# SENTIMENT IDENTIFICATION ON ROMAN URDU DATA

# OVERVIEW

- The Business problem
- Dataset features
- Data cleaning
- Exploratory data analysis
- Text normalization
- Vectorization
- Feature extraction
- Modeling (baseline and hyperparameter tuning)
- Final result
- Possible enhancements

# THE BUSINESS PROBLEM

- The challenge is to create a model that would identify sentiments given in the Roman Urdu language

# DATASET FEATURES

- **Review**: Document review  (Input field)

- **Sentiment :** Positive/ Neutral or Negative (Output field)

# DATA CLEANING AND TEXT NORMALIZATION

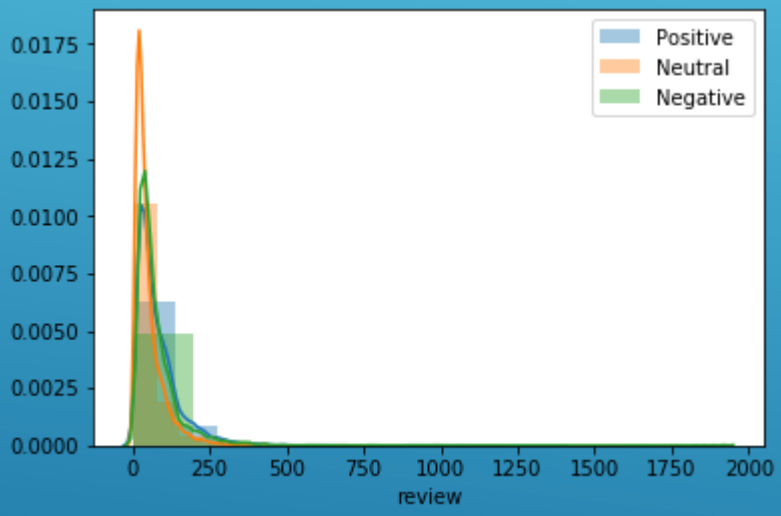
- Data quality check
- Removing na values and NULL records
- Lowercasing the words
- Removing punctuations and stop words
- Removing numbers and spaces

# EXPLORATORY DATA ANALYSIS

- Summary statistics

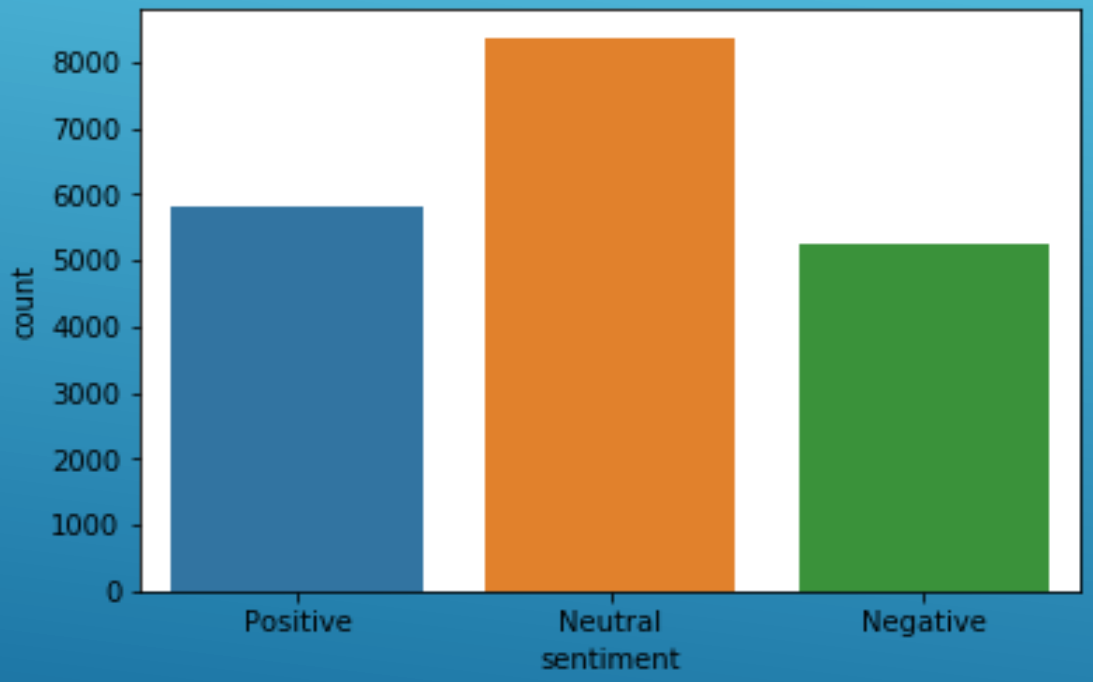
- Story generation from reviews using wordcloud

# EXPLORATORY DATA ANALYSIS




Distribution of length of reviews

# EXPLORATORY DATA ANALYSIS CONTD..




Count of reviews by sentiment

# EXPLORATORY DATA ANALYSIS CONTD..


Wordcloud for Positive words


Wordcloud for Negative words


Wordcloud for Neutral words

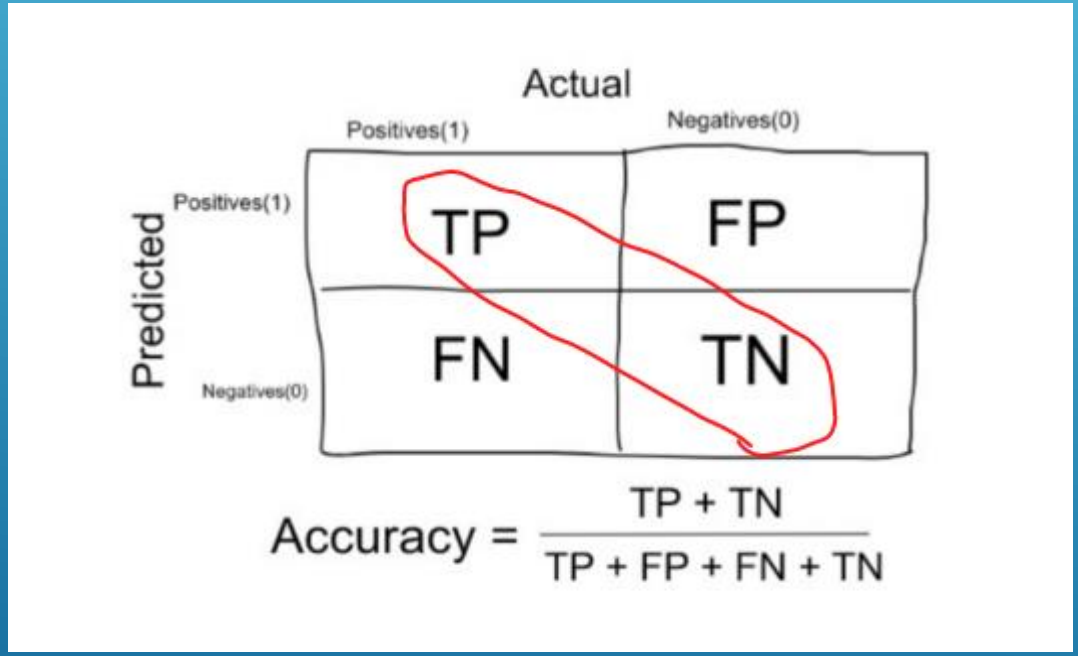
# EXPLORATORY DATA ANALYSIS CONTD..

```
Topic 0: ke | mein | ki | aur | se | ne | ka | ko | hai | bhi
Topic 1: ko | ki | allah | kay | ka | aur | me | say | ho | nay
Topic 2: hai | to | ka | ki | ko | ho | ha | ye | hain | hy
```

Topic modeling

# EVALUATION METRIC: ACCURACY

- Accuracy is the evaluation metric used here.



# TEXT PROCESSING

- Tokenization

  Splitting reviews into tokens (words)

- Vectorization

# VECTORIZATION

- CountVectorizer (Bag of words)

  Returns an encoded vector with integer count for each word

- TF-IDF

  This is to capture rarity of the word. This is to find frequent terms from the document that isn't so frequent within the whole document corpus.

- Word2vec

  Convert each word to a vector in the vector space. Vectors for the words that share context are placed next to each other. This maintains the semantics of the word based on its context.

# MODELING

- Baseline model using pipelines

  Using Logistic regression and Random forest with vectorizations

```
Estimator: Logistic Regression with CountVectorizer
Test set accuracy score: 0.647

Estimator: Logistic Regression with Tfidf
Test set accuracy score: 0.638

Estimator: Logistic Regression with w2v
Test set accuracy score: 0.551

Estimator: Random Forest with CountVectorizer
Test set accuracy score: 0.587

Estimator: Random Forest with Tfidf
Test set accuracy score: 0.595

Estimator: Random Forest with w2v
Test set accuracy score: 0.513
```

# MODELING CONTD..

- After hyper parameter tuning:

```
Estimator: Logistic Regression with CountVectorizer
Best params: {'cv__max_df': 0.8, 'cv__ngram_range': (1, 2), 'logreg__C': 1.0, 'logreg__penalty': 'l2'}
Test set accuracy score for best params: 0.649

Estimator: Random Forest with Tfidf
Best params: {'rf__max_features': 'auto', 'tfidf__max_df': 0.8}
Test set accuracy score for best params: 0.595

Classifier with best test set accuracy: Logistic Regression with CountVectorizer
```

# POSSIBLE ENHANCEMENTS:

- Balance the count of reviews on each of the sentiment category before modeling
- Word2vec performs better with bigger corpus. So we can get more data if possible.
- Use Sentence2Vec or even doc2vec where we can learn from feature representations of sentences/ documents instead of word semantics
- If one can understand the language better, a new dictionary can be made for some words (such as "not great" ) that fall under a Neutral Sentiment. If these words are found then that review can be pushed to negative.
- Look for incorrect classifications (False Positives and False Negatives) and find a pattern which is being missed by the model to use it for maximizing the model's capacity
- Use the vectors generated from LDA as a features in the classification model
- Use other models such as SVM, KNN, Naïve Bayes etc
- Use Randomizedsearch to narrow down the best parameters before using Gridsearch
- - Use Eli5 to debug transformations such as Count vectorizer/ tfidf vectorizer and understand the decisions behind the model.
- - Make the code more modular/ OOPS oriented

# REFERENCES:


- - https://scikit-learn.org/stable/auto_examples/model_selection/grid_search_text_feature_extraction
- - http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/ for Word2vecml
- - https://towardsdatascience.com/unsupervised-nlp-topic-models-as-a-supervised-learning-input-cf8ee9e5cf28 for Topic modeling
- - http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/ for Word2vec
- - https://www.kdnuggets.com/2018/01/managing-machine-learning-workflows-scikit-learn-pipelines-part-3.html for pipelines