AI – 511 ML PROJECT
WHY SO HARSH?

- TEAM – ML CODERS
- IMT2020042 – GOUSEPEER ARELLA
- IMT20205I3 - VISHAL CHEETI

PREPROCESSING

1. Expand Contractions
2. LowerCase
3. Removing Emoticons
4. Remove Punctuations
5. Remove stopwords
6. Remove words and digits containing digits
7. Rephrase Text
8. Removing Whitespaces
9. Lemmatization
10. Text Vectorization

EXPANDING CONTRACTIONS

```
train_df_text_list = []
test_df_text_list = []

for line in train_df_text:
    train_df_text_list.append(line)

for line in test_df_text:
    test_df_text_list.append(line)


for i in range(len(train_df_text_list)):
    expanded_words = []
    for word_in_line in train_df_text_list[i].split():
        try:
            expanded_words.append(contractions.fix(word_in_line))
        except:
            expanded_words.append(" ")
    expanded_text = ' '.join(expanded_words)
    train_df_text_list[i] = expanded_text
```

- Contractions is the shortened form of words like **don't** stands for **do not, aren't** stands for **are not.** We need to expand this contraction in data for better data analysis.
- Import contractions library and apply contractions.fix to expand the contracted words

LOWER CASE

Lowe Case

```
for i in range( len(train_df_text_list)):
    train_df_text_list[i]= train_df_text_list[i].lower()
train_df_text_list[2]
```

```
'sandbox?? i did your madre did in the sandbox'
```

- If the text is in same case it is very easy for machine to interpret the words because the lowercase and uppercase are treated differently by the machine.
- So, we need to make the text in same case and the most preferred case is lowercase to avoid such problems.

REMOVING EMOTICONS

```
for i in range(len(train_df_text_list)):
    emoticon_removed = train_df_text_list[i]
    for emot in EMOTICONS_EMO:
        emoticon_removed = emoticon_removed.replace(emot, " "+EMOTICONS_EMO[emot].replace(" ","_"))
        train_df_text_list[i] = emoticon_removed

for i in range(len(test_df_text_list)):
    emoticon_removed = test_df_text_list[i]
    for emot in EMOTICONS_EMO:
        emoticon_removed = emoticon_removed.replace(emot, " "+EMOTICONS_EMO[emot].replace(" ","_"))
        test_df_text_list[i] = emoticon_removed
```

- Machine cant read emojis and emoticons, so we convert the emoticons into their respective mood text like smile, cry etc

REMOVING PUNCTUATIONS

```
train_punc_removed=[]
for i in range(len(train_df_text_list)):
    train_punc_removed = ""
    for char in train_df_text_list[i]:
        if char in string.punctuation:
            train_punc_removed+=" "
        else:
            train_punc_removed+=char
    train_punc_removed_join = ''.join(train_punc_removed)
    train_df_text_list[i]= train_punc_removed_join
```

- There are total of 32 main punctuations that should be taken care of.
- By using the string module with a regular expression to replace a punctuation with an empty string.

REMOVING WORDS AND DIGITS CONTAINING DIGITS

- If there exists a word containing the digits, or just existence of digits then we remove such words as they do not provide any required information to us regarding the sensitivity of the text.

```
train_num_removed = []
for i in range(len(train_df_text_list)):
    train_num_removed = ' '.join(s for s in train_df_text_list[i].split() if not any(c.isdigit() for c in s))
    train_df_text_list[i] = train_num_removed
```

REMOVING STOPWORDS

- Stopwords are the most commonly occurring words in a text which do not provide any valuable information.
- stopwords like they, there, this, where etc are some of the stopwords. NLTK library is a common library that is used to remove stopwords and include approximately 180 stopwords which it removes. If we want to add any new word to a set of words then it is easy using the add method.
- Here we added new words like http,https,www.

REMOVING STOPWORDS

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
```

```
stop_words = set(stopwords.words('english'))
stop_words.add("http")
stop_words.add("https")
stop_words.add("www")

train_stopw_removed = []
for i in range(len(train_df_text_list)):
    train_stopw_removed = [word for word in train_df_text_list[i].split() if word not in stop_words]
    train_stopw_removed_join = ' '.join(train_stopw_removed)
    train_df_text_list[i] = train_stopw_removed_join

test_stopw_removed = []
for i in range(len(test_df_text_list)):
    test_stopw_removed = [word for word in test_df_text_list[i].split() if word not in stop_words]
    test_stopw_removed_join = ' '.join(test_stopw_removed)
    test_df_text_list[i] = test_stopw_removed_join
```

REMOVING WHITESPACES

- Most of the time text data contain extra spaces or while performing the above preprocessing techniques more than one space is left between the text so we need to control this problem.
- We used regular expression to solve this problem.

```
for i in range(len(train_df_text_list)):
     train_df_text_list[i] = re.sub(r'\s+', ' ',train_df_text_list[i])
```

STEMMING VS LEMMATIZATION

Stemming	Lemmatization
Stemming is a process that stems or removes last few characters from a word, often leading to incorrect meanings and spelling.	Lemmatization considers the context and converts the word to its meaningful base form, which is called Lemma.
For instance, stemming the word 'Caring' would return 'Car'.	For instance, lemmatizing the word '**Caring**' would return '**Care**'.
Stemming is used in case of large dataset where performance is an issue.	Lemmatization is computationally expensive since it involves look-up tables and what not.

CONCLUSION OF THE ABOVE SLIDE

- A lot of knowledge and understanding about the structure of language is required for lemmatization.
- Hence, in any new language, the creation of stemmer is easier in comparison to lemmatization algorithm.
- Lemmatization and Stemming are the foundation of derived (inflected) words and hence the only difference between lemma and stem is that lemma is an actual word whereas, the stem may not be an actual language word.
- Lemmatization uses a corpus to attain a lemma, making it slower than stemming. Further, to get the proper lemma, you might have to define a parts-of-speech. Whereas, in stemming a step-wise algorithm is followed making it faster.

LEMMATIZATION

```
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('averaged_perceptron_tagger')
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
def get_wordnet_pos(word):
    tag = word[0]
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}
    return tag_dict.get(tag)


for i in range(len(train_df_text_list)):
    line = train_df_text_list[i]
    tok_list = nltk.pos_tag(nltk.word_tokenize(line))
    pos_final = list(map(lambda x: (x[0], get_wordnet_pos(x[1])),tok_list ))
    train_lemma_line = []
    for word, tag in pos_final:
        if tag is None:
            train_lemma_line.append(word)
        else:
            train_lemma_line.append(lemmatizer.lemmatize(word, tag))
    lemma_line = " ".join(train_lemma_line)
    train_df_text_list[i] = lemma_line
```

```
for i in range(len(train_df_text_list)):
    line = train_df_text_list[i]
    tok_list = nltk.pos_tag(nltk.word_tokenize(line))
    pos_final = list(map(lambda x: (x[0], get_wordnet_pos(x[1])),tok_list ))
    train_lemma_line = []
    for word, tag in pos_final:
        if tag is None:
            train_lemma_line.append(word)
        else:
            train_lemma_line.append(lemmatizer.lemmatize(word, tag))
    lemma_line = " ".join(train_lemma_line)
    train_df_text_list[i] = lemma_line

for i in range(len(test_df_text_list)):
    line = test_df_text_list[i]
    tok_list = nltk.pos_tag(nltk.word_tokenize(line))
    pos_final = list(map(lambda x: (x[0], get_wordnet_pos(x[1])),tok_list ))
    test_lemma_line = []
    for word, tag in pos_final:
        if tag is None:
            test_lemma_line.append(word)
        else:
            test_lemma_line.append(lemmatizer.lemmatize(word, tag))
    lemma_line = " ".join(test_lemma_line)
    test_df_text_list[i] = lemma_line
```

TF –IDF VECTORIZATION

- Term frequency-inverse document frequency (TF-IDF) gives a measure that takes the importance of a word into consideration depending on how frequently it occurs in a document and a corpus.
- Term frequency (TF)
- Inverse document frequency (IDF)
- tf(t,d) = count of t in d / number of words in d
- df(t) = occurrence of t in documents

  df(t) = N(t)

  where df(t) = Document frequency of a term t

  N(t) = Number of documents containing the term t
- idf(t) = log(N/ df(t))
- tf-idf(t, d) = tf(t, d) * idf(t)

END OF EVAL-1

- This is the pre-processing that is required for the given data and we have done it efficiently.

WORD EMBEDDING – VECTORIZATION OF WORD USING TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer
word_vectorizer = TfidfVectorizer(  strip_accents = 'unicode',
                                    analyzer = 'word',
                                    stop_words = 'english',
                                    ngram_range = (1, 2),
                                    min_df = 2,
                                    max_df = 0.5)
word_vectorizer.fit(train_df_text_list+test_df_text_list)

train_word_features = word_vectorizer.transform(train_df_text_list)

test_word_features = word_vectorizer.transform(test_df_text_list)
```

WORD EMBEDDING – VECTORIZATION OF WORD USING TF-IDF

- Hyperparameters:
- strip_accents = 'unicode'

  Remove accents and perform other character normalization during the preprocessing step.

  'ascii' is a fast method that only works on characters that have an direct ASCII mapping.

  'unicode' is a slightly slower method that works on any characters.
- analyzer = 'word'

  Whether the feature should be made of word or character.
- stop_words = 'english'

  Remove all the stop words of english

WORD EMBEDDING – VECTORIZATION OF WORD USING TF-IDF

- ngram_range = (1, 2)
- ngram is the set of words together.Range (1,2) signifies that we take single words while tokenization or we take asset of twe words together.
- The lower and upper boundary of the range of n-values for different n-grams to be extracted.All values of n such that min_n <= n <= max_n will be used. For example, an ngram_range of (1,1) means only unigrams, (1,2) means unigrams and bigrams, and (2,2)means only bigrams. Only applies if the analyser is not callable.
- min_df = 2,
- max_df = 0.5

CHARACTER EMBEDDING – VECTORIZATION OF CHARACTERS USING TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer
word_vectorizer = TfidfVectorizer(  strip_accents = 'unicode',
                                    analyzer = 'word',
                                    stop_words = 'english',
                                    ngram_range = (1, 2),
                                    min_df = 2,
                                    max_df = 0.5)
word_vectorizer.fit(train_df_text_list+test_df_text_list)

train_word_features = word_vectorizer.transform(train_df_text_list)

test_word_features = word_vectorizer.transform(test_df_text_list)
```

CHARACTER EMBEDDING – VECTORIZATION OF CHARACTERS USING TF-IDF

- Hyperparameters:
- strip_accents = 'unicode'

- analyzer = 'char',then stop_words hyperparameter holds no significance and this is pretty logical.
- ngram_range = (2, 6)
- min_df = 2
- max_df = 0.5

CLASSIFICATION MODELS

- Role of pickle file
- Classification Models tried
- Analysis & results

ROLE OF PICKLE FILE

- A pickle file is used to store a trained model.
- The file format for a pickle file is **“.pckl”**.
- The trained model can be loaded from this file to make predictions without re-training.
- The use of a pickle file saves training time.
- If we use a pickle file, our code becomes more efficient and easier to debug.
- The application of pickle file is to serialize your machine learning algorithms and save the serialized format.

CLASSIFICATION MODELS TRIED

- Probabilistic models

1. Logistic Regression
2. Naive bayes

- Treebased Models

1. Random Forest

- Using cross Validation

1. Logistic regression CV

- Ensemble models

1. XG Boost

- Linear models

1. SGD Classifier
2. Ridge Classifier

CLASSIFICATION MODELS TRIED

- Here is a list of all the classification models which we tried to get the best possible accuracy.
- Logistic Regression - Fast
- Naïve bayes classifier - Fast
- Random forest classifier - Slow
- XG Boost - Slow
- SGD Classifier - Slow
- Ridge Classifier -

RESULTS

Model	Validation score	Kaggle score
Logistic Regression	0.9835	0.98039
Naïve Bayes	0.93095	0.92615
Random Forest	0.96521	0.95601
Ridge		
XG Boosting	0.96669	0.97143
SGD Classifier	0.88042	0.88358

MODEL EXPLANANTIONS

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics

model = LogisticRegression(class_weight = 'balanced',max_iter= 3500)
l = 1
for i in columns:

    x_train_lst = train_dfjoin
    y_train_lst = main_train[i].to_numpy()
    model.fit(x_train_lst, y_train_lst)
    pickle.dump(model,open(i+"logistic.pkl",'wb'))
    y_prediction = model.predict_proba(test_dfjoin)[:, 1]
    prediction_df_lst.insert(l,i,y_prediction)
    l= l + 1

prediction_df_lst.to_csv('logistic3.csv', index=None)
```

MODEL EXPLANANTIONS

- LOGISTIC REGRESSION
- class_weight = 'balanced'
- Balanced class weights can be automatically calculated within the sample weight function. Set class_weight = 'balanced' to **automatically adjust weights inversely proportional to class frequencies in the input data**
- max_iter= 3500
- Maximum number of iterations taken for the solvers to converge

MODEL EXPLANANTIONS

- NAÏVE BAYES

```
from sklearn.naive_bayes import MultinomialNB
Naive_bayes = MultinomialNB()
l=1
for i in columns:

    x_train_lst = train_dfjoin
    y_train_lst =  main_train[i].to_numpy()
    Naive_bayes.fit(x_train_lst, y_train_lst)
    pickle.dump(Naive_bayes, open(i+"naive.pkl", 'wb'))
    y_prediction = Naive_bayes.predict_proba(test_dfjoin)[:, 1]
    prediction_df_lst.insert(l,i,y_prediction)
    l=l+1
prediction_df_lst.to_csv('bayes.csv', index = None)
```

MODEL EXPLANATIONS

```
from sklearn.ensemble import RandomForestClassifier
RFC = RandomForestClassifier(max_features = 4000, max_depth = 100, min_samples_split = 10, criterion = 'gini', n_estimators = 120,
l = 1
for i in columns:

    x_train_lst = train_dfjoin
    y_train_lst =  main_train[i].to_numpy()
    RFC.fit(x_train_lst, y_train_lst)
    pickle.dump(RFC, open(i+"rfc.pkl", 'wb'))
    y_prediction = RFC.predict_proba(test_dfjoin)[:, 1]
    prediction_df_lst.insert(l,i,y_prediction)
    l=l+1
prediction_df_lst.to_csv('rfc.csv', index = None)
```

MODEL EXPLANATIONS

- RANDOM FOREST CLASSIFIER

1. max_depth = 100

   The maximum depth of the decision tree. If this value is not specified, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. We tried leaving this none in our code initially. In that case, we did not get an output even after 1000 minutes of code execution.

2. min_samples_split = 10

   The minimum number of samples is required to split an internal node of the decision trees.

3. criterion = 'gini'

   The function is to measure the quality of a split. This parameter is a tree-specific parameter.

MODEL EXPLANATIONS

4. n_estimators = 120

   The number of trees in the forest. The default value is 100, but, for better accuracy, we have increased the count by 20 trees.

5. min_weight_fraction_leaf = 0.0

   The minimum weighted fraction of the total weights (of all the input samples) is required to be at a leaf node.

6. max_leaf_nodes = None

   Grow trees with max_leaf_nodes in best-first fashion.

MODEL EXPLANATIONS

```
import xgboost as xgb

model = xgb.XGBClassifier(objective="binary:logistic", random_state=42)
l = 1
for i in columns:

    x_train_lst = train_dfjoin
    y_train_lst =  main_train[i].to_numpy()
    model.fit(x_train_lst, y_train_lst)
    pickle.dump(model, open(i+"_XGBoosting.pkl", "wb"))
    y_prediction = model.predict_proba(test_dfjoin)[:, 1]
    prediction_df_lst.insert(l,i,y_prediction)
    l = l+1

prediction_df_lst.to_csv('xgb.csv', index = None)
```

MODEL EXPLANATIONS

- XG Boost

1.XG Boost basically means,extreme gradient boost.

2.This algorithm is implementation with of gradient boost with decision trees.

3.In this algorithm, decision trees are created in sequential form. Weights are assigned to all the independent variables which are then fed to decision trees which predicts results.

4.The weights of wrongly classified variables are increased and these variables are fed into second decision tree. These individual classifiers then ensemble to give a strong and more precise model.

MODEL EXPLANATIONS

- objective="binary:logistic"

  “binary:logistic” –logistic regression for binary classification, output probability

- random_state=42

  Controls the random seed given to each Tree estimator at each boosting iteration. In addition, it controls the random permutation of the features at each split

MODEL EXPLANATION

```
from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.calibration import CalibratedClassifierCV

model = make_pipeline(StandardScaler(with_mean=False), SGDClassifier(loss='hinge',max_iter=1000, tol=1e-3))

l =1
for i in columns:

    x_train_lst = train_dfjoin
    y_train_lst =  main_train[i].to_numpy()
    model.fit(x_train_lst, y_train_lst)

    calibrator = CalibratedClassifierCV(model, cv='prefit')

    model1 = calibrator.fit(x_train_lst, y_train_lst)

    pickle.dump(model1, open(i+"_SGDC.pkl", 'wb'))
    y_prediction = model1.predict_proba(test_dfjoin)[:, 1]
    prediction_df_lst.insert(l,i,y_prediction)
    l = l+1

prediction_df_lst.to_csv('sgdc.csv', index = None)
```

MODEL EXPLANATIONS

- SGD Classifier

1. Stochastic Gradient Descent (SGD) is a simple yet efficient optimization algorithm used to find the values of parameters/coefficients of functions that minimize a cost function

2. In other words, it is used for discriminative learning of linear classifiers under convex loss functions such as SVM and Logistic regression.

MODEL EXPLANATIONS

- loss='hinge'
- ‘hinge’ gives a linear SVM.

- max_iter=1000
- The maximum number of passes over the training data (aka epochs).

- tol=1e-3
- This parameter represents the stopping criterion for iterations.

LIBRARIES USED

- NumPy
- Pandas
- NLTK
- Regex
- Pickle
- Sklearn

REFERENCES

- All classroom material like slides, lectures.
- Online video lectures from YouTube.
- Following are some websites which helped a lot.

Documentation

1. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
2. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegressionCV.html
3. https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

REFERENCES

4. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
5. https://www.kaggle.com/code/sudhirnl7/simple-naive-bayes-xgboost/notebook
6. https://scikit-learn.org/stable/modules/sgd.html

ARTICLES

1. https://stackoverflow.com/
2. http://www.geeksforgeeks.org/
3. http://medium.com/
4. https://towardsdatascience.com/
5. http://www.analyticssteps.com/

CONTRIBUTION

- Whole project was team effort and both contributed in their own way, coming up with ideas, models and code.

THANK YOU