EPAR Sentiment Analysis

Dr. Baishali Dutta

Outline

01 Problem Statement

02 Data Exploration, Cleaning and Preprocessing

03 Machine Learning Models

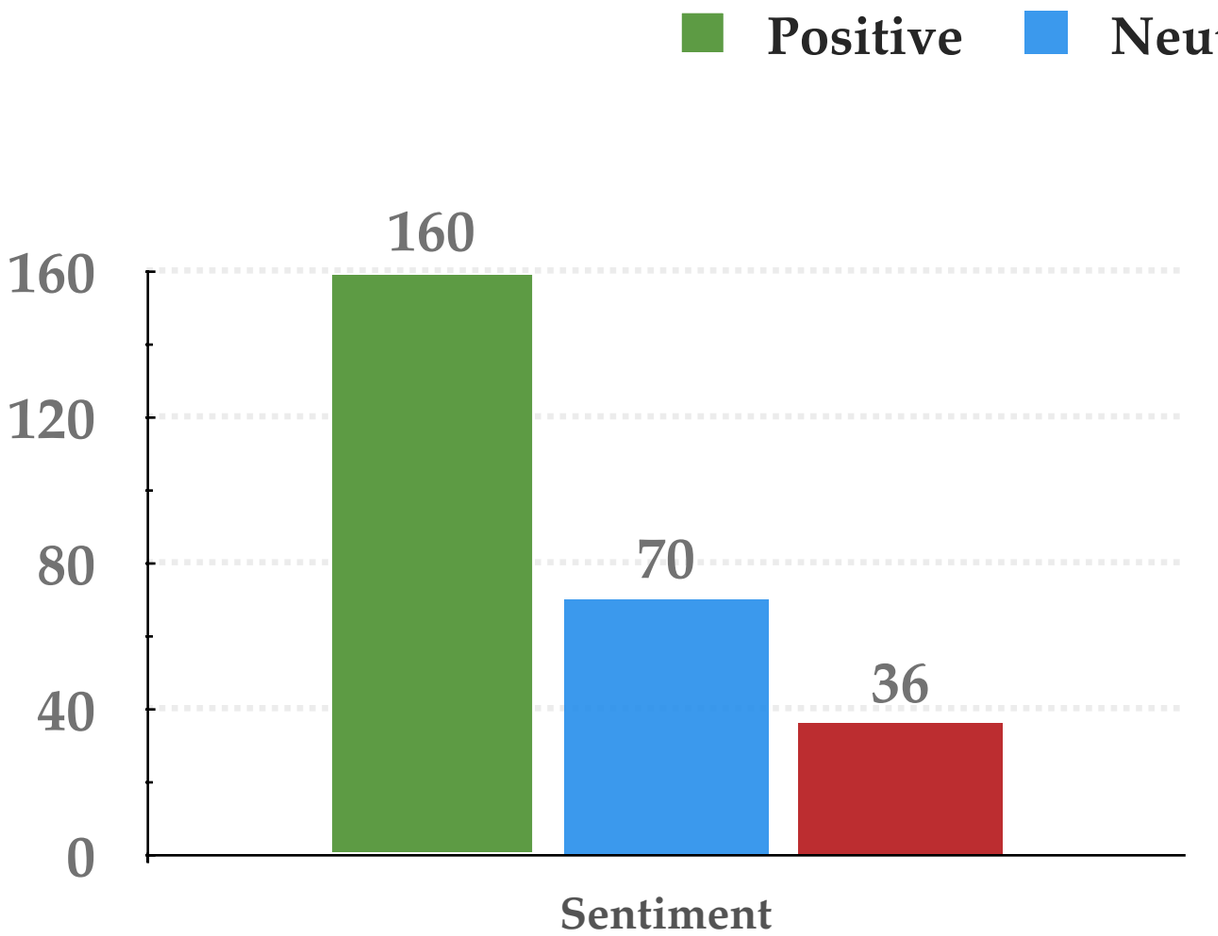
04 Further Steps

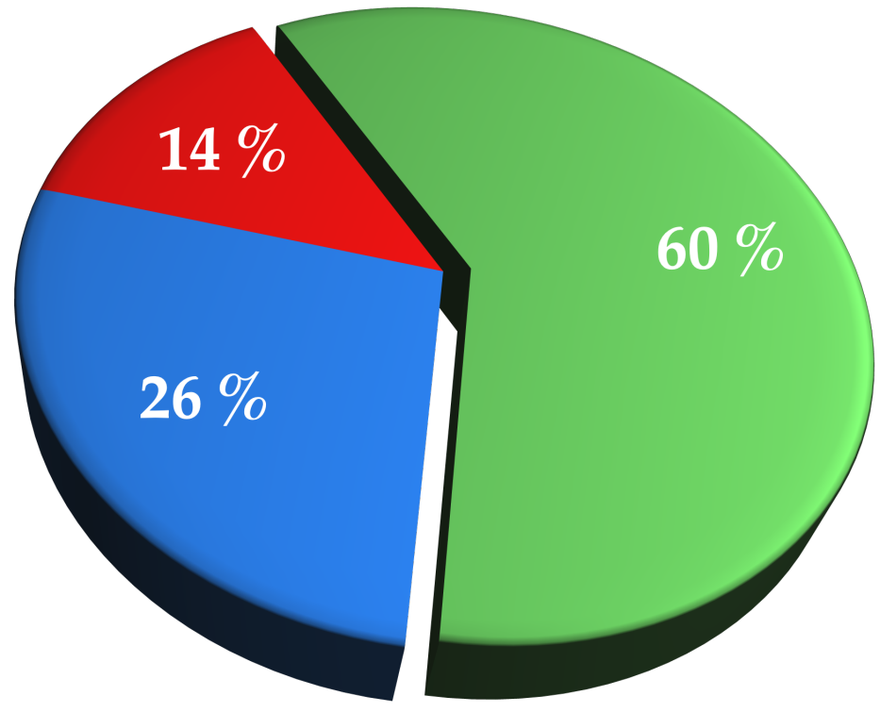
Problem Statement

EPAR Sentiment Analysis

- European Public Assessment Report (EPAR) is a scientific evaluation, submitted by pharmaceutical companies in support of a new drug or new indication for an existing drug, published by European Medicine Agency (EMA)
- 266 sentences selected from different EPARs are provided
- Sentences are manually labelled in three categories - **Positive**, **Neutral** and **Negative**
- Goal is to develop a machine learning algorithm that could automatically label the sentences for future inputs
- ***Multi-class Classification*** problem statement

Exploratory Data Analysis (EDA)





- Small unbalanced dataset, no null entries present
- Mutually exclusive labels - each sentence is associated with only one label

Data Cleaning

Text Processing

Lower Case All Words

- lower, camel or upper case written words treated as a same word
- all, All, ALL converted to all

Removed Punctuations

- . , ; : etc.
- Do not add any helpful information

Fixed Misspelled Words

- Fixed ***Proflie to Profile*** manually
- No library used as many pharmaceutical terminologies are not present in the library's English corpus

Removed Stopwords

- *the, in, among, for, where* etc.
- Do not add any helpful information

Not Used

Stemming/Lemmatisation

- Finds the base form of words
- Debatable in Sentiment Analysis
- Breaks parts-of-speech tagging
- Alters polarity of a word

Reference

Data Preprocessing

Text Processing

Transforming texts into meaningful representation of numbers using the ***TF-IDF Vectorizer***

```
X = dataset['Sentence']
vectorizer = TfidfVectorizer(max_features=2500,
                             max_df=0.8,
                             min_df=min_df)
X = vectorizer.fit_transform(X)
```

Preparing the Target Column

Positive, Neutral and Negative columns are merged into a single target column - ***Sentiment***

```
dataset['Sentiment'] = \
        dataset['Positive'] * 100 + \
        dataset['Negative'] * -100
dataset['Sentiment'] = dataset['Sentiment']\
    .map({100: 'Positive', -100: 'Negative', 0: 'Neutral'})
```

Used ***Label Encoder*** on the Sentiment Column - converted
Negative: 0, Neutral: 1, Positive: 2 (for classical machine learning classifiers)

```
y = dataset['Sentiment']
le = LabelEncoder()
y = le.fit_transform(y)
```

Multi-class Classification

- Since the given dataset is quite small, chose to look at Classical Machine Learning (ML) classifiers rather than neutral network in the beginning

Classical ML Models

- **Support Vector Machine (linear and gaussian kernel)**
- **Naive Bayes**
- **Logistic Regression**
- **Decision Tree**
- **XGBoost**
- **Random Forest**

Train-test-split (configurable)
80% training, 20% test

Cross-validation for each classifier
- k-fold
- Leave-one-out

Grid Search to find best performing model

Validation and Grid Search

Train-test split (hold-out)

- Split dataset into training and test sets (used 80/20 % splitting)
- For large dataset, this method performs well

Cross-validation (k-fold)

- Resampling procedure, useful for limited dataset, less biased than train-test-split
- Shuffles data randomly, split the dataset in k-folds, keep one fold for test, iterates until all splits are used as test

Cross-validation (leave-one-out)

- Extended version of k-fold, where k is no. of data points in the dataset

Grid Search

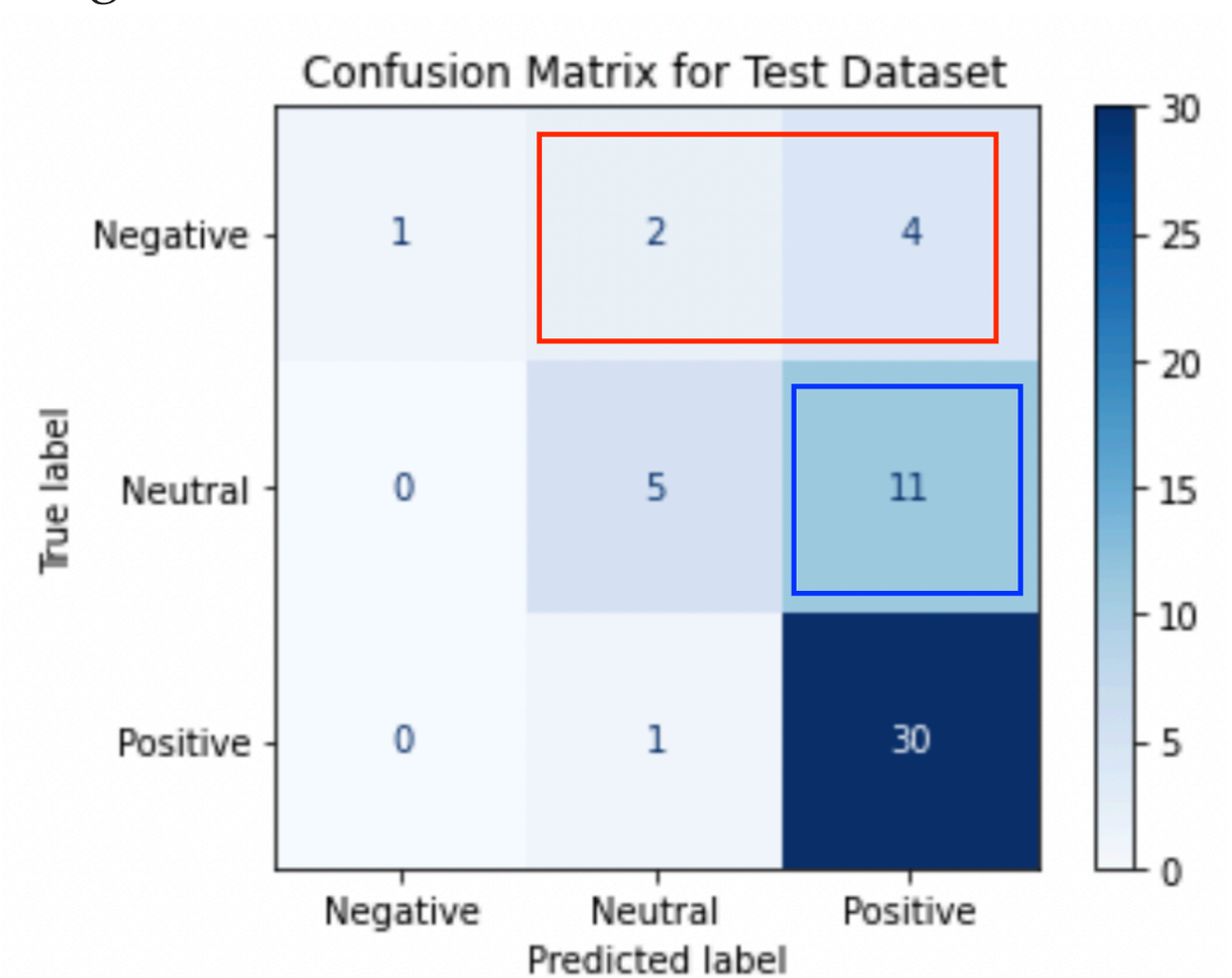
- Tunes hyperparameters of the classifier to find the optimal performance, uses the above validation at will

Support Vector Classifier (SVC)

- Supervised classical ML algorithm, finds hyper-plane that differentiates classes
- Projects data into hyper-planes using different types of kernels - linear, rbf (gaussian) etc.

Train-test split Validation

- 80% training and 20% test data, **linear kernel**



Support Vector Classifier (SVC)

- 80% training and 20% test data, **linear kernel**

	precision	recall	f1-score	support
Negative	1.00	0.14	0.25	7
Neutral	0.62	0.31	0.42	16
Positive	0.67	0.97	0.79	31
accuracy			0.67	54
macro avg	0.76	0.47	0.49	54
weighted avg	0.70	0.67	0.61	54

67 % accuracy, however recall and f1-score are not good

k-fold (k=36) cross-validation

Mean accuracy: 68.6%

Leave-one-out cross-validation

Mean accuracy: 69.5%

Support Vector Classifier (SVC)

Grid search with cross validation

Tuneable hyperparameters-

- kernel (types of hyperplane)
- Regularization Coefficient 'C' / penalty term for outliers/noise
- Kernel Coefficient 'gamma' for non-linear kernel such as rbf(gaussian)

```
parameters = [
        {
            'C': [1, 10, 100, 1000],
            'kernel': ['linear']
        },
        {
            'C': [1, 10, 100, 1000],
            'kernel': ['rbf'],
            'gamma': [0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
        }
]
```

Grid search with k-fold

Mean accuracy: 71.6%

Best Parameters

```
best_parameters = grid_search.best_params_
print(best_parameters)
```

```
{'C': 10, 'gamma': 0.9, 'kernel': 'rbf'}
```

Grid search with leave-one-out

Mean accuracy: 73.7%

Best Parameters

```
best_parameters = grid_search.best_params_
print(best_parameters)
```

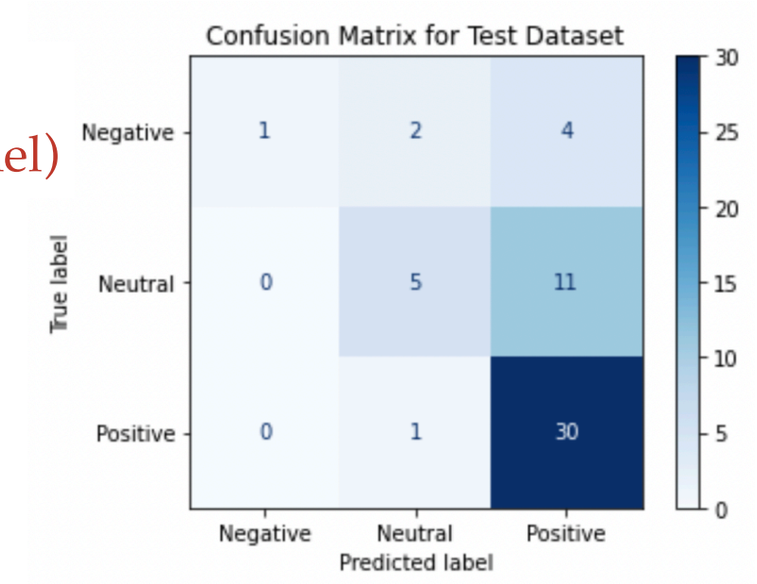
```
{'C': 10, 'gamma': 0.9, 'kernel': 'rbf'}
```

SVC (linear kernel) vs Linear SVC

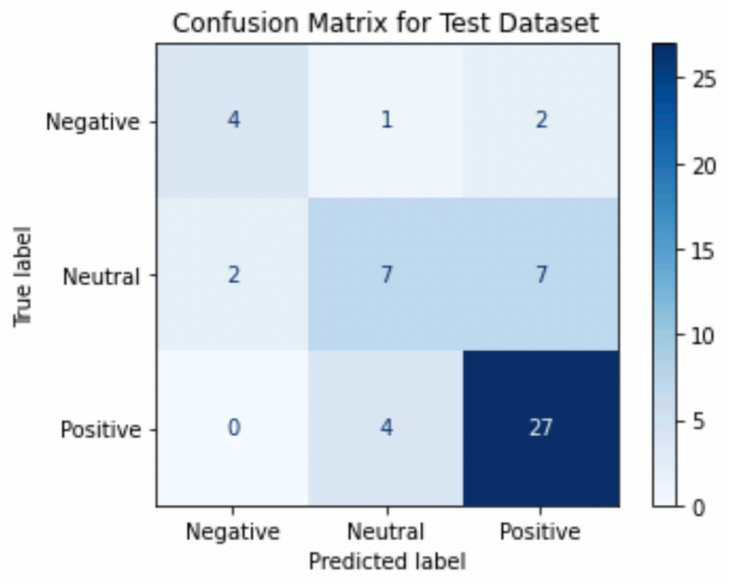
- **Linear SVC:**
  - Differs with SVC (linear kernel) in the loss function
  - Linear SVC minimises squared hinge loss while the SVC (linear kernel) minimises the regular hinge loss

- **Using train-test-split (80/20%), comparing the results of Linear SVC vs SVC (linear kernel)**

SVC
(Linear kernel)



Linear SVC



	precision	recall	f1-score	support
Negative	1.00	0.14	0.25	7
Neutral	0.62	0.31	0.42	16
Positive	0.67	0.97	0.79	31
accuracy			0.67	54
macro avg	0.76	0.47	0.49	54
weighted avg	0.70	0.67	0.61	54

	precision	recall	f1-score	support
Negative	0.67	0.57	0.62	7
Neutral	0.58	0.44	0.50	16
Positive	0.75	0.87	0.81	31
accuracy			0.70	54
macro avg	0.67	0.63	0.64	54
weighted avg	0.69	0.70	0.69	54

Tuned hyperparameters for each classifiers

Grid search with k-fold (k=36)

```
==========================================================
       Performing Grid Search with K-Folds CV
==========================================================
==========================================================
         Best Performing Model for Naive Bayes
==========================================================
{'alpha': 0.1}
Best Score: 68.11%
==========================================================
         Best Performing Model for Decision Tree
==========================================================
{'criterion': 'entropy', 'splitter': 'random'}
Best Score: 64.04%
==========================================================
         Best Performing Model for Logistic Regression
==========================================================
{'C': 1000, 'solver': 'sag'}
Best Score: 73.51%
==========================================================
         Best Performing Model for Random Forest
==========================================================
{'criterion': 'gini', 'max_depth': 50, 'n_estimators': 100}
Best Score: 72.32%
==========================================================
         Best Performing Model for XGBoost
==========================================================
{'n_estimators': 10}
Best Score: 67.46%
==========================================================
         Best Performing Model for Linear Support Vector
==========================================================
'C': 10}
est Score: 72.02%
==========================================================
         Best Performing Model for Kernel Support Vector Machine
==========================================================
'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
est Score: 73.36%
```

Tuned hyperparameters for each classifiers

Grid search with leave-one-out

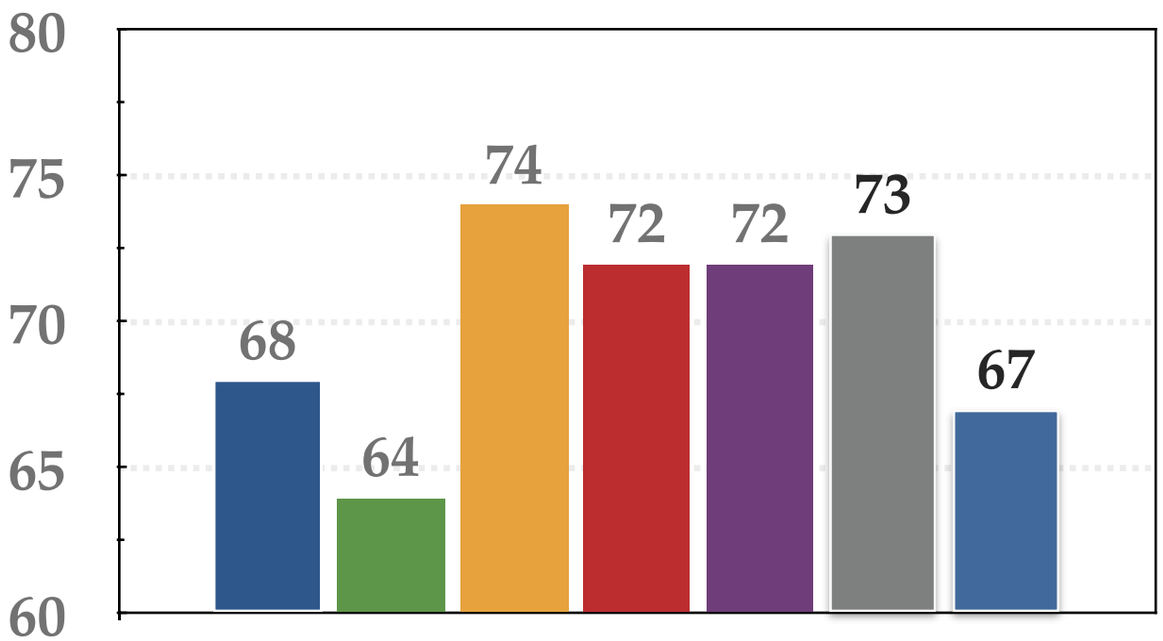
```
==========================================================
     Performing Grid Search with Leave One Out CV
==========================================================
==========================================================
         Best Performing Model for Naive Bayes
==========================================================
{'alpha': 0.001}
Best Score: 71.80%
==========================================================
         Best Performing Model for Decision Tree
==========================================================
{'criterion': 'entropy', 'splitter': 'random'}
Best Score: 66.54%
==========================================================
         Best Performing Model for Logistic Regression
==========================================================
{'C': 100, 'solver': 'newton-cg'}
Best Score: 74.81%
==========================================================
         Best Performing Model for Random Forest
==========================================================
{'criterion': 'gini', 'max_depth': 50, 'n_estimators': 100}
Best Score: 72.56%
==========================================================
         Best Performing Model for Linear Support Vector
==========================================================
{'C': 10}
Best Score: 74.44%
==========================================================
         Best Performing Model for Kernel Support Vector Machine
==========================================================
{'C': 10, 'gamma': 0.2, 'kernel': 'rbf'}
Best Score: 74.06%

==========================================================
         Best Performing Model for XGBoost
==========================================================
{'n_estimators': 10}
Best Score: 68.05%
```

Mean accuracies of best tuned models

- In both cross-validation procedures, the best accuracy model is the Logistic Regression with 74-75% accuracy

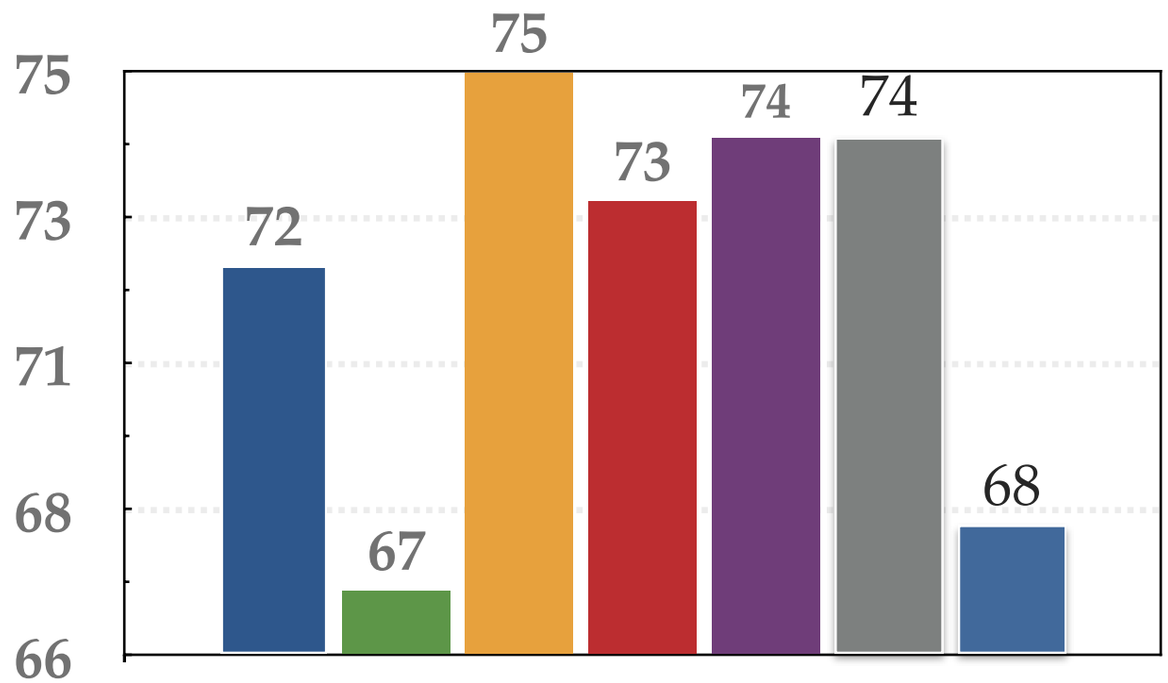
- Naive Bayes
- Decision Tree
- Logistic Regression
- Random Forest
- Linear SVC
- Kernel SVC
- XGBoost

k-fold cross-validation




```
==========================================================
         Best Performing Model for Logistic Regression
==========================================================
{'C': 1000, 'solver': 'sag'}
Best Score: 73.51%
```

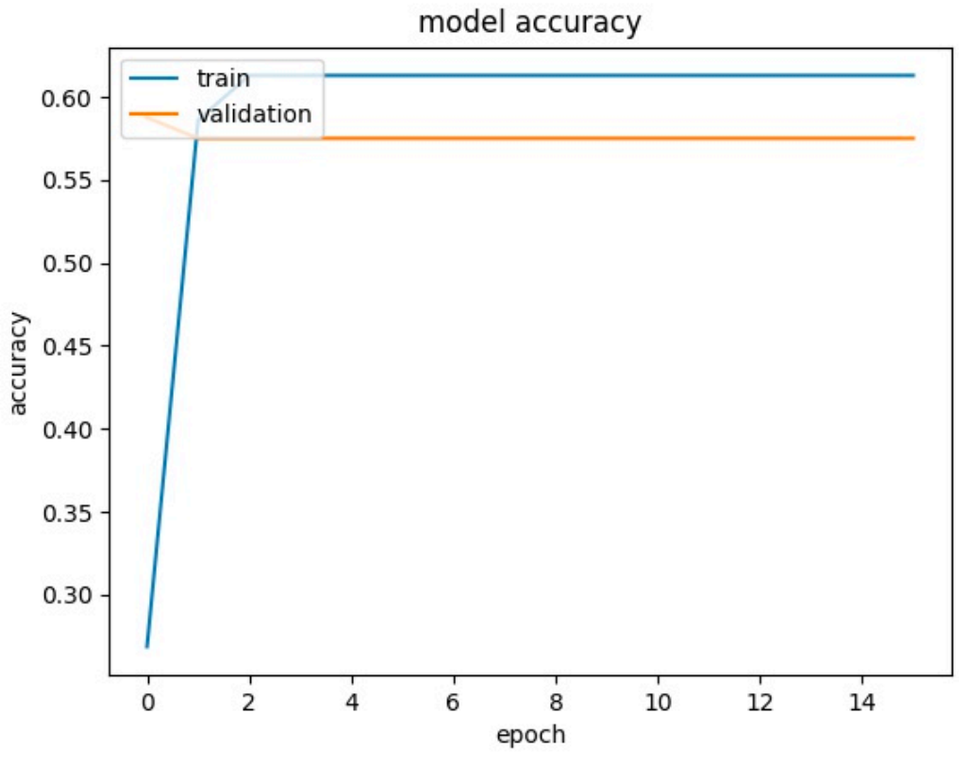
Leave-one-out cross-validation

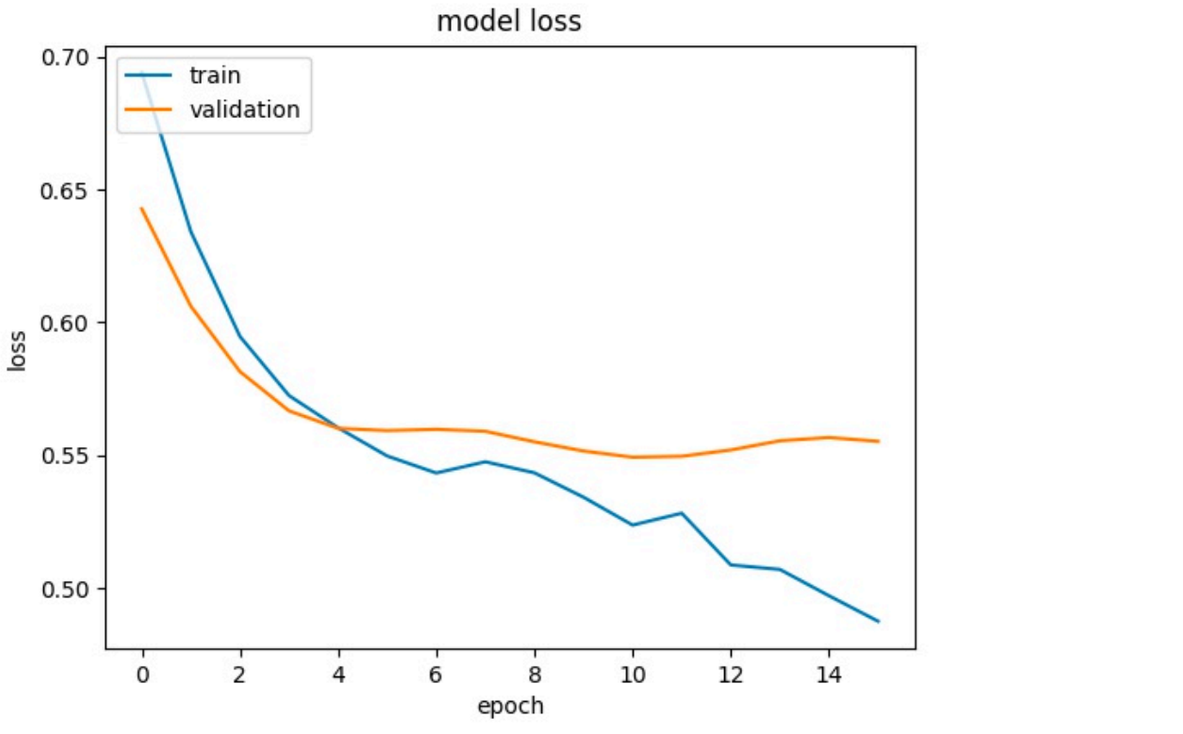



```
==========================================================
         Best Performing Model for Logistic Regression
==========================================================
{'C': 100, 'solver': 'newton-cg'}
Best Score: 74.81%
```

Deep Learning Model (Bi-directional LSTM)

- As the dataset is small, deep learning models should be avoided as they can easily be overfitted
- Since Recurrent Neural Network (RNN) is primarily built to tackle NLP problems, a quick test is performed using Bidirectional Long Short-Term Memory (LSTM) RNN model on this dataset as well





- Though the model can be optimised to a greater extent, I strongly believe that using deep learning models on 266 data points should be highly avoided (hence no further tuning is performed)

Further Steps

- Though different classifiers were considered, there still exists scope of experimenting with other non-covered classical ML classifiers/hyperparameters
- So far accuracy is considered to find the best tuned model. However, other scoring metrics (such as **recall, f1-score, roc_auc score** etc.) can be used for such an unbalanced dataset.
- **Transfer learning:** with more future data, we can use the best tuned model to partially fit the new data for incremental learning of the existing model
- In our scenario, we can build an end-to-end pipeline for the analysis, where new EPARs (PDF documents) can be provided to the pipeline and sentences can be extracted automatically from the specific section (namely ***Conclusion on Clinical Efficacy***) and use the above mentioned transfer learning approach to further improve the existing model
- As seen in the evaluation matrices, the algorithms struggle to identify the **Negative** and **Neutral** sentences. With enhanced statistical approaches (higher confidence-level/probability threshold for each decision), we can improve the model for the **Negative** and **Neutral** cases to avoid any **ambiguities** there.