most common words appearing in our data so we constructed a word cloud.
(we came across the technique to achieve this in the referenced [10] tutorial).

Now that we had a clear view of our data, we were ready to begin the preprocessing of data.

Data Preprocessing

Firstly we started our preprocessing by reading in our csv files into PandasDF, one for our train data and another for our test data.

Secondly, in our data exploration we came across some data that we knew we wouldn't need for training our specific model that would only be classifying the tweets sentiment. Hence, we dropped the following data from the dataframes:

- The "selected_text" column from the train data as we wouldn't be working on extracting the text that supported the tweets sentiment.
- Any rows that were missing the text field.
- Any duplicating data.

Thirdly, we had to clean our data due to the usual messiness of textual data. There were a few things we had to consider for the cleaning that we'll discuss separately in the "Data Cleaning" section.

Fourthly, we split our data into training and validation sets, and converted them and the test data frame into a list format which is compatible with tensorflow.

Finally, we used sklearn.preprocessing.LabelEncoder [11] class from the scikit package to convert our textual data into numerical values where each label will be represented by a number so that our MLP can output it. For the same reason, we included a hub layer to our model to convert the text sentences into numerical input so that it can be accepted by the input layer of the MLP.

Data Cleaning

Text based-data, especially when working on a dataset such as tweets can be quite messy. There are no textual rules that need to be followed when creating a tweet; all you have to do is type your thoughts and press "tweet". Due to the nature of our data, we decided to perform multiple data cleaning procedures such as

- Expanding contractions: i.e. we'll -> we will, don't -> do not, etc. (we utilized python's contractions[1] package for this procedure.)
- Removing URLs and HTML tags, with the help of the RegEx generator.[2]
- Removing non-ASCII characters.
- Removing emojis (since we are working on tweets, we come across a lot of emojis that do not have any meaning to our model). To avoid disruptions, we remove emojis altogether.[3]
- Removing punctuations.
- Correcting spelling mistakes using TextBlob. [4]

The Network

TensorFlow

Tensorflow is a popular platform that provides a multitude of functionalities, tools and libraries for machine learning. It is fast, efficient and easy to use for beginners. Throughout our implementation, we often referred to official tutorials by Tensorflow, especially one that described the details of text classification[9]. This helped us identify the correct ways to structure our code, parse our data and build our model.

When researching Tensorflow, we decided to use the Keras framework which would give us easy-to-use functionalities for model building, training and testing.

Keras Sequential Model

For the structure of our model, we picked Keras' Sequential model. This is suited for a plain stack of layers with exactly 1 input and output tensors. It consists of:

- The Hub layer - one of the most important parts of our model structure, it is a pre-trained text embedding layer, which allows us to perform transfer learning i.e. translate the inputted sentences into numbers. This way they get converted into embedding vectors.
- Dropout layers - regularization technique to reduce overfitting (more accuracy on the training data and less accuracy on the test/unseen data). This allows our model to be more robust.
    - Rate - the rate attribute on the dropout layer represents the fraction of input to drop.
- Dense layers - the dense layer performs the below operation on its input

    ```
    output = activation(dot(input, kernel) + bias)
    ```

    - Activation function "reLU" - is one of more popular non-linear activation functions, which might be the most popular in neural networks due to its ability to not activate all neurons at the same time. This allows the activation function to be significantly more computationally efficient.
    - Activation function "sigmoid" - a good activation function to use for classification. It transforms values generated by the model in the range of 0 to 1 which then allows for effective and fast classification. (Mostly used for binary classification but works for other models too.)
    - Kernel regularization - allow us to penalize layer parameters during optimization. These are summed into the loss function for network optimization.[5]

The Loss Function

```
# the optimizer Adam allows us to maximize our categorical accuracy
model.compile(loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
              optimizer='adam',
              metrics=["CategoricalAccuracy"])
```

Categorical Cross Entropy

$$CCE(p,t) = -\sum_{c=1}^{C} t_{o,c} \log\left(p_{o,c}\right)$$

Categorical cross entropy is a loss function implemented by Keras which is designed for multi-class classification. Many machine learning problems have binary outputs, for example the problems that can be answered by a binary "yes" or "no". Our model categorizes the input into 3 different classes: neutral, positive or negative. The way this loss function works is that It takes in two inputs, prediction p and target t, and gets recomputed for every sample and gets merged together. The advantage of this approach is that categorical cross entropy allows us to compute the loss value for multiclass classification problems – while remaining flexible with respect to the actual target class.[12]

The Optimiser

We researched text classification implementations across the internet to understand which Keras optimizer would be best suited for our problem. We found that Adam(Adaptive Moment Estimation)[6] is generally the more preferred Keras optimizer and widely used on a range of problems due to its efficiency.

```
# the optimizer Adam allows us to maximize our categorical accuracy
model.compile(loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
              optimizer='adam',
              metrics=["CategoricalAccuracy"])
```

We use the Adam optimization algorithm during the compilation of our model.

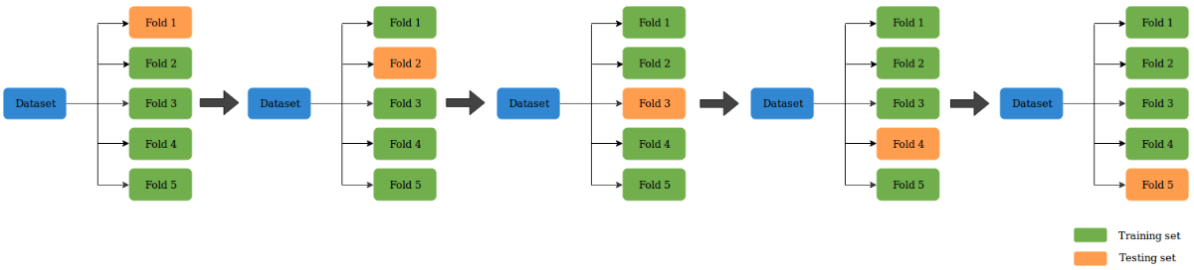
Cross Fold Validation

It can be difficult to evaluate a machine learning model, usually we split the data into a train set and a test set, we train our model on our train data and then use the test portion of the data to test our model and determine its accuracy. However, this isn't a reliable way to determine model accuracy as how a model performs on one test set can greatly differ from how it will perform on another test set [13]. Also there exists the risk of the model receiving data that they have memorised and so we get a high accuracy score and then it performs very poorly on any unseen data. Because the data we have access to is of course limited, Cross Fold Validation offers a clever way to use our limited data and still be able to have multiple test and train sets from it.

We felt this was specifically important for our data because we risk the chance of the test set containing a lot of one of the three sentiments e.g. positive, that it's good at and then we get a high accuracy that doesn't truly reflect our model. Or it could even not get tested on one of the sentiments at all e.g. the test set only includes positive and negative sentiments and no neutral sentiments. So through employing Cross Fold Validation we know our model has been tested on every sentiment and the average of the accuracy results from the folds would be a true reflection of our model.

The main basics of Cross Fold Validation are:

1. It shuffles the data set randomly to avoid any bias on how the data was stored.
2. It takes your data and splits into the number of folds that you specify. In our model we specified a total of 5 folds.
3. Then in each iteration the test set will be one of those folds and the training set the remaining folds.
4. It fits the training set to the model as many times as specified in the epochs, and then uses the test set of that fold to test the model and evaluate its mean accuracy for that fold.



5 Fold Cross Validation[13]

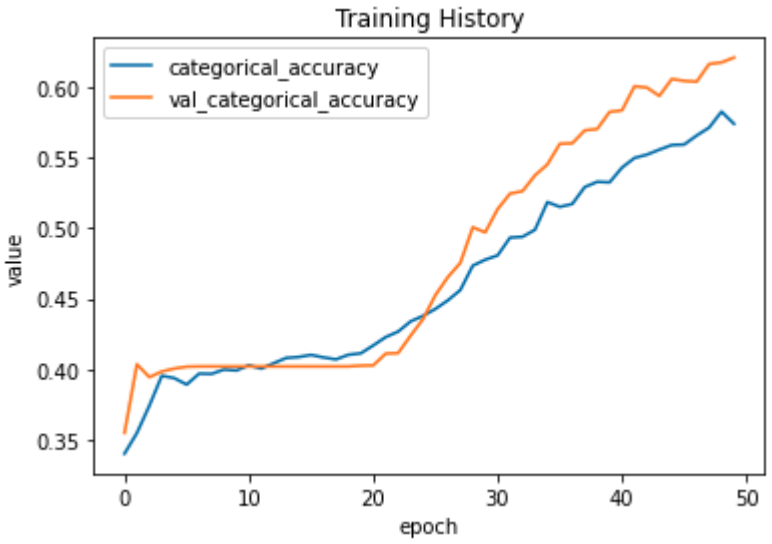
Results

For our model we focused on calculating two metrics, the loss and the categorical accuracy. The loss function as mentioned previously was Categorical Cross Entropy and we ended up with a final loss of 0.86.
The categorical accuracy, which calculates the percentage of predicted values that match the actual values, of our data is 70%.

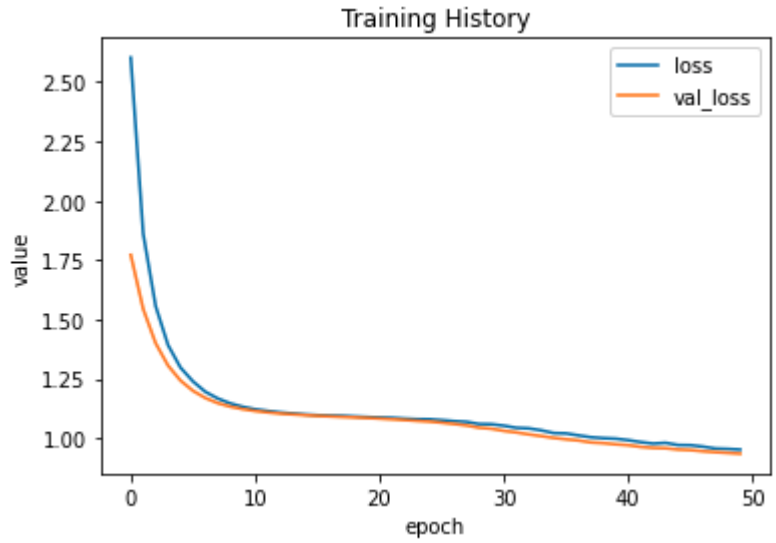
Evaluation of Results

Accuracy and Loss

As mentioned our accuracy is evaluated to 70%, so that means we have a 70% chance of classifying the sentiment of a tweet correctly. We are happy with this accuracy as even when looking through the tweets ourselves we classified them wrong a few times. Because our model does use natural language and the data set isn't that big we think it's a fair enough accuracy for the time being.



Here we plotted the accuracy value of the model increasing with each epoch. We were happy to see our accuracy value climbing and that there isn't any overfitting occurring.



Here we plotted the loss value of the model decreasing with each epoch. We were happy to see our loss value converging and that there isn't any overfitting occurring.

Overall, we would have liked to get a lower loss but we were happy to see our loss decrease as our accuracy increased.

Predictions

whered you go	neutral	neutral
watching body of liesgood film	positive	neutral
happy mothers day mumm xoxo	positive	positive
so i really need to put the laptop down start...	neutral	negative
im sorry at least its friday	negative	negative
feels sorry every time im printing out i use l...	negative	negative
4n ma rog never heard of it esti beat acum ...	negative	neutral
i always forget something when i travel i am ...	neutral	neutral
should have left car and walked home i might n...	neutral	neutral
im only updating this so that bretts phone bee...	neutral	neutral
hi there i agree small children should be r...	positive	neutral
hope ur havin fun in da club	positive	positive
i miss my old phone it worked so good until i ...	neutral	neutral
thinks sg is wonderful	positive	positive
I am really sorry i know wallah how you feel...	negative	negative
is watching acoustic performances in the mood...	neutral	neutral
ill oscillate from one to the other	neutral	neutral

Here is an extraction of our model's predictions in the right column and the actual result in the left column.
As we can see, a fair amount of the results were classified correctly.
The sentiment the model most seemed to struggle with is neutral as most of its mistake is classifying non-neutral tweets as neutral.

Other metrics

	precision	recall	f1-score	support
positive	0.78	0.69	0.73	1075
negative	0.74	0.52	0.61	983
neutral	0.58	0.76	0.66	1376
accuracy			0.67	3434
macro avg	0.70	0.66	0.67	3434
weighted avg	0.69	0.67	0.67	3434

Precision is the ratio of the number of true positives (the amount the model said were right and they were) and the number of false positives (the amount the model said were right and they weren't)[15] for each label. Our model presented with a precision of 0.78 for "positive" classification, 0.74 for "negative" classification and 0.58 for "neutral" classification. So our model was most precise with positive sentiment tweets and rarely incorrectly classified a tweet as positive and was least precise with neutral sentiment tweets and frequently classified a tweet incorrectly as neutral.

Recall is the ratio of the number of true positives (the amount it got correct) and the number of false negatives (the number it got wrong)[14] for each label. Our model presented with a recall of 0.69 for "positive" classification, 0.52 for "negative" classification and 0.76 for "neutral" classification. So our model did best at classifying "neutral" tweets and did worst at classifying "negative" tweets.

F1-score is the weighted average of recall and precision[16]. Our model presented with a f1-score of 0.73 for "positive" classification, 0.61 for "negative" classification and 0.66 for

“neutral” classification. So our model overall performed best with positive sentiments and worst with negative sentiments.

Impact of Varying Hyperparameters

The multi layer perceptron has a range of external configuration variables that can be custom tuned depending on the model and dataset you're working with. A correct choice of a combination of hyper parameters can drastically improve the performance of your model, however, a wrong combination can cause a lot of damage. The tuning of these parameters must be done carefully.

We focused on tuning:

- epochs - the number of times the learning algorithm will work through the entire dataset
- batch size - the number of samples the model works through before updating internal parameters

We created sets of different values for each of the parameters and trained our model for all the combinations between them. The values we chose were either random or found in other models (i.e. 60 is a frequent number for epochs and 128 is one we often saw for the batch size). We then picked the combination with the highest number of categorical accuracy and generated predictions using these values for our hyper parameters.
Through our experimentation of the different hyperparameters we were expecting the highest number of epochs and the highest batch value to yield the best results. However, after examining the results of the different variations we found that wasn’t the case and our best results yielded from epochs = 60 and batch = 20.

```
Epoch 60/60
536/536 [==============================] - 3s 6ms/step - loss: 0.8089 - categorical_accuracy: 0.6890 - val_loss: 0.8465 - val_categorical_accuracy: 0.6763
Score for fold 2: loss of 0.8367669582366943; categorical_accuracy of 67.41409301757812%
```

Result of epoch = 60 batch = 20

```
Epoch 100/100
84/84 [==============================] - 1s 12ms/step - loss: 0.8499 - categorical_accuracy: 0.6617 - val_loss: 0.8680 - val_categorical_accuracy: 0.6606
Score for fold 4: loss of 0.8582397103309631; categorical_accuracy of 66.42399430274963%
```

Result of epoch = 100 batch = 128

References

[1] https://pypi.org/project/contractions/
[2] https://regexr.com/
[3] https://stackoverflow.com/questions/33404752/removing-emojis-from-a-string-in-python
[4] https://towardsdatascience.com/textblob-spelling-correction-46321fc7f8b8
[5] https://keras.io/api/layers/regularizers/
[6] https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6
[7] https://www.kaggle.com/c/tweet-sentiment-extraction/overview
[8] https://seaborn.pydata.org/generated/seaborn.histplot.html
[9] https://www.tensorflow.org/tutorials/keras/text_classification
[10] https://www.datacamp.com/community/tutorials/wordcloud-python
[11] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html
[12]https://www.machinecurve.com/index.php/2019/10/22/how-to-use-binary-categorical-crossentropy-with-keras/#categorical-crossentropy-for-multiclass-classification
[13] https://medium.datadriveninvestor.com/k-fold-cross-validation-6b8518070833
[14] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html
[15] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html
[16] https://www.ritchieng.com/machinelearning-f1-score/