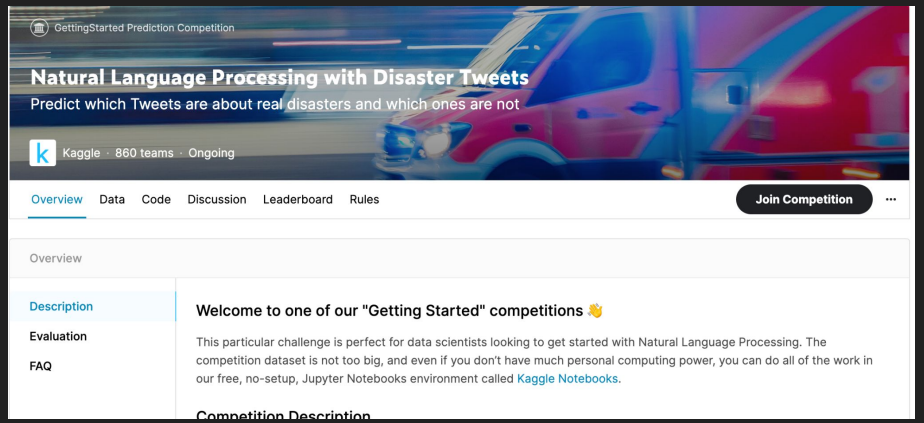
NLP with Disaster Tweets

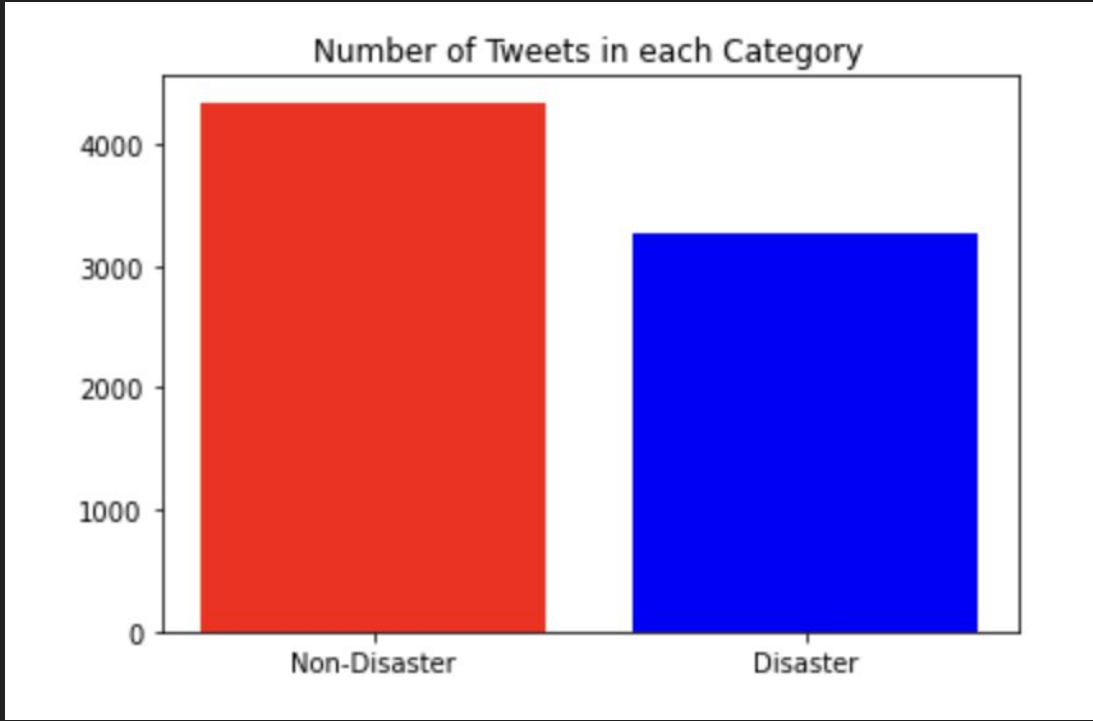
Kevin Hamakawa and Taylor Kim

Motivation

Utilize NLP to classify tweets with disaster-related words as disaster and non-disaster.

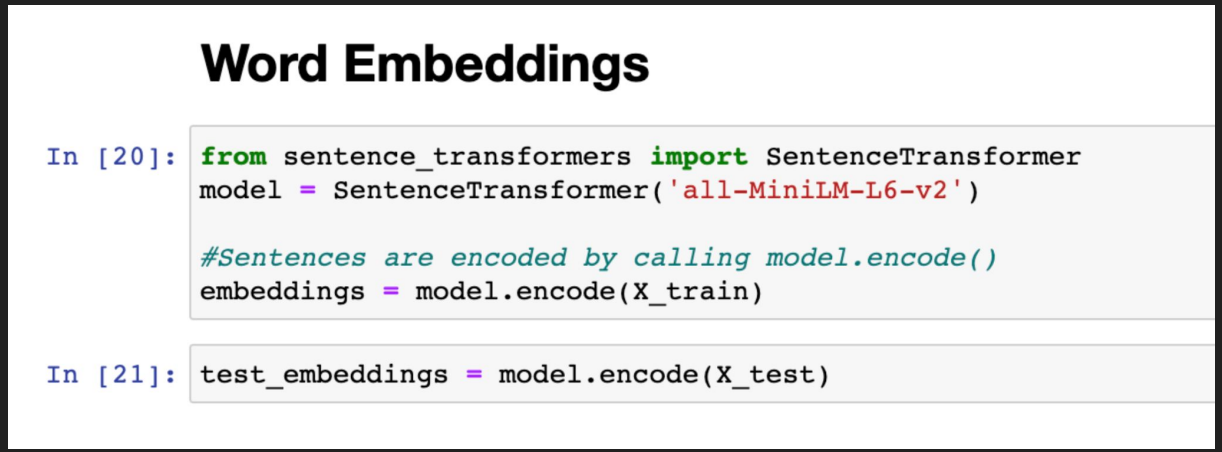


EDA


Number of Tweets in each Category
4000
3000
2000
1000
0
Non-Disaster
Disaster

Approach

- Vectorize the sentences
- Using sentence transformers to make word embeddings for each sentence



Word Embeddings

In [20]:
```
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('all-MiniLM-L6-v2')

#Sentences are encoded by calling model.encode()
embeddings = model.encode(X_train)
```

In [21]:
```
test_embeddings = model.encode(X_test)
```

Approach

- Each Tweet gets converted to a vector of length 384 composed of numbers
- For example:

```
: train_df["text"][0]

: 'Our Deeds are the Reason of this #earthquake May ALLAH Forgive us all'
```

```
In [57]: embeddings[0]

Out[57]: array([-1.82754919e-02, -5.15176877e-02,  1.95072703e-02,  1.09355217e-02,
                 7.46812746e-02,  4.37218137e-02, -8.01321939e-02, -4.74993177e-02,
                -2.37632226e-02,  2.84239464e-02,  5.68280034e-02, -9.67782550e-03,
                -3.11478321e-02,  3.97447012e-02,  2.11315174e-02,  5.87576814e-02,
                 3.60816577e-03,  7.78805790e-03, -7.42244646e-02,  5.60301356e-02,
                 1.97820030e-02,  2.44401190e-02,  4.13863361e-02,  3.37453261e-02,
                -2.20054556e-02, -2.04689484e-02,  7.95850996e-03, -4.70432341e-02,
                -1.07182130e-01, -8.26304853e-02,  9.33964550e-03,  4.62996438e-02,
                 4.62390706e-02, -3.57235819e-02,  2.48602238e-02, -2.15252116e-02,
                 7.12632686e-02, -7.53818676e-02,  1.24257870e-01, -1.75643098e-02,
                -3.17390822e-02, -9.17508230e-02,  7.86719546e-02,  6.36794493e-02,
                 9.44883935e-03,  3.36217545e-02,  9.89887770e-03, -3.33410613e-02,
                 1.43310605e-02, -3.27223651e-02, -7.11811408e-02, -6.68847933e-02,
                -3.99896316e-03, -7.76442140e-02, -5.47121046e-04,  2.14312524e-02,
                 5.40910736e-02,  5.58665544e-02,  3.14942002e-02,  1.91090871e-02,
                 7.30824545e-02, -7.42673203e-02, -6.68672612e-03, -3.34590413e-02,
                 3.17173265e-02,  3.30928527e-03, -3.81083004e-02,  6.82716491e-03,
                 4.07549739e-02, -5.68232173e-03, -3.42633482e-03,  5.97031377e-02,
                -4.89715375e-02,  1.42453657e-02, -8.82842392e-03,  4.36788499e-02,
```

Applying Machine Learning Process

- Once we split and embedded all the data, we can now apply it to different classifying models.
- SVM, MLP, KNN, Random Forest, Naive Bayes

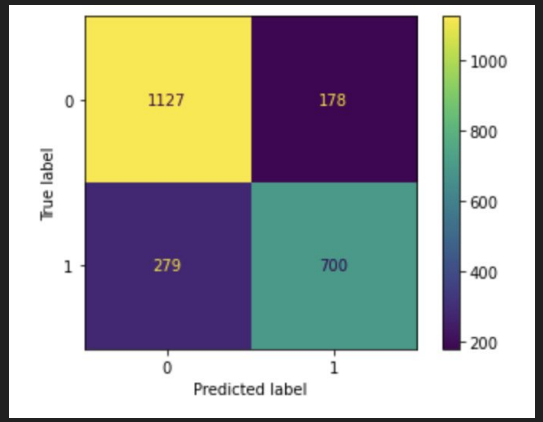
SVM

Baseline Accuracy:

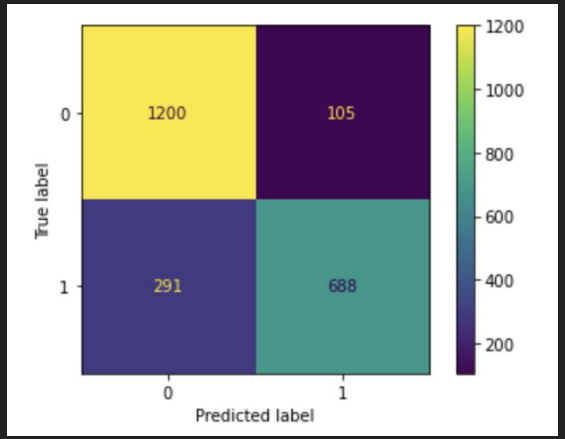
Kernel = “Linear”: 81.92%

Trying SVM

```
In [22]: from sklearn import svm

In [61]: clf = svm.SVC(kernel = 'linear')
         clf.fit(embeddings, y_train)

Out[61]: SVC(kernel='linear')

In [62]: # score of accuracy.
         clf.score(test_embeddings, y_test)

Out[62]: 0.8191768826619965

In [63]: clf = svm.SVC(kernel = 'poly')
         clf.fit(embeddings, y_train)

Out[63]: SVC(kernel='poly')

In [64]: # score of accuracy.
         clf.score(test_embeddings, y_test)

Out[64]: 0.8314360770577933
```



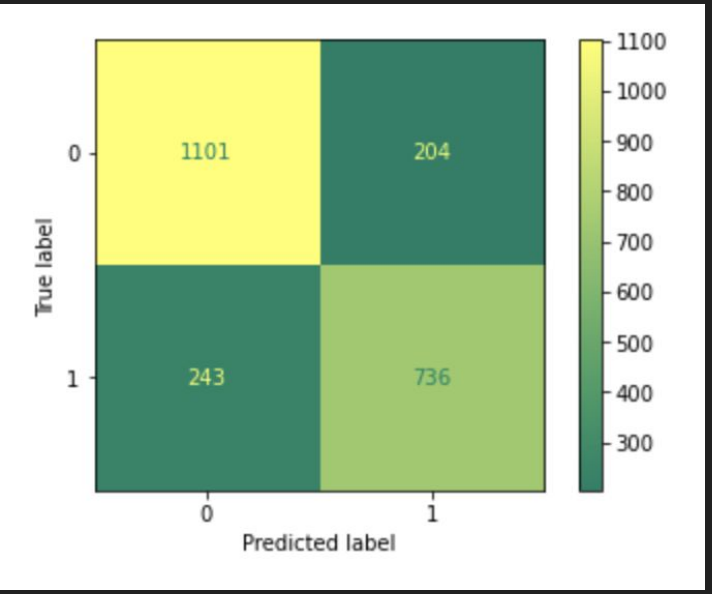
Kernel = “Poly”: 83.14%



MLP

Baseline Accuracy:

80.43%



Trying on MLP

In [25]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
```

In [65]:
```
clf = MLPClassifier(random_state=1, max_iter=300)
clf.fit(embeddings, y_train)
```

```
/Users/taylorkim/opt/anaconda3/lib/python3.9/site-packages/sklearn/neural_netwo
ergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and the
  warnings.warn(
```

Out[65]:
```
MLPClassifier(max_iter=300, random_state=1)
```

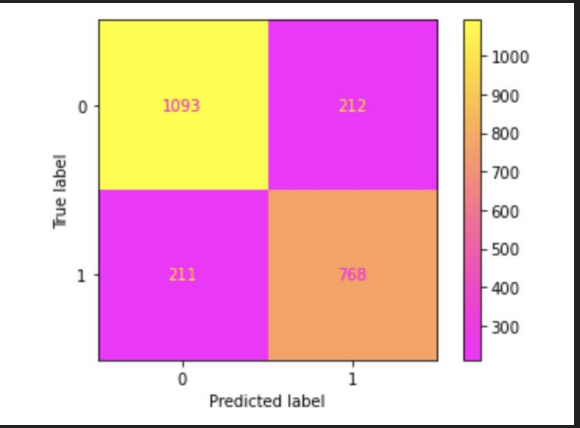
In [66]:
```
clf.score(test_embeddings, y_test)
```

Out[66]:
```
0.8042907180385289
```

KNN

Baseline Accuracy:

81.48%



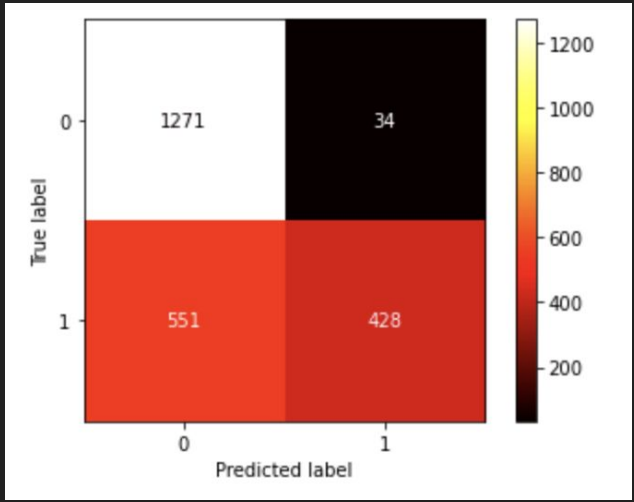
Trying on KNearest Neighbors

```
In [28]: from sklearn.neighbors import KNeighborsClassifier

In [29]: clf = KNeighborsClassifier(n_neighbors=10)

In [30]: clf.fit(embeddings, y_train)
Out[30]: KNeighborsClassifier(n_neighbors=10)

In [31]: clf.score(test_embeddings, y_test)
Out[31]: 0.8147985989492119
```

Random Forest

Baseline Accuracy:

74.39%




```
In [86]: from sklearn.ensemble import RandomForestClassifier

In [87]: clf = RandomForestClassifier(max_depth=2, random_state=0)

In [88]: clf.fit(embeddings, y_train)
Out[88]: RandomForestClassifier(max_depth=2, random_state=0)

In [89]: clf.score(test_embeddings, y_test)
Out[89]: 0.7438704028021016
```

Conclusion

- Overall the SVM model was the strongest, producing an accuracy of 83.14%.
- But as you can see, a lot of the models produced accuracies that were very similar.
- In this specific project, we did a lot of trial and error and playing around with the different models.
- Looking forward, we would try to find ways to optimize the parameters using more specific methods.