Sentimental Analysis on IMDB Movie Reviews

Mod 9 Presentation

Max (???) Lim Zheng Wei

Intro

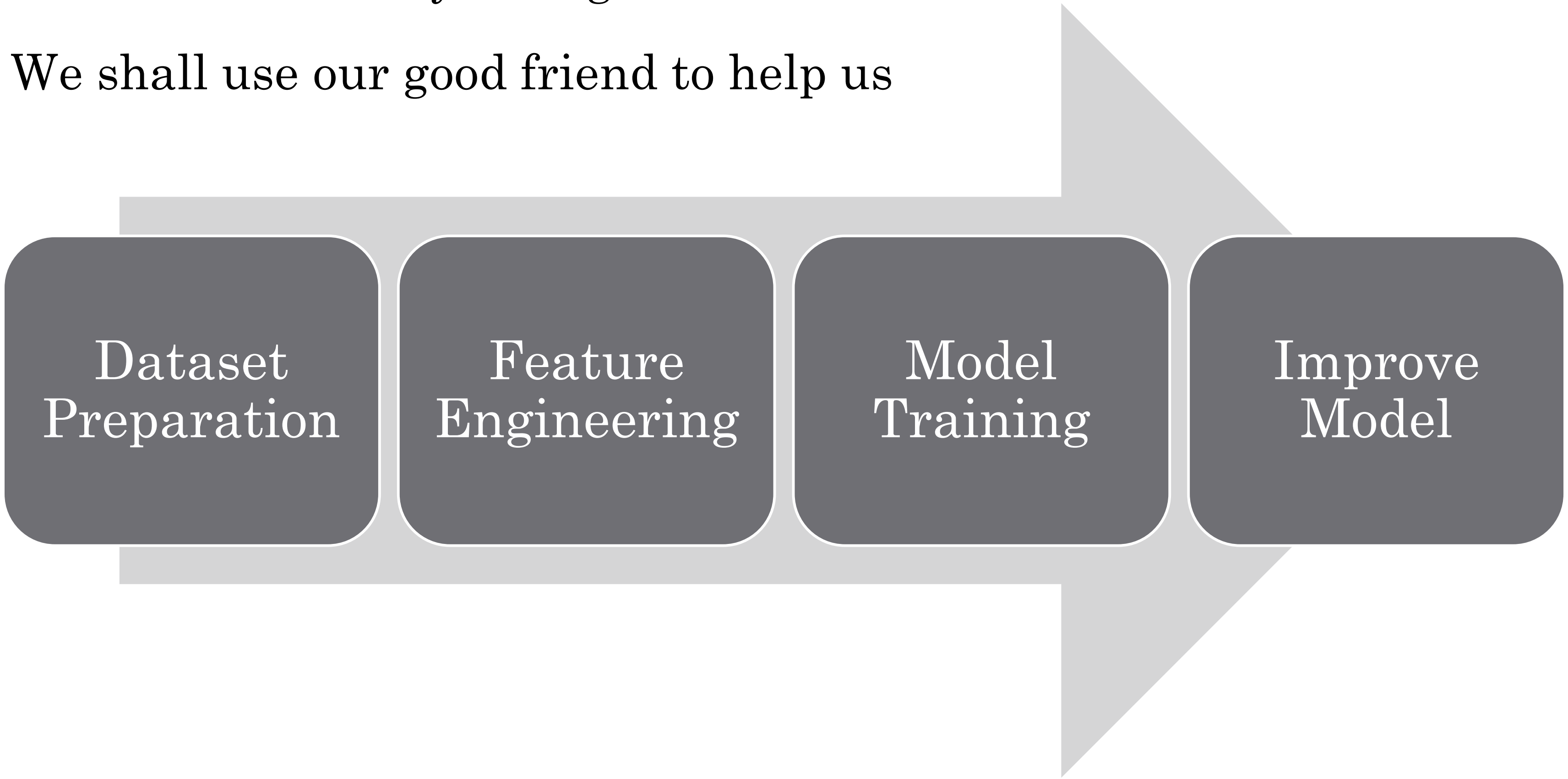
- Sentiment analysis refers to the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information

Applications include

- Reviews, survey responses
- Online and social media, and healthcare materials applications
- Customer service

Objective & Scope

- Do sentiment analysis engine on IMDB reviews
- We shall use our good friend to help us



Data Preparation

1. Import relevant libraries & data
2. Do some simple EDA
3. Clean text
4. Common word removal
5. Rare word removal
6. Word Cloud
7. Modelling
8. Set feature labels

Import Data

```
train_csv = 'C:/Users/zheng/Desktop/Data Science/Presentations/Mod 9//Train.csv'
test_csv = 'C:/Users/zheng/Desktop/Data Science/Presentations/Mod 9//Test.csv'
valid_csv = 'C:/Users/zheng/Desktop/Data Science/Presentations/Mod 9//Valid.csv'

train = pd.read_csv(train_csv)
test = pd.read_csv(test_csv)
valid = pd.read_csv(valid_csv)
```

```
train.head()
```

	text	label
0	I grew up (b. 1965) watching and loving the Th...	0
1	When I put this movie in my DVD player, and sa...	0
2	Why do people who do not know what a particula...	0
3	Even though I have great interest in Biblical ...	0
4	Im a die hard Dads Army fan and nothing will e...	1

CSI Time…

- 40000 entries
- 0 null value
- Positive (1) or negative (0) value pretty equally distributed
- Too many examples for our computation capacity
- Split the dataset and reduce it.

```
train.label.unique()
```

```
array([0, 1], dtype=int64)
```

```
train.label.value_counts()
```

```
0    20019
1    19981
Name: label, dtype: int64
```

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40000 entries, 0 to 39999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   text    40000 non-null  object
 1   label   40000 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 625.1+ KB
```

CNY Cleaning

```
import nltk
from nltk.stem import WordNetLemmatizer, LancasterStemmer
from nltk.corpus import stopwords
from nltk import word_tokenize, sent_tokenize

from string import punctuation
```

```
stopword = nltk.corpus.stopwords.words('english')
stopword.remove('not')
```

```
def clean(text):
    wn = nltk.WordNetLemmatizer()
    stopword = nltk.corpus.stopwords.words('english')
    tokens = nltk.word_tokenize(text)                                          #tokenize
    lower = [word.lower() for word in tokens]                                  #lowercase
    no_stopwords = [word for word in lower if word not in stopword]            #remove stopwords
    no_alpha = [word for word in no_stopwords if word.isalpha()]               #remove numbers
    lemm_text = [wn.lemmatize(word) for word in no_alpha]                      #lemmatize
    clean_text = lemm_text
    return clean_text
```

```
train['clean']=train['text'].apply(clean)
train['clean']=train['clean'].apply(lambda x: " ".join([str(word) for word in x]))
```

Common Word Removal

- Pre… in
  a ge…
- Car… xt
  dat…
- Che… xt
  dat…

```
freq = pd.Series(' '.join(train['clean']).split()).value_counts()[:10]
freq
```

```
br           161465
movie         80000
film          72171
one           42900
like          32243
time          23745
good          23198
character     22272
would         21182
even          19881
dtype: int64
```

```
freq = list(freq.index)
```

```
freq.remove('like')
```

```
freq.remove('good')
```

```
freq
```

```
['br', 'movie', 'film', 'one', 'time', 'character', 'would', 'even']
```

```python
train['clean'] = train['clean'].apply(lambda x: " ".join(x for x in x.split() if x not in freq))
train['clean'].sample(20)
```

```
10292    best rainer werner fassbinder made successful ...
30843    quite brutal huge implausibility silly script ...
32239    dungeon harrow lot thing could made quite good...
11670    without shadow doubt absolute worst steven sea...
6307     maybe title trailer certainly interview dvd di...
26117    really liked get shorty completely disappointi...
24419    liked lead relied heavily charming smile care ...
4586     respect mike hodges liked get carter immensely...
27466    omg reason giving instead tom hank funny apart...
28809    saw kid yanked rotation left bad taste mouth c...
10395    tattooed stranger another rare screened year s...
4960     art student rome possessed something dream nai...
36811    subject child terminally ill difficult saddeni...
24323    yes reviewer already stated may vintage l h fa...
15246    careful tell watching long could become easy p...
2568     mad dog earle back along moll marie fickle clu...
11742    think testosterone instead estrogen get genera...
1772     really special beautiful start three orphan sh...
34214    contains fact widely reported exactly truth to...
27264    batman return tim burton succumbed important p...
Name: clean, dtype: object
```

Rare Word Removal

- Ren
- Bec n
  and
- Car l
  the

```
freq1 = pd.Series(' '.join(train['clean']).split()).value_counts()[-7196:]
freq1
```

```
adante          1
anointing       1
notation        1
evinces         1
brasileira      1
               ..
stinkbombs      1
vajna           1
stygian         1
stensvold       1
flaxen          1
Length: 7196, dtype: int64
```

```
freq1 = list(freq1.index)
```

```
%%time
train['clean'] = train['clean'].apply(lambda x: " ".join(x for x in x.split() if x not in freq1))
train['clean'].sample(20)
```

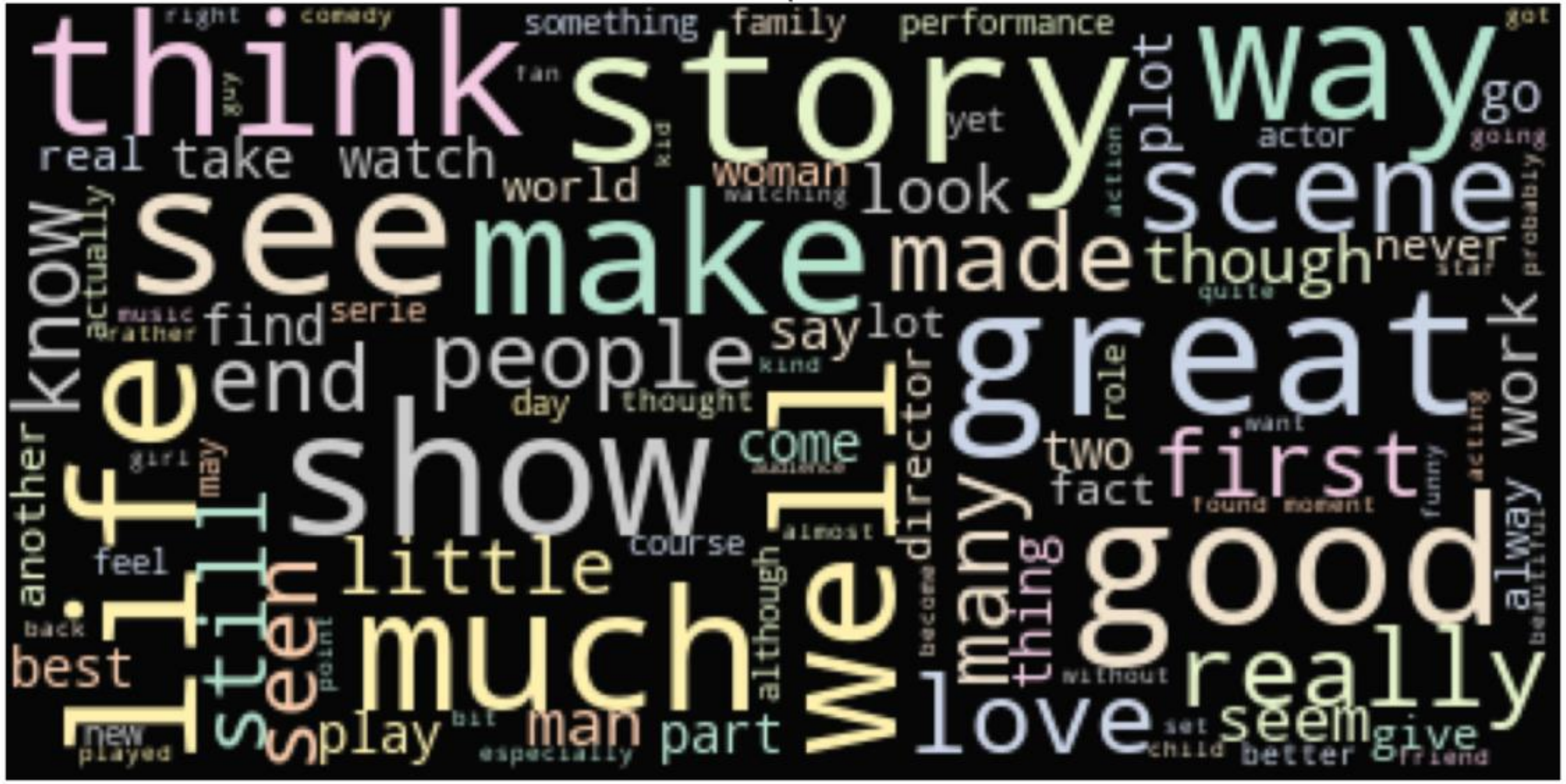
```
Wall time: 22min 28s

20832    director edward sedgwick old hand visual comed...
7204     homeward bound beautiful part shadow fall ditc...
25802    seem made ready watched made thought watching ...
8962     government elected three year term reg said li...
3817     young erendira tyrranical grandmother provide ...
557      grading curve word greatest ever made exactly ...
11049    saw came year old classic rock still never lik...
16318    found west point agreeable although doubt watc...
39022    supernatural vengeful police officer back thir...
16584    te cartoon instead country cousin visually muc...
2323     start offering like nearly said going step far...
38643    shintarô katsu gained ton fame playing wonderf...
15203    happy coincidence year jimmy stewart kim novak...
10809    kick as powerful acting story push u live drea...
39587    waited come canada finally excited see really ...
3704     prone ranting expectation low start seem like ...
26683    question command training cadet major chick da...
9596     refreshing change pace mindless hong kong tria...
21570    director know camera many option always always...
20502    deserved dead redefines term bad bad stranger ...
Name: clean, dtype: object
```

Word Cloud

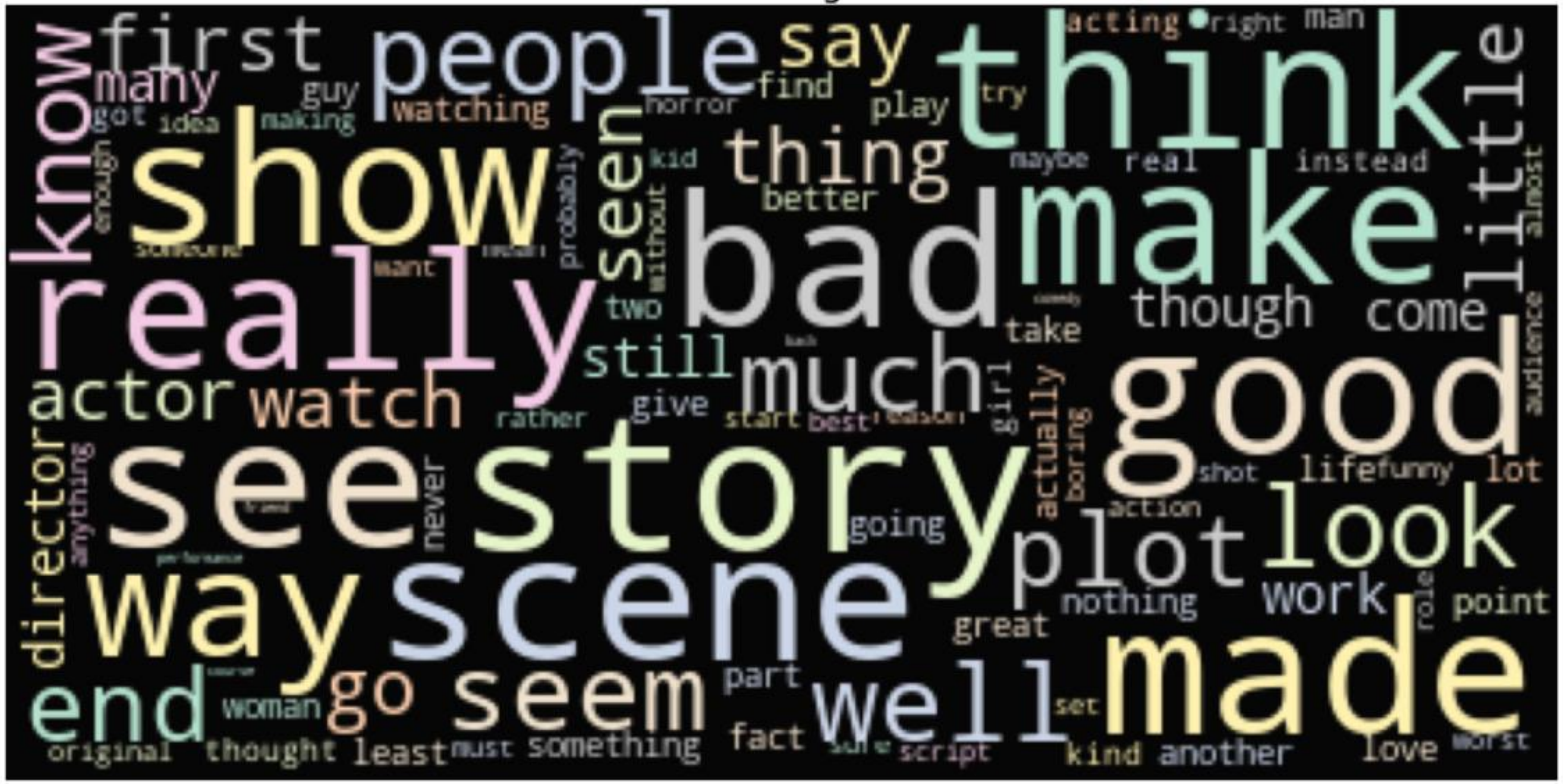
```
%%time

from wordcloud import WordCloud
from collections import Counter

def generate_wordcloud(words,sentiment):
    plt.figure(figsize=(16,13))
    wc = WordCloud(background_color="black", max_words=100, max_font_size=50)
    wc.generate(words)
    plt.title("Most common {} words".format(sentiment), fontsize=20)
    plt.imshow(wc.recolor(colormap='Pastel2', random_state=17), alpha=0.98)
    plt.axis('off')


print("Creating word clouds...")
positive_words=" ".join(train[train.label==1]['clean'].values)
negative_words=" ".join(train[train.label==0]['clean'].values)

generate_wordcloud(positive_words,"positive")
generate_wordcloud(negative_words,"negative")
```

```
Creating word clouds...
Wall time: 1min
```

Most common positive words



Most common negative words



Modelling

```python
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import average_precision_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
```

```python
# helper function to show results and charts
def show_summary_report(actual, prediction, predict_proba):

    if isinstance(actual, pd.Series):
        actual = actual.values
    if actual.dtype.name == 'object':
        actual = actual.astype(int)
    if prediction.dtype.name == 'object':
        prediction = prediction.astype(int)

    accuracy_ = accuracy_score(actual, prediction)
    precision_ = precision_score(actual, prediction)
    recall_ = recall_score(actual, prediction)
    roc_auc_ = roc_auc_score(actual, predict_proba)

    print('Accuracy : %.4f [TP / N] Proportion of predicted labels that match the true labels. Best: 1, Worst: 0' % accuracy_)
    print('Precision: %.4f [TP / (TP + FP)] Not to label a negative sample as positive.        Best: 1, Worst: 0' % precision_)
    print('Recall   : %.4f [TP / (TP + FN)] Find all the positive samples.                     Best: 1, Worst: 0' % recall_)
    print('ROC AUC  : %.4f                                                                     Best: 1, Worst: < 0.5' % roc_auc_)
    print('-' * 107)
    print('TP: True Positives, FP: False Positives, TN: True Negatives, FN: False Negatives, N: Number of samples')

    # Confusion Matrix
    mat = confusion_matrix(actual, prediction)

    # Precision/Recall
    precision, recall, _ = precision_recall_curve(actual, prediction)
    average_precision = average_precision_score(actual, prediction)

    # Compute ROC curve and ROC area
    fpr, tpr, _ = roc_curve(actual, predict_proba)
    roc_auc = auc(fpr, tpr)
```

```python
    # plot
    fig, ax = plt.subplots(1, 3, figsize = (18, 6))
    fig.subplots_adjust(left = 0.02, right = 0.98, wspace = 0.2)

    # Confusion Matrix
    sns.heatmap(mat.T, square = True, annot = True, fmt = 'd', cbar = False, cmap = 'Blues', ax = ax[0])

    ax[0].set_title('Confusion Matrix')
    ax[0].set_xlabel('True label')
    ax[0].set_ylabel('Predicted label')

    # Precision/Recall
    step_kwargs = {'step': 'post'}
    ax[1].step(recall, precision, color = 'b', alpha = 0.2, where = 'post')
    ax[1].fill_between(recall, precision, alpha = 0.2, color = 'b', **step_kwargs)
    ax[1].set_ylim([0.0, 1.0])
    ax[1].set_xlim([0.0, 1.0])
    ax[1].set_xlabel('Recall')
    ax[1].set_ylabel('Precision')
    ax[1].set_title('2-class Precision-Recall curve')

    # ROC
    ax[2].plot(fpr, tpr, color = 'darkorange', lw = 2, label = 'ROC curve (AUC = %0.2f)' % roc_auc)
    ax[2].plot([0, 1], [0, 1], color = 'navy', lw = 2, linestyle = '--')
    ax[2].set_xlim([0.0, 1.0])
    ax[2].set_ylim([0.0, 1.0])
    ax[2].set_xlabel('False Positive Rate')
    ax[2].set_ylabel('True Positive Rate')
    ax[2].set_title('Receiver Operating Characteristic')
    ax[2].legend(loc = 'lower right')

    plt.show()

    return (accuracy_, precision_, recall_, roc_auc_)
```

Set Feature Labels

```
# Features and Labels
X = train['clean']
y = train['label']

# split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.9, random_state = 42)
X_train, X_test, y_train, y_test = train_test_split(X_train,y_train,test_size = 0.2, random_state=42)

print("Data distribution:\n- Train: {} \n- Test: {}".format(len(y_train),len(y_test)))
```

```
Data distribution:
- Train: 3200
- Test: 800
```

2. Feature Engineering

A. CountVectorizer()

B. TF-IDF Vectorizer()

- Word Level
- N-Gram Level
- Character Level

C. Topic Model

```python
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
```

2A CountVectorizer

```
%%time

# create a matrix of word counts from the text
count_vect = CountVectorizer(token_pattern = r'\w{1,}')
```

```
Wall time: 0 ns
```

```
%%time

# do the actual counting
A = count_vect.fit_transform(X_train, y_train)
```

```
Wall time: 582 ms
```

```
%%time


# Transform documents to document-term matrix.
X_train_count = count_vect.transform(X_train)
X_test_count = count_vect.transform(X_test)
```

```
Wall time: 660 ms
```

2B.1 TF-IDF Vectorizer

```
%%time
# word level tf-idf
tfidf_vect = TfidfVectorizer(analyzer = 'word',
                             token_pattern = r'\w{1,}',
                             max_features = 5000)
print(tfidf_vect)

B1 = tfidf_vect.fit(X_train, y_train)
X_train_tfidf = tfidf_vect.transform(X_train)
X_test_tfidf  = tfidf_vect.transform(X_test)
```

```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.float64'>, encoding='utf-8',
                input='content', lowercase=True, max_df=1.0, max_features=5000,
                min_df=1, ngram_range=(1, 1), norm='l2', preprocessor=None,
                smooth_idf=True, stop_words=None, strip_accents=None,
                sublinear_tf=False, token_pattern='\\w{1,}', tokenizer=None,
                use_idf=True, vocabulary=None)
Wall time: 1.24 s
```

2B.2 TF-IDF Vectorizer N-Gram

```
%%time
# ngram level tf-idf
tfidf_vect_ngram = TfidfVectorizer(analyzer = 'word',
                                   token_pattern = r'\w{1,}',
                                   ngram_range = (2, 3),
                                   max_features = 5000)
print(tfidf_vect_ngram)

B2 = tfidf_vect_ngram.fit(X_train, y_train)
X_train_tfidf_ngram = tfidf_vect_ngram.transform(X_train)
X_test_tfidf_ngram  = tfidf_vect_ngram.transform(X_test)
```

```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.float64'>, encoding='utf-8',
                input='content', lowercase=True, max_df=1.0, max_features=5000,
                min_df=1, ngram_range=(2, 3), norm='l2', preprocessor=None,
                smooth_idf=True, stop_words=None, strip_accents=None,
                sublinear_tf=False, token_pattern='\\w{1,}', tokenizer=None,
                use_idf=True, vocabulary=None)
Wall time: 5.34 s
```

2B.3 TF-IDF Vectorizer N-Gram Char

```
%%time
# characters level tf-idf
tfidf_vect_ngram_chars = TfidfVectorizer(analyzer = 'char',
                                         token_pattern = r'\w{1,}',
                                         ngram_range = (2, 3),
                                         max_features = 5000)
print(tfidf_vect_ngram_chars)

B3 = tfidf_vect_ngram_chars.fit(X_train, y_train)
X_train_tfidf_ngram_chars = tfidf_vect_ngram_chars.transform(X_train)
X_test_tfidf_ngram_chars  = tfidf_vect_ngram_chars.transform(X_test)
```

```
TfidfVectorizer(analyzer='char', binary=False, decode_error='strict',
                dtype=<class 'numpy.float64'>, encoding='utf-8',
                input='content', lowercase=True, max_df=1.0, max_features=5000,
                min_df=1, ngram_range=(2, 3), norm='l2', preprocessor=None,
                smooth_idf=True, stop_words=None, strip_accents=None,
                sublinear_tf=False, token_pattern='\\w{1,}', tokenizer=None,
                use_idf=True, vocabulary=None)
Wall time: 9.66 s
```

Latent Dirichlet Allocation(LDA)

- A recurring theme in NLP is to understand large corpus of texts through topics extraction
- Understanding key topics will always come in handy
- Unsupervised machine-learning model that takes documents as input and finds topics as output
- Also shows in what percentage each document talks about each topic

There are 3 main parameters of the model:

- the number of topics
- the number of words per topic
- the number of topics per document

2C

Number of topics

```
%%time
# train a LDA Model
lda_model = LatentDirichletAllocation(n_components = 20, learning_method = 'online', max_iter = 20)

X_topics = lda_model.fit_transform(X_train_count)
topic_word = lda_model.components_
vocab = count_vect.get_feature_names()
```

```
Wall time: 56.5 s
```

Number of words per topic

```
# view the topic models
n_top_words = 10
topic_summaries = []
print('Group Top Words')
print('-----', '-'*80)
for i, topic_dist in enumerate(topic_word):
    topic_words = np.array(vocab)[np.argsort(topic_dist)][:-(n_top_words+1):-1]
    top_words = ' '.join(topic_words)
    topic_summaries.append(top_words)
    print('  %3d %s' % (i, top_words))
```

```
Group Top Words
----- ---------------------------------------------------------------------------
    0 matthau greek logan principle chorus rusty ramgopal pierre fraternity kersey
    1 song streisand ant singing musical tourist red sing broadway ahmad
    2 match demon hart hogan naschy v wwe royal ogre rumble
    3 war life world american love novel young human soldier man
    4 seagal bank barney arthur cagney steven robin sidekick shark thompson
    5 indian western columbo john peter de wayne ford house custer
    6 hitler bruno gu agonizing raccoon irritating downfall ninja hudson count
    7 melville rififi delon heist maradona varma blunt cercle almasy explaining
    8 poirot prue phoebe ustinov explosion book suchet kells gypo leila
    9 resident suzanne spaghetti machine robinson sammo lil meeker kar loudly
   10 edmund colonel olivia lifeforce norwegian hooper abigail patrick overacting cliche
   11 rackham agatha h nun recommendation l costello ski akshay flavia
   12 herzog patty doyle bush arab bukowski theo thornway rourke perm
   13 holmes bourne karloff bergman ultimatum tingle damon supremacy chase rebecca
   14 andrew madonna aweigh racing wa shahrukh wai lundgren spinning lau
   15 jack dawson frost marine snowman underdog camp angela popularity island
   16 hitchcock alfred jeremy lane helsing rambo cassavetes addict tarzan darren
   17 jesse nolte sybok shatner havers trek kirk cobra enterprise brando
   18 like good get story really see make scene well could
   19 corny spike show lee network jordan fox season wanda oppenheimer
```

3. Text Classification

1. Naïve Bayes
2. Linear Classifier
3. Support Vector Machine
4. Bagging Models
5. Boosting Models

3A.1 Naïve Bayes

```
%%time
# define model

model_3_A = MultinomialNB()

# fit the training dataset on the classifier

A1 =  model_3_A.fit(X_train_count, y_train)

# predict the labels on validation dataset

predictions_A1 = model_3_A.predict(X_test_count)
predict_proba_A1 = model_3_A.predict_proba(X_test_count)[:,1]
```

```
Wall time: 16 ms
```

3A.2,3,4

```
%%time
A2 = model_3_A.fit(X_train_tfidf, y_train)

predictions_A2 = model_3_A.predict(X_test_tfidf)
predict_proba_A2 = model_3_A.predict_proba(X_test_tfidf)[:,1]
```

```
Wall time: 8.01 ms
```

```
%%time
A3 = model_3_A.fit(X_train_tfidf_ngram, y_train)

predictions_A3 = model_3_A.predict(X_test_tfidf_ngram)
predict_proba_A3 = model_3_A.predict_proba(X_test_tfidf_ngram)[:,1]
```

```
Wall time: 0 ns
```

```
%%time
A4 = model_3_A.fit(X_train_tfidf_ngram_chars, y_train)

predictions_A4 = model_3_A.predict(X_test_tfidf_ngram_chars)
predict_proba_A4 = model_3_A.predict_proba(X_test_tfidf_ngram_chars)[:,1]
```

```
Wall time: 32 ms
```

3B Linear Classifier

```
%%time
model_3_B = LogisticRegression(solver = 'lbfgs', max_iter = 100)


# fit the training dataset on the classifier

B1 =  model_3_B.fit(X_train_count, y_train)

# predict the labels on validation dataset

predictions_B1 = model_3_B.predict(X_test_count)
predict_proba_B1 = model_3_B.predict_proba(X_test_count)[:,1]
```

```
Wall time: 571 ms
```

3C Support Vector Machine

```
%%time
# define model

model_3_C = SVC(kernel='linear', probability=True)

# fit the training dataset on the classifier

C1 =  model_3_C.fit(X_train_count, y_train)

# predict the labels on validation dataset

predictions_C1 = model_3_C.predict(X_test_count)
predict_proba_C1 = model_3_C.predict_proba(X_test_count)[:,1]
```

```
Wall time: 1min 13s
```

3D Bagging Model

Goal: Reduce the variance of a decision tree classifier

Objective: Create several subsets of data from training sample chosen randomly with replacement. Each collection of subset data is used to train their decision trees. As a result, we get an ensemble of different models. Average of all the predictions from different trees are used which is more robust than a single decision tree classifier.

Analogy: Sort out items in a house

Partitioning of data	Random
Goal to achieve	Minimum variance
Methods used	Random subspace
Functions to combine single model	Weighted average
Example	Random Forest

Advantages:

- Reduces over-fitting of the model.
- Handles higher dimensionality data very well.
- Maintains accuracy for missing data.

Disadvantages:

- Since final prediction is based on the mean predictions from subset trees, it won’t give precise values for the classification and regression model.

3D

```
%%time
# define model

model_3_D = RandomForestClassifier(n_estimators = 100)

# fit the training dataset on the classifier

D1 =  model_3_D.fit(X_train_count, y_train)

# predict the labels on validation dataset

predictions_D1 = model_3_D.predict(X_test_count)
predict_proba_D1 = model_3_D.predict_proba(X_test_count)[:,1]
```

```
Wall time: 5.03 s
```

3E Boosting Model

- Create a collection of predictors
- Learners are learned sequentially with early learners fitting simple models to the data and then analysing data for errors. Consecutive trees (random sample) are fit and at every step
- Goal : Improve the accuracy from the prior tree. When an input is misclassified by a hypothesis, its weight is increased so that next hypothesis is more likely to classify it correctly. This process converts weak learners into better performing model.

- Analogy: Diagnostics Test, Streaming Test

Partitioning of data	Higher vote to misclassified samples
Goal to achieve	Increase accuracy
Methods used	Gradient descent
Functions to combine single model	Weighted majority vote
Example	Ada Boost

Advantages:

- Supports different loss function (we have used ‘binary:logistic’ for this example).
- Works well with interactions.

Disadvantages:

- Prone to over-fitting.
- Requires careful tuning of different hyper-parameters.

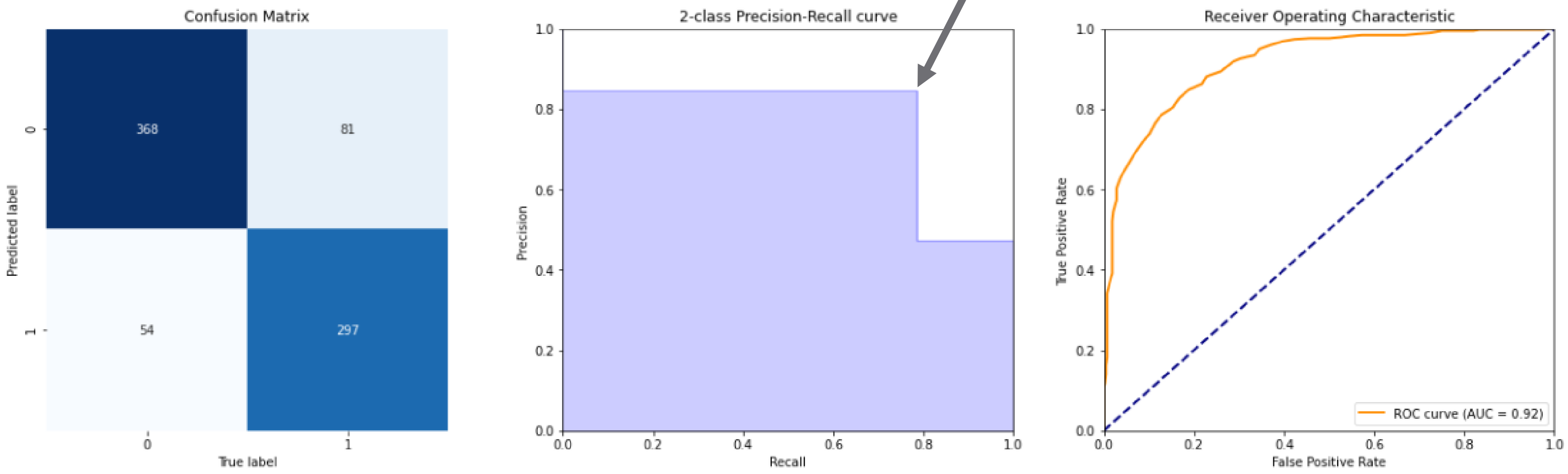
```
%%time
# define model

model_3_E = GradientBoostingClassifier()

# fit the training dataset on the classifier

E1 =  model_3_E.fit(X_train_count, y_train)

# predict the labels on validation dataset

predictions_E1 = model_3_E.predict(X_test_count)
predict_proba_E1 = model_3_E.predict_proba(X_test_count)[:,1]
```

```
Wall time: 8.1 s
```

4. Summary Report

```
show_summary_report(y_test, predictions_D2, predict_proba_D2)
```

```
Accuracy : 0.8313 [TP / N] Proportion of predicted labels that match the true labels. Best: 1, Worst: 0
Precision: 0.8462 [TP / (TP + FP)] Not to label a negative sample as positive.        Best: 1, Worst: 0
Recall   : 0.7857 [TP / (TP + FN)] Find all the positive samples.                     Best: 1, Worst: 0
ROC AUC  : 0.9160                                                                     Best: 1, Worst: < 0.5
-----------------------------------------------------------------------------------------------------------
TP: True Positives, FP: False Positives, TN: True Negatives, FN: False Negatives, N: Number of samples
```




```
(0.83125, 0.8461538461538461, 0.7857142857142857, 0.916039770305173)
```

	NB Count Vectorizer	NB TF-IDF Vectorizer	NB TF-IDF N-Gram	NB TF-IDF Char	LC Count Vectorizer	LC TF-IDF Vectorizer	LC TF-IDF N-Gram	LC TF-IDF Char
Accuracy	0.837500	0.842500	0.758750	0.797500	0.840000	0.845000	0.760000	0.813750
Precision	0.854286	0.851955	0.745358	0.805085	0.820513	0.829016	0.733668	0.795866
Recall	0.791005	0.806878	0.743386	0.753968	0.846561	0.846561	0.772487	0.814815
ROC_AUC	0.894957	0.918754	0.836781	0.883943	0.911075	0.927029	0.827779	0.889685

GB Count Vectorizer	GB TF-IDF Vectorizer	GB TF-IDF N-Gram	GB TF-IDF Char
0.805000	0.783750	0.655000	0.775000
0.772059	0.746988	0.593750	0.752551
0.833333	0.820106	0.854497	0.780423
0.892722	0.879438	0.738697	0.872897

SVC Count Vectorizer	SVC TF-IDF Vectorizer	SVC TF-IDF N-Gram	SVC TF-IDF Char	RF Count Vectorizer	RF TF-IDF Vectorizer	RF TF-IDF N-Gram	RF TF-IDF Char
0.822500	0.838750	0.732500	0.813750	0.847500	0.822500	0.708750	0.791250
0.796482	0.820051	0.699029	0.786967	0.835079	0.820652	0.679012	0.804035
0.838624	0.843915	0.761905	0.830688	0.843915	0.798942	0.727513	0.738095
0.897731	0.922180	0.804408	0.893729	0.925685	0.913451	0.788645	0.865286

Conclusion

- Every model is pretty solid
- Pretty accurate
- Best model is LC TF-IDF Vectorizer

References

- https://en.wikipedia.org/wiki/Sentiment_analysis
- https://www.kdnuggets.com/2019/09/overview-topics-extraction-python-latent-dirichlet-allocation.html
- https://analyticsindiamag.com/primer-ensemble-learning-bagging-boosting/#:~:text=Bagging%20is%20a%20way%20to,based%20on%20the%20last%20classification.