Mark Kim

Professor Anagha Kulkarni

CSC 620

December 10, 2022

I Regular Expressions

Regular expression (RegEx) is a domain specific language that allows us to search for lexical patterns in a corpus. By applying such expressions, we can normalize text by removing stop words, punctuation, etc. This tool, however, can be indiscriminate in its application. Nevertheless, it can be a powerful tool for finding (and/or replacing) text according to static rules set by the user.

II Edit Distance

Edit distance is a method of quantifying similarities or dissimilarities between text. If two texts have a low edit distance, they are highly similar, and high edit distance means the texts have low similarity. *Minimum* edit distance simply quantifies the minimum number of editing operations it takes to convert one string to the next. These edit operations consist of *insertion*, *deletion*, and *substitution*. The Levenshtein formulation of operations applies a cost for each operation as follows:

- Insertion: 1
- Deletion: 1
- Substitution: 2

The operations alone does not allow us to find the minimum distance. One must also take into account *alignment* to find the minimum distance.

III N-gram based Language Modeling

N-gram based language modeling is a method of modeling that uses the counts of words in a corpus to determine the probability that a particular word will occur. This type of modeling is based off of the Chain Rule of Probability. This means that the probability of a particular sequence of words is simply the product of the probabilities of each word that occurs in the corpus given the words preceding it:

$$P(w_1, w_2, \cdots, w_n) = \prod_i P(w_i \mid w_1 w_2 \cdots w_{i-1}).$$

It is sufficient, however, to simplify these calculations as follows:

- Unigram Model: $P(w_1, w_2, \cdots, w_n) \approx \prod_i P(w_i)$
- Bigram Model: $P(w_1, w_2, \cdots, w_n) \approx \prod_i P(w_i \mid w_{i-1})$

IV Text Classification using Naïve Bayes

The Naïve Bayes method of text classification relies on the Bayes Rule which states

$$P(c \mid d) = \frac{P(d \mid c)P(c)}{P(d)},$$

where c is the class and d is the document (the denominator is ignored for our use). Then our predicted class will be

$$\begin{aligned} C_{MAP} &= \operatorname*{argmax}_{c \in C} P(c \mid d) \\ &= \operatorname*{argmax}_{c \in C} P(d \mid c)P(c) \\ &= \operatorname*{argmax}_{c \in C} P(x_1, x_2, \cdots, x_n \mid c)P(c) \end{aligned}$$

Despite the fact that probabilities of features (which can be words, characters, bigrams, etc.) are not independent given a document class c, we can approximate the probabilities of those features

(given a class c) as follows:

$$C_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{x \in X} P(x \mid c).$$

We need to find the Maximum Likelihood Estimates by using the relative frequencies in the data which can be calculated by

$$\hat{P}(x_i \mid c_j) = \frac{\operatorname{count}(x_i, c_j)}{\sum_{k=1}^{n} \operatorname{count}(x_k, c_j)}$$

Once trained, we use our argmax function to predict/assign a class label to the document we are trying to classify.

V Text Classification using Logistic Regression

Similar to Naïve Bayes, Logistic Regression uses a feature set for text classification and is also uses a supervised learning model. Logistic regression uses the statistical modelling technique called regression analysis. This method of analysis consists of two parts:

1. The Objective/Loss Function
2. Optimization of the Objective Function by using Stochastic Gradient Descent

The Objective Function tells us the loss (number of classification errors) when using a particular weight vector $\vec{w}$ and bias b when using the linear regression equation

$$\begin{aligned} z &= \vec{w} \cdot \vec{x} + b \\ &= \left(\sum_{i=1}^{n} w_i x_i \right) + b \end{aligned}$$

For a single class evaluation (class a/not class a), our MLE is the sigmoid function

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}.$$

For multiclass evaluation, we use the softmax function which essentially generalizes the sigmoid function with

$$[\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_k] = \text{softmax}\left(\vec{Z}\right) = [\text{softmax}\left(Z_1\right), \text{softmax}\left(Z_2\right), \cdots, \text{softmax}\left(Z_k\right),]$$

where

$$\hat{y}_i = P(y = i \mid x) = \text{softmax}(Z_i) = \frac{e^{\vec{w}_i \cdot \vec{x} + b_i}}{\sum_{j=1}^{k} e^{\vec{w}_j \cdot \vec{x} + b_j}}, \quad 1 \leq i \leq k$$

which gives us a probability distribution of classes.

During the training step, we compare the ground truth labels with the MLE results with a Cross-Entropy Loss Function. This loss function is then optimized (minimized) via Gradient Descent (e.g. calculating the partial derivative, and using that to "descend" towards a lower minimum and repeating).

VI Text Classification Evaluation

Evaluating text classification can be done using an *extrinsic*, or *intrinsic* approach. For the intrinsic case, we do a qualitative evaluation, whereas for the extrinsic case, we use the models to see which produces better results. Using intrinsic evaluation, labelled data is split between a training and test set. The model is trained on the training set, then the model's predictions for the test set are compared against the ground truth labels to see how well the predictions match. *Precision* and *recall* are the metrics used to evaluate the effectiveness of the classification model. Precision measures the percent of a true positive prediction that were actually that ground truth class:

$$P = \frac{tp}{(tp + fp)}, \quad \text{where } tp = \text{true positive, and } fp = \text{false positive.}$$

Whereas, recall measures the percent of a ground truth class that are predicted as that particular class:

$$R = \frac{tp}{(tp + fn)}, \quad \text{where } tp = \text{true positive, and } fp = \text{false negative.}$$

A combined metric called F-measure will assess the inherent tradeoff between precision and recall with

$$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}, \quad 0 \leq \beta \leq \infty.$$

VII Vector Semantics

Vector semantics are a type of lexical semantics where words are represented as a vector. This vector representation captures the multi-dimensional attributes of a word such as its similarity or relation to other words, or context. This meaning of words, called *word embeddings*, can be statically engineered (feature engineering) or dynamically found through various methods like using deep learning algorithms. Once trained on a data set, these vectors can be compared to other word vectors to determine their similarity by measuring the difference in angles between them using cosine, which is based off of the vector dot product of the two vectors. This can be computed as:

$$\text{cosine}(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|v||w|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2} \sqrt{\sum_{i=1}^{N} w_i^2}}.$$

These comparisons can then be used to model language, text classification, and as I have recently learned other applications that may not be related to language.

VIII Feed-Forward Neural Networks

Feed-Forward neural networks are the simplest type of multi-layer neural network. These networks consist of inputs values, one or more hidden layers, and an output layer. Their standard architec-

ture is of fully connected layers, which means that each node of the previous layer has weights for every node of the current layer. These feed-forward neural networks basically expand upon linear regression; instead of doing a single linear regression calculation for a single node, many linear regressions are applied over the number of nodes in each layer of the neural network. This calculation is applied through matrix multiplication which yield the output vectors for the next layer which are the inputs for the next matrix multiplication, and so on. This part of the training is the *forward pass*. The weights are then updated by back-propagating the weight updates in the *backward pass*.

IX Sequence Processing using Recurrent Neural Networks

Recurrent neural networks provide a solution to the inherently sequential nature and varying length of language. In a recurrent neural network, inputs are presented one at a time. The network contains one or more cycles in its hidden layers (i.e. connections) which provide a memory or context for each sequence step. Because of this ability to provide that memory, prior context is no longer static as it is in a sliding window approach. This works by combining the hidden layer values in a previous sequence step with the input of the current sequence step. This is done by adding the matrix multiplication results from the corresponding weights for the input and the previous hidden layer.

X Sequence Processing using Transformers

RNN's have two primary issues:

1. their cyclical nature produces a vanishing gradient (e.g. multiplying numbers less than 1 produce ever decreasing magnitudes)

2. the inherent sequential nature of RNN's make distributed parallel computation difficult.

Transformers were developed to solve these problems. In a transformer, cycles are eliminated with a return to architectures similar to FFNN's. This newer neural network architecture introduced the concept of a self-attention layer. This layer has access to all the input items preceding it and no access to inputs beyond the current one in the sequence. The self-attention layer then is able to assign levels of attention to all previous words by comparing the words (their embeddings) using a dot product of the word vectors, which produces a scalar value. We come back to the softmax function that constrains the scalar value to a probability value, which is then used to weight the input vectors. The aggregated value, which is calculated by summing the values of all the previous weighted input vectors, generates the output vector. Perhaps the most intriguing aspect of this type of processing is that each input embedding plays three roles with respect to the attention process:

query the current focus of attention when being compared to all preceding inputs

key a preceding input when compared to the current focus of attention

value a value to compute the output of the current focus of attention.

Each of these roles are associated with its own parameter matrix. Finally, each of these parameter matrices transforms the original input embedding into a new embedding that captures all of the role-specific information.