# Title Slide

Hello everyone. My name is Karim, and today I will be presenting my work on Drug Review Sentiment Classification using different NLP approaches.

# Content Slide

A brief overview of what I will discuss includes: An intro to Natural Language Processing and Sentiment Analysis. Describe the dataset. Talk about some of the literature online related to the topic. Discuss the methodology used. Show the results. Additionally, highlight one of the advanced models, BERT and its application. And conclude with further study ideas.

# Natural Language Processing Slide

NLP is basically a family AI tools that help machines understand human languages and derive meaning from language, whether it’s written text or speech.

There are multiple applications for NLP, some of them include:

* Text classification
* Sentiment Analysis, which is the focus of my project
* Speech recognition
* Text and speech generation
* Translation
* Auto-correct

# Sentiment Analysis Slide

In this project, I focused Sentiment analysis or classification allows us to classify text or speech into pre-defined sentiments or class: whether positive, negative, neutral, etc.

The goal here is to train a machine learning algorithm on some data which is already labelled or tagged. Then extract features from this data and train the model. This is so any new untrained data can be classified into one of the pre-defined sentiment.

For example, in the sentences below, the model I built was able to assign a value to the sentence: 1 is positive, 0 is negative.

# The Dataset

The dataset I used is the drug review dataset, which has patient reviews on multiple drugs treating different conditions. The review has a rating (positive, negative) reflecting the patient’s overall all satisfaction.

This dataset was obtained by crawling online Pharma websites, in this case it was druglib and drugs.com. I downloaded from the UCI repository website.

The data has 121,573 reviews. I split it 60:20:20

Some stats: the median review length is 112 words (including stop words, and other characters)

There is small class imbalance which at first impacted my results (f1 score). So I solved there using two methods:

1. Class weight multiplier: So the under-represented class has a high weight when training the model.
2. Oversampling from the underrepresented class. Basically duplicating some of the records to get the desired 50:50 split.

# Literature Review Slide

There wasn’t much literature related to the dataset itself. However, A few of the papers online that I found useful are:

1. Law et al used standard ML models to classify travel reviews sentiment. They used SVM and Naïve Bayes and achieved good accuracy.
2. Carley et al discusses a new variation on LSTM networks which is called an attention network. I tried to create a simple version of this model in this project.
3. Rao et al uses an LSTM model which works well with textual data.
4. Munikar and company applied the BERT transformer model on the movie reviews dataset. I applied this on my dataset as well. And it worked very well.
5. Lian et al. uses inherent word embeddings, basically create embeddings specific to the dataset rather than use pre-defined ones
6. Lastly, chen et al uses an interesting mix of CNN and LSTM where a CNN model is combined with an LSTM layer to produce high accuracy

# Methodology Slide

My process flow for tackling this process is:

1. Start with the dataset, prepare the file,
2. Then pre-processing the text features, clean it, vectorize and tokenize.
3. Assigned weights to the pre-processed words, using either TF-IDF or word embeddings
4. Apply different ML learning models
5. Apply Deep learning models
6. Then finish with the main model for the project, which is the BERT transformer. I will then use the BERT model to classify any new text data.

I used python with notebooks in colab pro to maximum tpu

I use the f1 score metric (precision, recall). The AUC score (area under the ROC curve)

And the binary accuracy

# Data Preparation Slide

First I needed to merge the multiple CSV files of the data since it came from different websites. I removed some unnecessary columns that described the drugs, conditions, and such., I kept the text feature, and the labels.

Then I did some basic EDA on the data. For instance, the distribution of words is non-uniform. I suspect that the two website from which the data was extracted have different character limits.

Nonetheless, there were no major differences between the positive and negative classes.

One interesting observation is that The number of stop words and other punctuation was also similar between the two datasets. The median was around 50 words, and removing stop words brings down the word count significantly.

# Pre-processing the text Slide

One the files were ready, I can apply pre-processing methods:

First:

1. I applied spell checking and correction using the Pyspelling library in Python. I used a strict threshold for correction so to not over correct some of the text, such as the drug names in the xt
2. Remove any HTML tags and characters, of which there was a lot given the crawling method
3. Remove any URL strings
4. I used the NLTK tool kit in python to do the following:
   1. Remove punctuation.
   2. Remove stop words: the, of, and, …etc
   3. Apply lemmatization which is similar to stemming in that in deprecates the word’s last few characters but differs in that it uses the contexts to apply it rather than do it for all words.

In this example, Before pre-processing, the word count was 128 words. After pre-processing, it was brought down to 63, reduced by more than 50%. This is the main purpose of feature engineering is to reduce the dimensionality of the data. The word count is also more uniformly distributed.

# Tokenization and Vectorization Slide

Once the data is cleaned, I used two different methods for tokenization:

1. Use the TF-IDF method to assigned weights to the words. I applied this to the simple machine learning models I ran.
2. For the Deep Learning models, I used the skip-gram methodology:
3. The CBOW method uses the surrounding words or context to predict the next words.
4. Whereas the skip-gram method does the opposite. It uses the word to predict the context or surrounding words. This is actually the model that FB, Google recommend for this application even though it might be slower.

A skip gram basically takes the word and checks the surrounding to see the probability that they appear together.

# Word Embeddings Slide

Another method to assign weights to words is the word embeddings:

* Word embeddings are basically a learned representation of the text where the words that have the same meaning have a similar representation.

There are three main embeddings:

1. Word2vec by Google: older model, 2013
2. GloVe by Stanford Uni: new model, uses word co-occurrences in sentences
3. fastText by facebook: takes word parts into account. Better able to handle out of voacb words (unlike word2vec)

Both are downloadble and cab be applied on text

I did not use word2vec because while it’s the original model, fasttext is similar but more accurate because it includes word frequencies. GloVe and FastText and more granular

Word2Vec takes texts as training data for a neural network. The resulting embedding captures whether words appear in similar contexts.

GloVe focuses on words co-occurrences over the whole corpus. Its embeddings relate to the probabilities that two words appear together.

FastText improves on Word2Vec by taking word parts into account, too. This trick enables training of embeddings on smaller datasets and generalization to unknown words.

# Results ML

* using a simple pipeline to tokenize then apply the model.
* I only used TF-IDF to create a simple pipeline to iterate through the models.
* Surprisingly, the logistic regression model produce the best results at .91 accuracy
* Naïve Bayes, SVM were very similar
* Ensemble methods did not do well, which makes sense given the type of problem we have.

# Results Deep Learning

* I applied 4 different Deep Learning Models using 3 different word embeddings
* I used inherent word embeddings: simple tokenization on text data with no external weights
* I use GloVe and FastText embeddings which give weights to words based on each embedding

The models I applied are:

* Bi-directional LSTM to read text from both directions
* CNN model with an LSTM layer, so it applies the filters on sequneces, maxpools them, then adds an LSTM layer
* Pooled GRU, similar to LSTM
* HAN (Hieracrhical Attnetion Network) similar to LSTM but with a time-distributed layer (time or sequential)

Generally all models performed well

For the inherent embeddings I found that the CNN-LSTM model performed best, but took the longest to train, 1 hour per epoch

For glove, the LSTM model performed best, 0.92 accuracy

For fastText, the HAN model performed best, though fastText didn’t do as well as Glove

# BERT Model

Lastly, the main model I focused on was the BERT model.

BERT – Bidirectional Encoder Representation From Transformer. It’s a fairly recent model. The first one was released two years ago by Google researchers

It is an encoder model, so It is a language representation model which every input is connected with every output, the weights are updated based on that.

It can read text input sequentially at the same time from both directions

Trained on 3.3 mill words: both Wikipedia and online crawl

There are different variations of this model, depending on the number of parameters and layers uses.

I used the base BERT model. I also used one the newest variation called ELECTRA base, which is essentially the same concept but uses a different input masking method. It runs slightly faster and more accurate. It is discriminative, similar to GAN. 12 base layers

“Efficiently Learning an Encoder that Classifies Token Replacements Accurately

# Results and Application

To run the model, I used Tensoreflow where I load the model architecture and weights.

* I also needed to convert my data from csv, or array into a tesnor so I can input the model.
* It wasn’t fully cleaned and pre-processed. I only removed the HTML tags and URL Links. This is because BERT can understand the sentence with all the impurities that may exist.
* [BERT-Base](https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/3), [Uncased](https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/3) and [seven more models](https://tfhub.dev/google/collections/bert/1) with trained weights released by the original BERT authors.
* [Small BERTs](https://tfhub.dev/google/collections/bert/1) have the same general architecture but fewer and/or smaller Transformer blocks, which lets you explore tradeoffs between speed, size and quality.
* [ALBERT](https://tfhub.dev/google/collections/albert/1): four different sizes of "A Lite BERT" that reduces model size (but not computation time) by sharing parameters between layers.
* [BERT Experts](https://tfhub.dev/google/collections/experts/bert/1): eight models that all have the BERT-base architecture but offer a choice between different pre-training domains, to align more closely with the target task.
* [Electra](https://tfhub.dev/google/collections/electra/1) has the same architecture as BERT (in three different sizes), but gets pre-trained as a discriminator in a set-up that resembles a Generative Adversarial Network (GAN).
* BERT with Talking-Heads Attention and Gated GELU [[base](https://tfhub.dev/tensorflow/talkheads_ggelu_bert_en_base/1), [large](https://tfhub.dev/tensorflow/talkheads_ggelu_bert_en_large/1)] has two improvements to the core of the Transformer architecture.

Once I got the results, I created a simple user input where new text can be assigned weights.

# Conclusion Slide

1. I was surprised that the simple logistic regression performed well on the pre-processed data
2. The BERT model produced the best results. Using Electra was more efficient.