A multimodel appoach on text classification tasks

By:

Nikhil Ankolkar, Mikkel Peter Lemming, Zhan Su, Ioannis Kritikos and Mikkel Tonnhøj Petersen

Outline

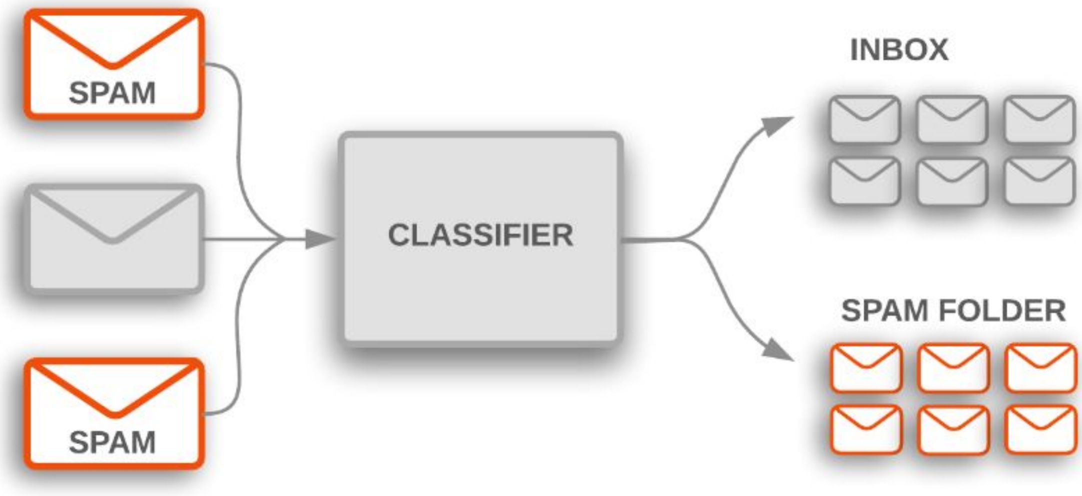
- Motivation
- Text Classification Task
- Feature Extraction for Text
- Models and Results
- Conclusions and Future work

Motivation

- Is traditional machine learning good enough for our datasets?
- Why deep learning model can get better performance?
- What is the difference of features in traditional ML and NN.

02 - Text Classification Task

Text classification is a common NLP task that assigns a label or class to text.



Dataset

SST2

The Stanford Sentiment Treebank consists of sentences from movie reviews and human annotations of their sentiment. The task is to predict the sentiment of a given sentence. It uses the two-way (positive/negative) class split, with only sentence-level labels.

Dataset Preview

Subset: sst2

Split: train

sentence (string)	label (class label)	idx (int)
hide new secretions from the parental units	0 (negative)	0
contains no wit , only labored gags	0 (negative)	1
that loves its characters and communicates something rather beautiful about human nature	1 (positive)	2

Dataset

MRPC

The Microsoft Research Paraphrase Corpus (Dolan & Brockett, 2005) is a corpus of sentence pairs automatically extracted from online news sources, with human annotations for whether the sentences in the pair are semantically equivalent.

Dataset Preview

Subset: mrpc

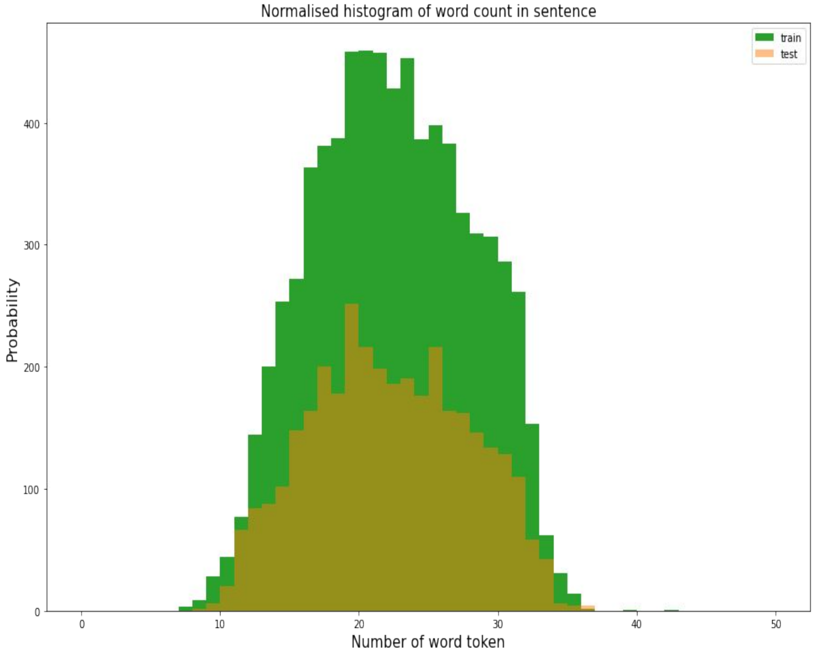
Split: train

sentence1 (string)	sentence2 (string)	label (class label)	idx (int)
Amrozi accused his brother , whom he called " the witness " , of deliberately distorting his evidence .	Referring to him as only " the witness " , Amrozi accused his brother of deliberately distorting his evidence .	1 (equivalent)	0
Yucaipa owned Dominick 's before selling the chain to Safeway in 1998 for $ 2.5 billion .	Yucaipa bought Dominick 's in 1995 for $ 693 million and sold it to Safeway for $ 1.8 billion in 1998 .	0 (not_equivalent)	1
They had published an advertisement on the Internet on June 10 , offering the cargo for sale , he added .	On June 10 , the ship 's owners had published an advertisement on the Internet , offering the explosives for sale .	1 (equivalent)	2
Around 0335 GMT , Tab shares were up 19 cents , or 4.4 % , at A $ 4.56 , having earlier set a record high of A $ 4.57 .	Tab shares jumped 20 cents , or 4.6 % , to set a record closing high at A $ 4.57 .	0 (not_equivalent)	3

Statistics of Dataset

Statistics of MRPC and SST2

	train pairs	dev pairs	test pairs
MRPC	3668	408	1725
SST2	67349	872	1821



Feature Extraction from Text

For words to be processed by machine learning models, we need some form of numeric representation that models can use in their calculation.

TF-IDF

Word2vec(Word embedding)

TF-IDF

Not all words have the same impact on the meaning of a document, and should thus not be treated as so. Each word contributes a TF-IDF value, based on its frequency in the document and its presence in other documents. TF-IDF thus aims to generate the meaning of a sentence, by primarily looking for unique words.

$$w_{t,d} = \begin{cases} (1 + \log_2 f_{t,d}) \times \log_2 \frac{N}{N_t} & \text{if } f_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

TF-IDF Weighting Scheme

N = total number of documents in the collection

N_t = number of documents containing t

Word2Vec

Word embedding creates a vector containing information about its similarity to other words, in their vectors similarities. The vector is affected by the words surrounding the target word, this gives the word some context, for witch it can compare itself for other words. Words often surrounded by the same words have the same context, and will in general have some inherent similarity.

```
model.wv.most_similar('dollar')
```

```
[('swiss', 0.4707739651203155),
 ('euro', 0.461805731058l207),
 ('franc', 0.453893035650253 3),
 ('currency', 0.4377799034118652 3),
 ('1.2998', 0.41703617572784424),
 ('1.2980', 0.415885984897613 5),
 ('capital', 0.41353121399879456),
 ('steeply', 0.41195234656333923),
 ('centime', 0.40646597743034363),
 ('greenback', 0.4034469723701477)]
```

```
million :
[('billion', 0.962777316570282), ('cents', 0.9595215916633606), ('$', 0.9588825702667236), ('points', 0.954538106918335), ('u
p', 0.9540597200393677)]
```

```
million :
[('hopped-up', 0.9714972376823425), ('bon', 0.9709244966506958), ('jersey', 0.9704394936561584), ('richard', 0.97033214569091
8), ('program', 0.9691138863563538)]
```

03: Our models

Tree based solutions:

XGBoost

Sklearn Random forest

LightGBM

1. Good “out-of-the-box” models
2. Easy to train and HP optimize
3. These models are not made for this task

03: Our models

Tree based solutions:

XGBoost

Sklearn Randomforest

LightGBM

1. Using standard “tfid” positive / negative words are not weighted over other kinds of words

2. Thus acc(SST2) < acc(MRPC)

3. This is also what we found!

4. However the models using SST2 data were only slightly better than a random guesser

Traditional Supervised Learning models

Naive Bayes

SVM-SVC

SGDClassifier

Naive Bayes

- Bayes' Theorem
- Why Naive?

Assumptions: each feature/variable of the same class makes an independent, equal contribution to the outcome.
These assumptions are not in general true in real-world situations

Naive Bayes

- How does the model deal with unseen data?

alpha parameter - smoothing

Overall:

1. One of the simplest and fastest classification algorithms
2. can be used for large datasets
3. Requires a small amount of training data to learn the parameters

SVM-SVC

SVM-SVC

Each observation is plotted as a point in an n-dimensional space, n is the number of features in the dataset

Task: find the optimal hyperplane that successfully classifies the data points into their respective classes

Overall:

It is a good classifier, BUT:

1. slow
2. works on small datasets, impractical for large datasets

SGDClassifier

SGDClassifier

It applies linear classifiers (SVM, logistic regression, etc.) with SGD training.
by default: linear support vector machine (SVM)

Almost the same result compared to SVM but in less than 1s

	Accuracy	Training time
SGDClassifier	0.8291	0.6275
SVC	0.8348	810.67

Overall:

1. A good and extremely fast classifier
2. recommended for large datasets

Deep Learning solutions:

CNN

1. Many ways to optimize a CNN model and detects important features well ✓
2. High potential for other NLP tasks such as Sentiment Analysis, Spam Detection or Topic Categorization. ✓
3. Long training time and more difficult to implement ✗

CNN with Word2Vec Embedding

1. Implemented through Keras
2. Converted texts to Sequences
3. Continuous bag of words method to create the Word2Vec embedding layer trained on both datasets
4. Hyperparameter tuning with Keras Tuner
5. Accuracy (SST2) < Accuracy (MRPC)

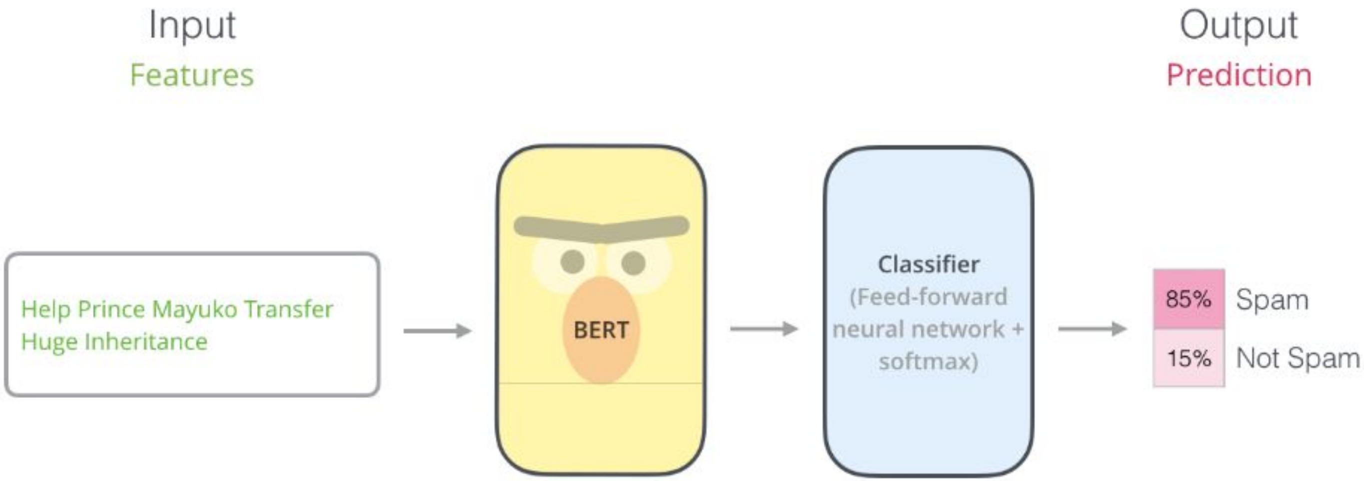
Deep Learning solutions:

RNN

1. Similar to the CNN model
2. The aim was to use LSTM layers from Tensorflow
3. However failed to improve the validation set to more than a random guess model (acc = 0.5)
4. This would be a prime example of future work.

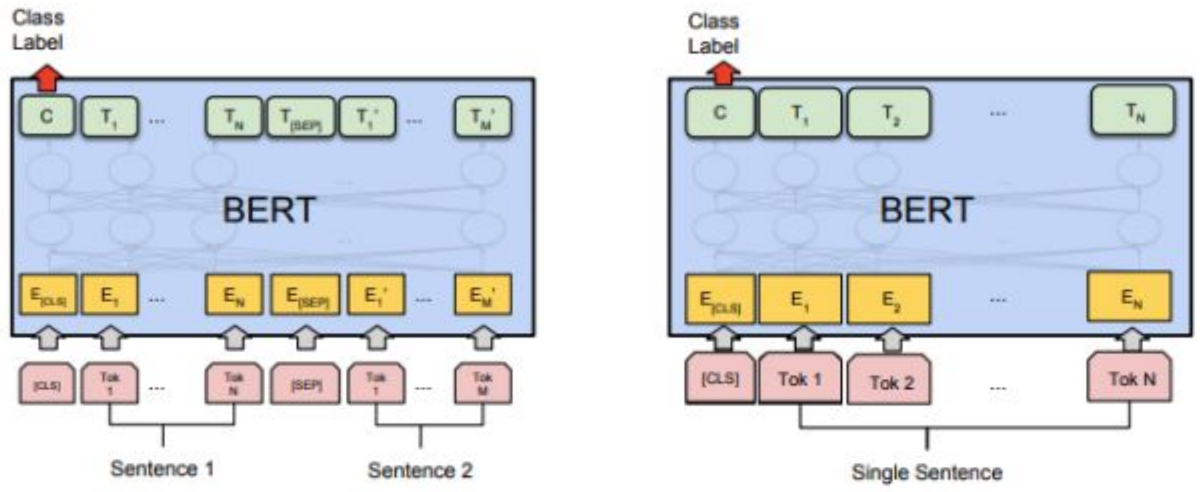
BERT

Using BERT to classify single piece of text



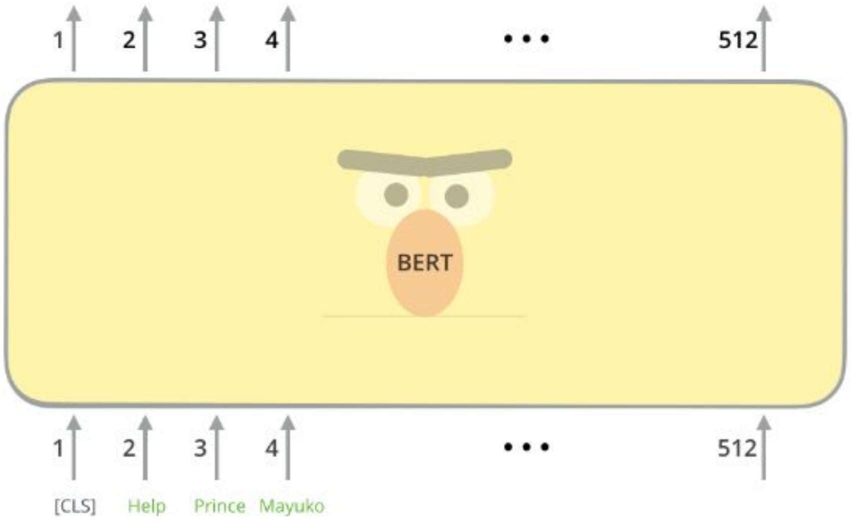
Finetuning BERT

BERT is a pre-trained language model. To use BERT to classification task, we need to finetune the BERT to our datasets (MRPC and SST2).



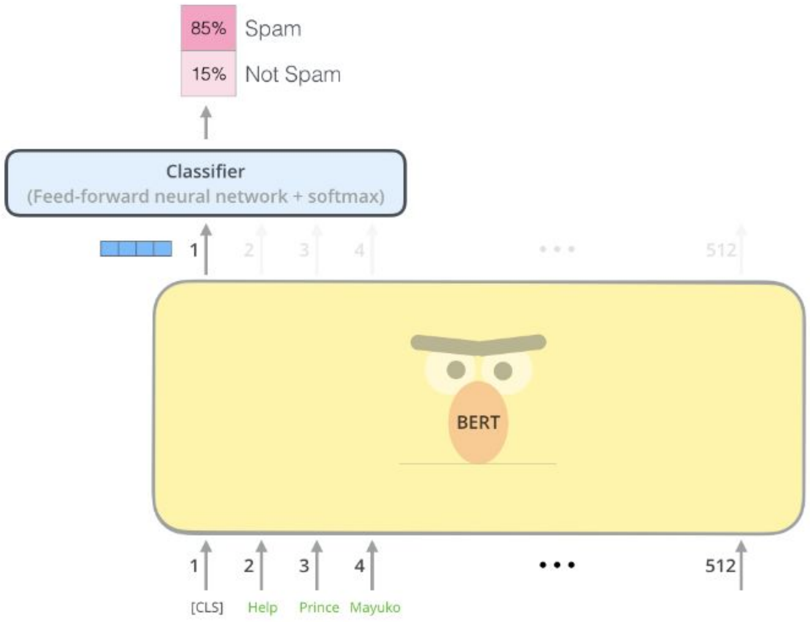
Model Architecture

The first input token is supplied with a special [CLS] token for reasons that will become apparent later on. CLS here stands for Classification.



Model Architecture

For the sentence classification example we've looked at above, we focus on the output of only the first position (that we passed the special [CLS] token to)

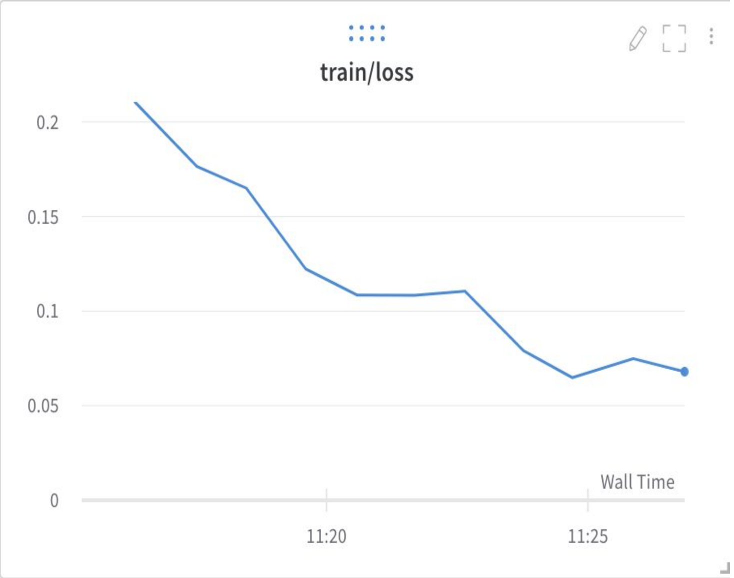


Bert Performance

evaluation on dev datasets of MRPC and SST2

	ACC	F1	inference_time
SST2	0.9220	-	1.97
MRPC	0.8504	0.8968	1.14

Training



train/loss
0.2
0.15
0.1
0.05
0
Wall Time
11:20
11:25

train/learning_rate
1.5e-5
1e-5
5e-6
0
Wall Time
11:20
11:25

Evaluation

Table 2: Evaluation on MRPC dataset

Model	MRPC	
	F1-score	ACC
Traditional ML		
Lightgbm	0.8000	0.6790
Xgboost	0.8254	0.7317
Randomforest	0.8175	0.6914
Multinomial Naive Bayes	0.8288	0.7205
SVC	0.8290	0.7230
SGDClassifier (trigrams)	0.8310	0.7328
Neural Network Model		
CNN	0.8120	0.6940
RNN	-	-
Pretrained Language Model		
BERT	0.8967	0.8505

Table 3: Evaluation on SST2 dataset

Model	SST2
	ACC
Traditional ML	
Lightgbm	0.6437
Xgboost	0.6954
Randomforest	0.6149
Multinomial Naive Bayes	0.8004
SVC	0.8348
SGDClassifier	0.8291
Neural Network Model	
CNN	0.6910
RNN	-
Pretrained Language Model	
BERT	0.9220

Conclusions

- Traditional machine learning is more efficient than neural based model.
- Traditional machine learning models also can give competitive results.
- Sometimes it is difficult to train a neural based model. It is not very stable.
- Deep learning model can get the best performance on two datasets. MRPC and SST2.

References for Naive Bayes, SVM-SVC and SGDClassifier

https://web.stanford.edu/~jurafsky/slp3/4.pdf

https://towardsdatascience.com/text-classification-using-naive-bayes-theory-a-working-example-2ef4b7eb7d5a

https://scikit-learn.org/stable/modules/naive_bayes.html

https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989

https://www.analyticsvidhya.com/blog/2021/03/beginners-guide-to-support-vector-machine-svm/

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

https://michael-fuchs-python.netlify.app/2019/11/11/introduction-to-sgd-classifier/

APPENDIX

SST2_full dataset

Linear Support Vector Machine-SVC

default model

Accuracy: 0.8233

Training time: 1601.6370995044708s

Hyperparameter optimisation using RandomizedSearchCV

Hyperparameters used:

```
kernel = ['poly', 'rbf', 'sigmoid']
C = [50, 10, 1.0, 0.1, 0.01]
gamma = ['scale','auto']
```

Best hyperparameters found: {'kernel': 'rbf', 'gamma': 'scale', 'C': 10} Training time: 1h 48 m

Model after HP optimisation:

Accuracy:0.8348

Training time: 810.6734158992767s

Multinomial Naive Bayes

default model

Accuracy: 0.8004

Training time:0.03961038589477539s

Hyperparameter optimisation using RandomizedSearchCV

```
Hyperparameters used:
'alpha': np.linspace(0.1, 1.5, 80),
 'fit_prior': [True, False]
Training time: 7.104958295822143 6s
Best hyperparameters found:
'fit_prior': True, 'alpha':
0.18860759493670887

Model after HP optimisation
Accuracy 0.786697247706422
Training time: 0.040918827056884766s
```

We can see that the default model performs slightly better than the optimised one. The reason is that Multinomial Naive Bayes default model uses alpha=1, which is the Laplace Smoothing (add-one smoothing). The smoothing priors $\alpha \geq 0$ accounts for features not present in the learning samples and prevents zero probabilities in further computations. Setting $\alpha = 1$ is called Laplace smoothing, while $\alpha < 1$ is called Lidstone smoothing. (Reference:https://scikit-learn.org/stable/modules/naive_bayes.html). So, instead of excluding unseen words and by extension sentences from the validation dataset (minimising more our dataset), we give 0 + 1 probabilities to the unseen data included in the validation set.

SGDClassifier

The model implements regularized linear models with stochastic gradient descent (SGD) learning. We fit a linear SVM with SGD, to compare its results in performance and training time with SVM-`SVC` without SGD. our intuition says that `SGDClassifier` will be much faster than SVM-SVC.

`SGDClassifier default model`

`Accuracy:` 0.8211009174311926

`Training time: 0.1601462364196777s`

We can see that its performance is imperceptibly lower than SVM-SVC but it is, indeed, much (incredibly) faster. This is the reason why SGDClassifier is preferred when dealing with with large datasets.

Hyperparameter optimisation using RandomizedSearchCV

Hyperparameters used:

```
loss = ['hinge', 'log', 'modified_huber',
'squared_hinge', 'perceptron']
penalty = ['l1', 'l2', 'elasticnet']
alpha = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
learning_rate = ['constant', 'optimal',
'invscaling', 'adaptive']
eta0 = [1, 10, 100]
```

Best hyperparameters found:

```
penalty= 'l2', loss = 'squared_hinge',
learning_rate='adaptive', eta0= 10,
alpha= 0.0001

Model after HP optimisation

Accuracy: 0.8291
Training time: 0.6275079250335693s
```

MRPC_ful dataset

SVM-SVC

Default model

Accuracy:0.7230

F-1: 0.8290

Training time: 5.220731735229492s

Hyperparameter optimisation using RandomizedSearchCV

Hyperparameters used:

```
kernel = ['poly', 'rbf', 'sigmoid']
C = [100,90, 70, 60, 50, 30, 10, 1.0,
0.1, 0.01]
gamma = ['scale','auto']
```

Best hyperparameters found:

{'kernel': 'poly', 'gamma': 'scale', 'C': 10}

Training time: 30.388631105422974s

After having found that ‘poly’ is the best hyperparameter for kernel,we introduce another hyperparameter,which is used only when kernel=‘poly’, the degree parameter.

Hyperparameters used:

kernel = ['poly']

degree = [1, 2,3,4,5,6,7,8]

```
C = [100,90, 70, 60, 50, 30, 10, 1.0, 0.1, 0.01]
gamma = ['scale','auto']
```

```
Best hyperparameters found:
{'kernel': 'poly', 'gamma': 'scale', 'degree': 2,
'C': 90}
Training time: 26.8865807056427s
```

Model after hyperparameter optimisation:

Accuracy: 0.7181

F-1: 0.8265

Training time: 3.689194917678833s

Multinomial Naive Bayes

Default model

```
Accuracy: 0.7058
F-1: 0.8219
Training time: 0.0067245960235595 7s
```

```
Hyperparameters optimisation using RandomizedSearchCV
Hyperparametres used: {
 'alpha': np.linspace(0.1, 1.5, 80),
 'fit_prior': [True, False]}
Best hyperparameters found:
{'fit_prior': True, 'alpha': 0.6139240506329113}}
```

Model after hyperparameter optimisation

Accuracy 0.7205

F-11 0.8288

Training time: 0.0372576713562011 7s

SGDClassifier

Default model

Accuracy: 0.6740

F-1:   0.7718696397941681

Training time: 0.0409023761749267 6s

Hyperparameter optimisation using

RandomizedSearchCV

Hyperparameters used:

```
loss = ['hinge', 'log',
'modified_huber', 'squared_hinge',
'perceptron']
penalty = ['l1', 'l2', 'elasticnet']
alpha = [0.0001, 0.001, 0.01, 0.1, 1,
10, 100, 1000]
learning_rate = ['constant', 'optimal',
'invscaling', 'adaptive']
eta0 = [1, 10, 100]
```

Best hyperparameters found:

```
{'penalty': 'elasticnet', 'loss': 'log',
'learning_rate': 'invscaling', 'eta0':
1, 'alpha': 0.0001}
Training time: 4.86431217193603 5s


Model after hyperparamater optimisation:

Accuracy: 0.6985
F-1: 0.8183
Training time: 0.09494614601135254s
```

MRPC_full dataset using trigrams

SGDClassifier

Default model

Accuracy: 0.7107

F-1: 0.8138

Training time: 0.031213283538818360s

Hyperparameter optimisation using RandomizedSearchCV

Hyperparameters used: same as in the unigram approach

Hyperparameters found:

```
{'penalty': 'elasticnet', 'loss':
'hinge', 'learning_rate': 'invscaling',
'eta0': 100, 'alpha': 0.0001}
Training time: 4.816795349121094s


Model after HP optimisation
Accuracy: 0.7328
F-1: 0.8310
Training time: #0.8315131187438965s
```

Multinomial Naive Bayes using trigrams

Default model

```
Accuracy: 0.7156
F-1: 0.8247
Training time: 0.007960796356201172s
```

Hyperparameters used: same as in the unigram approach

```
Hyperparameters found:{'fit_prior':
True,
'alpha': 8088607594936708}

Training time 0.6476254463195801s
```

Model after HP optimisation:

Accuracy: 0.7156

F-1: 0.8247

Training time: 0.014194488525390625s

SVM-SV

Default model

Accuracy: 0.7205

F-1: 0.8298

Training time: 4.433847665786743s

Hyperparameter optimisation using RandomizedSearchCV

Hyperparameters used:kernel = ['linear', 'poly', 'rbf', 'sigmoid']
C = [100,90, 70, 60, 50, 30, 10, 1.0, 0.1, 0.01]
gamma = ['scale','auto']

Hypeparameters found:
Best: 0.814863 using {'kernel': 'linear', 'gamma': 'scale', 'C': 100}
Training time: 31.5537319183496s

Model after HP optimisation
Accuracy: 0.7205
F-1: 0.8283
Training time: 4.94310522079467 8s

SST2 Dataset

CNN with Word2Vec embedding

Hyperparameter optimization using Keras tuner "Random Search"

Hyperparameters used: Conv_layer 1 : [32,64],

Conv_layer 2: [32,64],

Kernel size: [3,5,7],

Learning rate: [0.001, 0.01, 0.1]

Hyperparameters:

conv_layers: 32

kernel_size: 3

units: 64

learning_rate: 0.001

Score: 0.6913140416145325

F1 Score: 0.67477196

Total elapsed time: 857 s

Early stopping implemented at patience = 3 at a total number of epochs = 5 when max trials = 6

```
train_tokens = []
for i in (SST2_full["train"]["sentence"]):
  train_tokens.append(i.split())
token = Tokenizer(num_words =  10000)
token.fit_on_texts(SST2_full[ "train"]["sentence"])
train_seq = token.texts_to_sequences(SST2_full[  "train"]["sentence"])
train_data = pad_sequences(train_seq, maxlen =   75)



valid_tokens = []
for i in (SST2_full["validation"]["sentence"]):
  valid_tokens.append(i.split())
token = Tokenizer(num_words =  10000)
token.fit_on_texts(SST2_full[ "validation"]["sentence"])
valid_seq = token.texts_to_sequences(SST2_full[  "validation"]["sentence"])
valid_data = pad_sequences(valid_seq, maxlen =   75)

tokens = train_tokens + valid_tokens
text = SST2_full[ "train"]["sentence"] + SST2_full["validation"]["sentence"]


training_labels = np.array(SST2_full[ 'train']['label'])
training_labels = training_labels.astype( 'float32').reshape(-1,1)
testing_labels = np.array(SST2_full[ 'validation']['label'])
testing_labels = testing_labels.astype( 'float32').reshape(-1,1)

print('Shape of input data:' , train_data.shape, valid_data.shape)
print('Shape of labels:' , training_labels.shape, testing_labels.shape)
```

```
w2v_model = word2vec.Word2Vec(min_count = 1, size = 10000,
sg = 1, window = 5)
w2v_model.build_vocab(tokens)
w2v_model.train(tokens,
                total_examples = w2v_model.corpus_count,
                epochs = w2v_model.iter)
w2v_embedding = w2v_model.wv.get_keras_embedding()
```

```python
def word2vec_CNN (hp):
  #x = Sequential()
  sequence_input = Input(shape=( 75), dtype='int32')
  embedded_sequences = w2v_embedding(sequence_input)
  x = Conv1D(filters = hp.Choice( 'conv_layers' , [32,64]),
                  kernel_size = hp.Choice(  'kernel_size' , [3,5,7]),
                  activation =  'relu')(embedded_sequences)
  x = MaxPooling1D()(x)
  x = Conv1D(filters = hp.Choice( 'conv_layers' , [32,64]),
            kernel_size = hp.Choice(  'kernel_size' , [3,5,7]),
            activation =  'relu')(x)
  x = MaxPooling1D()(x)
  x = Flatten()(x)
  x = Dropout( 0.5)(x)
  preds = Dense( 1, activation =  'softmax')(x)
  model = Model(sequence_input, preds)
  learning_rate_choice = hp.Choice(  'learning_rate' , values = [ 0.001,  0.01,  0.1])
  model. compile (loss =  'binary_crossentropy' ,
                metrics =  'accuracy',
                optimizer = keras.optimizers.Adam(learning_rate = learning_rate_choice))
  return  model

random_search = kt.RandomSearch(
    word2vec_CNN,
    objective =  'val_accuracy' ,
    max_trials =  5,
    directory =  'dir',
    project_name =  'search'
)
print(random_search.search_space_summary())
early_stopping = tf.keras.callbacks.EarlyStopping(monitor =    'val_loss', patience =  2)
random_search.search(train_data, training_labels, validation_split =    0.2, epochs =  1,
callbacks=[early_stopping])
```

MRPC Dataset

CNN with Word2Vec embedding

Hyperparameter optimization using Keras tuner "Random Search"

Hyperparameters used: Conv_layer 1 : [32,64,96],

Conv_layer 2: [32,64,96],

Kernel size: [3,5,7],

Learning rate: [0.001, 0.01, 0.1]

Hyperparameters:

conv_layers: 64

kernel_size: 7

Dense: 64

learning_rate: 0.001

Accuracy Score: 0.6948229074478149

F1 score: 0.812227

Total elapsed time: 2820 s

Early stopping implemented at patience = 3 out of 5 epochs with max trials = 6

```
MRPC_full_train = [ ' '.join(x) for x in
zip(MRPC_full["train"]["sentence1"],MRPC_full["train"]["sentence2"])]
MRPC_full_valid = [ ' '.join(x) for x in
zip(MRPC_full["validation"]["sentence1"],MRPC_full["validation"]["sentence2"])]

train_tokens = []
for i in (MRPC_full_train):
  train_tokens.append(i.split())
token = Tokenizer(num_words = 10000)
token.fit_on_texts(MRPC_full_train)
train_seq = token.texts_to_sequences(MRPC_full_train)
train_data = pad_sequences(train_seq, maxlen = 75)
print(train_data)


valid_tokens = []
for i in (MRPC_full_valid):
  valid_tokens.append(i.split())
token = Tokenizer(num_words = 10000)
token.fit_on_texts(MRPC_full_valid)
valid_seq = token.texts_to_sequences(MRPC_full_valid)
valid_data = pad_sequences(valid_seq, maxlen = 75)
print(valid_data)


training_labels = np.array(MRPC_full['train']['label'])
training_labels = training_labels.astype('float32').reshape(-1,1)
testing_labels = np.array(MRPC_full['validation']['label'])
testing_labels = testing_labels.astype('float32').reshape(-1,1)

print('Shape of input data:', train_data.shape, valid_data.shape)
print('Shape of labels:', training_labels.shape, testing_labels.shape)
```

```
tokens = list(train_tokens + valid_tokens)
print(tokens)
w2v_model = word2vec.Word2Vec(min_count = 1, size = 10000,
sg = 1, window = 5)
w2v_model.build_vocab(tokens)
w2v_model.train(tokens,
                total_examples = w2v_model.corpus_count,
                epochs = w2v_model.iter)
w2v_embedding = w2v_model.wv.get_keras_embedding()
```

```python
def word2vec_CNN (hp):
  sequence_input = Input(shape=( 75), dtype='int32')
  embedded_sequences = w2v_embedding(sequence_input)
  x = Conv1D(filters = hp.Choice( 'conv_layers' , [32,64,96]),
             kernel_size = hp.Choice( 'kernel_size' , [3,5,7]),
             activation =  'relu')(embedded_sequences)
  x = MaxPooling1D()(x)
  x = Conv1D(filters = hp.Choice( 'conv_layers' , [32,64,96]),
             kernel_size = hp.Choice( 'kernel_size' , [3,5,7]),
             activation =  'relu')(x)
  x = MaxPooling1D()(x)
  x = Flatten()(x)
  x = Dropout( 0.5)(x)
  preds = Dense( 1, activation =  'softmax')(x)
  model = Model(sequence_input, preds)
  learning_rate_choice = hp.Choice( 'learning_rate' , values = [ 0.001,  0.01,  0.1])
  model. compile (loss =  'binary_crossentropy' ,
                metrics =  'accuracy',
                optimizer = keras.optimizers.Adam(learning_rate = learning_rate_choice))
  return model

rs = kt.RandomSearch(
    word2vec_CNN,
    objective =  'val_accuracy' ,
    max_trials =  5,
    directory =  'dir',
    project_name =  'newproj'
)
print(rs.search_space_summary())
early_stopping = tf.keras.callbacks.EarlyStopping(monitor =   'val_loss', patience =  2)
random_search.search(train_data, training_labels, validation_split =   0.2, epochs =  1,
callbacks=[early_stopping])
```

```
pipeline = Pipeline(
    [
        ("vect", CountVectorizer()),
        ("tfidf", TfidfTransformer()),
        ("clf", xgb.XGBClassifier()),
    ]
)

parameters = {
    #"vect__max_df": (0.5, 0.75, 1.0),
    "vect__ngram_range": ((1, 1), (2, 2), (3,3)),  # unigrams or bigrams
    "clf__n_estimators": (50, 100, 150),
    "clf__max_depth": (2, 3),
    "clf__learning_rate": (0.05, 0.1, 0.2 ),
}
t1 = time.time()
# Find the best parameters for both the feature extraction and the
# classifier
grid_search = RandomizedSearchCV(pipeline, parameters, scoring="accuracy",
n_jobs=-2, verbose=1)
grid_search.fit(SST2_full["train"]["sentence"], SST2_full["train"]["label"]) #Add
early stopping and calidation sets...
print(f"Time for the fit was: {time.time()-t1}s")
print(f"Refit done in {grid_search.refit_time_} s!")
print(f"Best model is: {grid_search.best_params_}")
val_func(grid_search, SST2_full["validation"]["sentence"],
SST2_full["validation"]["label"])
```

```
Fitting 5 folds for each of 10 candidates, totalling 50
fits
Time for the fit was: 481.1514041423797 6s
Refit done in 5.674318790435791 s!
Best model is: {'vect__ngram_range': (1, 1),
'clf__n_estimators': 100, 'clf__max_depth': 2,
'clf__learning_rate': 0.1}
Best 5-fold log_loss was: 0.680134413923536
Best 5-fold accuracy was: 0.6954022988505747
Best 5-fold roc_auc was: 0.7231989424983477
Best 5-fold f1 score was: 0.7195767195767195
```

```python
pipeline = Pipeline(
   [
       ("vect", CountVectorizer()),
       ("tfidf", TfidfTransformer()),
       ("clf", xgb.XGBClassifier()),
   ]
)

parameters = {
   #"vect__max_df": (0.5, 0.75, 1.0),
   "vect__ngram_range": ((1, 1), (2, 2), (3,3)),  # unigrams or bigrams
   "clf__n_estimators": (50, 100, 150),
   "clf__max_depth": (2, 3),
   "clf__learning_rate": (0.05, 0.1, 0.2 ),
}

X_train = [' '.join(x) for x in
zip(MRPC_full["train"]["sentence1"],MRPC_full["train"]["sentence2"])]
X_val = [' '.join(x) for x in
zip(MRPC_full["validation"]["sentence1"],MRPC_full["validation"]["sentence2"])]
t1 = time.time()
# Find the best parameters for both the feature extraction and the
# classifier
grid_search = RandomizedSearchCV(pipeline, parameters, scoring= "accuracy", n_jobs=-2, verbose=1)
grid_search.fit(X_train, MRPC_full[ "train"]["label"]) #Add early stopping and calidation sets...
print(f"Time for the fit was: {time.time()-t1}s")
print(f"Refit done in {grid_search.refit_time_} s!")
print(f"Best model is: {grid_search.best_params_}")
val_func(grid_search, X_val,MRPC_full[ "validation"]["label"])
```

```
Fitting 5 folds for each of 10 candidates,
totalling 50 fits
Time for the fit was: 174.78201842308044s
Refit done in 1.939793586730957 s!
Best model is: {'vect__ngram_range': (1, 1),
'clf__n_estimators': 150, 'clf__max_depth': 2,
'clf__learning_rate': 0.2}
Best 5-fold log_loss was: 0.8300971963397432
Best 5-fold accuracy was: 0.7317073170731707
Best 5-fold roc_auc was: 0.5906593406593407
Best 5-fold f1 score was: 0.8253968253968255
```

```python
pipeline = Pipeline(
    [
        ("vect", CountVectorizer()),
        ("tfidf", TfidfTransformer()),
        ("clf", RandomForestClassifier()),
    ]
)


parameters = {
    #"vect__max_df": (0.5, 0.75, 1.0),
    "vect__ngram_range": ((1, 1), (2, 2), (3,3)),  # unigrams or bigrams
    "clf__n_estimators": (50,100,150),
    "clf__max_depth": (2, 3),
    "clf__min_samples_leaf": (1, 2, 4),
    "clf__max_features": ("sqrt", "log2"),
}
t1 = time.time()
# Find the best parameters for both the feature extraction and the
# classifier
grid_search = RandomizedSearchCV(pipeline, parameters, scoring="accuracy",
n_jobs=-2, verbose=1)
grid_search.fit(SST2_full["train"]["sentence"], SST2_full["train"]["label"])
print(f"Time for the fit was: {time.time()-t1}s")
print(f"Refit done in {grid_search.refit_time_} s!")
print(f"Best model is: {grid_search.best_params_}")
val_func(grid_search, SST2_full["validation"]["sentence"],
SST2_full["validation"]["label"])
```

```
Fitting 5 folds for each of 10 candidates,
totalling 50 fits
Time for the fit was: 86.17250180244446s
Refit done in 1.5549530982971191 s!
Best model is: {'vect__ngram_range': (1, 1),
'clf__n_estimators': 100, 'clf__min_samples_leaf':
4, 'clf__max_features': 'sqrt', 'clf__max_depth':
3}
Best 5-fold log_loss was: 0.685187301753825
Best 5-fold accuracy was: 0.6149425287356322
Best 5-fold roc_auc was: 0.696298744216788
Best 5-fold f1 score was: 0.6909090909090908
```

```
pipeline = Pipeline(
    [
        ("vect", CountVectorizer()),
        ("tfidf", TfidfTransformer()),
        ("clf", RandomForestClassifier()),
    ]
)

parameters = {
    #"vect__max_df": (0.5, 0.75, 1.0),
    "vect__ngram_range": ((1, 1), (2, 2), (3,3)),  # unigrams or bigrams
    "clf__n_estimators": (50,100,150),
    "clf__max_depth": (2, 3),
    "clf__min_samples_leaf": (1, 2, 4),
    "clf__max_features": ("sqrt", "log2"),
}
# Find the best parameters for both the feature extraction and the
# classifier
X_train = [' '.join(x) for x in
zip(MRPC_full["train"]["sentence1"],MRPC_full["train"]["sentence2"])]
X_val = [' '.join(x) for x in
zip(MRPC_full["validation"]["sentence1"],MRPC_full["validation"]["sentence2"])]
t1 = time.time()
# Find the best parameters for both the feature extraction and the
# classifier
grid_search = RandomizedSearchCV(pipeline, parameters, scoring= "accuracy", n_jobs=-2, verbose=1)
grid_search.fit(X_train, MRPC_full[ "train"]["label"]) #Add early stopping and calidation sets...
print(f"Time for the fit was: {time.time()-t1}s")
print(f"Refit done in {grid_search.refit_time_} s!")
print(f"Best model is: {grid_search.best_params_}")
val_func(grid_search, X_val,MRPC_full[ "validation"]["label"])
```

```
Fitting 5 folds for each of 10 candidates,
totalling 50 fits
Time for the fit was: 22.05402970314026s
Refit done in 0.18636751174926758 s!
Best model is: {'vect__ngram_range': (1, 1),
'clf__n_estimators': 50, 'clf__min_samples_leaf':
4, 'clf__max_features': 'sqrt', 'clf__max_depth':
2}
Best 5-fold log_loss was: 0.6283055605626955
Best 5-fold accuracy was: 0.691358024691358
Best 5-fold roc_auc was: 0.6164148351648351
Best 5-fold f1 score was: 0.8175182481751825
```

```python
pipeline = Pipeline(
   [
       ("vect", CountVectorizer()),
       ("tfidf", TfidfTransformer()),
       ("clf", lgb.LGBMClassifier(objective = "binary")),
   ]
)

parameters = {
   #"vect__max_df": (0.5, 0.75, 1.0),
   "vect__ngram_range": ((1, 1), (2, 2), (3,3)),  # unigrams or bigrams
   "clf__n_estimators": (50, 100),
   "clf__eta": (0.05, 0.1),
   "clf__tree_learner": ("serial", "feature"),
   "clf__max_depth": (2, 3,),
   #"clf__max_features": ("sqrt", "log2"),
}
t1 = time.time()
# Find the best parameters for both the feature extraction and the
# classifier
grid_search = RandomizedSearchCV(pipeline, parameters, scoring="accuracy",
n_jobs=-2, verbose=1)
grid_search.fit(SST2_full["train"]["sentence"], SST2_full["train"]["label"])
print(f"Time for the fit was: {time.time()-t1}s")
print(f"Refit done in {grid_search.refit_time_} s!")
print(f"Best model is: {grid_search.best_params_}")
val_func(grid_search, SST2_full["validation"]["sentence"],
SST2_full["validation"]["label"])
```

```
Fitting 5 folds for each of 10 candidates,
totalling 50 fits
Time for the fit was: 77.4414241313934 3s
Refit done in 1.2774310111999512 s!
Best model is: {'vect__ngram_range': (1, 1),
'clf__tree_learner': 'feature',
'clf__n_estimators': 50, 'clf__max_depth': 2,
'clf__eta': 0.05}
Best 5-fold log_loss was: 0.6758634524301005
Best 5-fold accuracy was: 0.6436781609195402
Best 5-fold roc_auc was: 0.680105750165234 7
Best 5-fold f1 score was: 0.6555555555555556
```

```
pipeline = Pipeline(
   [
       ("vect", CountVectorizer()),
       ("tfidf", TfidfTransformer()),
       ("clf", lgb.LGBMClassifier(objective =  "binary")),
   ]
)

parameters = {
   #"vect__max_df": (0.5, 0.75, 1.0),
   "vect__ngram_range": ((1, 1), (2, 2), (3,3)),  # unigrams or bigrams
   "clf__n_estimators": (50, 100),
   "clf__eta": (0.05, 0.1),
   "clf__tree_learner": ("serial", "feature"),
   "clf__max_depth": (2, 3,),
   #"clf__max_features": ("sqrt", "log2"),
}
X_train = [' '.join(x) for x in
zip(MRPC_full["train"]["sentence1"],MRPC_full["train"]["sentence2"])]
X_val = [' '.join(x) for x in
zip(MRPC_full["validation"]["sentence1"],MRPC_full["validation"]["sentence2"])]
t1 = time.time()
# Find the best parameters for both the feature extraction and the
# classifier
grid_search = RandomizedSearchCV(pipeline, parameters, scoring= "accuracy", n_jobs=-2, verbose=1)
grid_search.fit(X_train, MRPC_full[ "train"]["label"]) #Add early stopping and calidation sets...
print(f"Time for the fit was: {time.time()-t1}s")
print(f"Refit done in {grid_search.refit_time_} s!")
print(f"Best model is: {grid_search.best_params_}")
val_func(grid_search, X_val,MRPC_full[ "validation"]["label"])
```

```
Fitting 5 folds for each of 10 candidates,
totalling 50 fits
Time for the fit was: 19.309007644653320s
Refit done in 0.25750041007995605 s!
Best model is: {'vect__ngram_range': (1, 1),
'clf__tree_learner': 'feature',
'clf__n_estimators': 50, 'clf__max_depth': 3,
'clf__eta': 0.1}
Best 5-fold log_loss was: 0.6965631128939668
Best 5-fold accuracy was: 0.6790123456790124
Best 5-fold roc_auc was: 0.5913461538461539
Best 5-fold f1 score was: 0.8
```

01 - Why we choose this project

BERT

- Paper about a Deep learning model
- Bert was really good at text classification

Got us thinking

- 4 weeks wasn't enough time for a complex model
- "Is this good enough" - every student as NBI at some point

Many models

- Test several models to compare to BERT
- Great opportunity to code/test a lot
- Can we get close enough?