



**NANYANG
TECHNOLOGICAL
UNIVERSITY**
SINGAPORE

EE6405

Natural Language Processing

Dr. Simon Liu
NTU Electrical and Electronic Engineering

Evaluation Metrics

Word Embeddings



Evaluation Metrics



Evaluation Metrics

- After the process of text classification, there is a need for us to evaluate the accuracy of our models quantitatively.
- Evaluation metrics assist us in the following tasks:

Objective Comparisons

Metrics provide a common framework for assessing and comparing performance, allowing us to objectively compare the performances of our NLP models.

Model Selection

Evaluation metrics help us make informed choices in selecting the best-performing model or system among multiple candidates.

Hyperparameter Tuning

When tuning hyperparameters and optimising models, evaluation metrics guide the process. Evaluation metrics allow us to assess the impact of hyperparameter choices and select the set of hyperparameters that lead to the best performance.

Evaluation Metrics

- Text models are generally split into two types;

- Predictive: Models trained to make predictions or classifications based on input data. These are our classification models and regression models.

- Evaluation metrics include:

- Confusion Matrix
- F1 Scores
- Area Under Curve (AUC-ROC)

- Generative: Models trained to create new data, typically in the form of text. These come in the form of AI Chatbots, translators and text summarisers.

- Evaluation metrics include:

- BLEU
- ROUGE
- METEOR

Confusion Matrix

A confusion matrix is an $N \times N$ matrix, where N is the number of predicted classes.

- For a binary prediction, the confusion matrix will be a 2×2 matrix.
- A 2×2 matrix will feature 4 different combinations of predicted and actual values.
 - **True Positive (TP):** Accurately predicted positive values.
 - **True Negative (TN):** Accurately predicted negative values.
 - **False Positive (FP):** (Type 1 Error): Negative values inaccurately predicted to be positive.
 - **False Negative (FN):** (Type 2 Error): Positive values inaccurately predicted to be negative.

Confusion Matrix

Other metrics can be calculated from the confusion matrix, including:

- **Accuracy:** the proportion of the total number of predictions that are correct.
- **Positive Predictive Value or Precision:** the proportion of positive cases that are correctly identified.
- **Negative Predictive Value:** the proportion of negative cases that are correctly identified.
- **Sensitivity or Recall:** the proportion of actual positive cases which are correctly identified.
- **Specificity:** the proportion of actual negative cases which are correctly identified.
- **Rate:** It is a measuring factor in a confusion matrix. It has also 4 types TPR, FPR, TNR, and FNR.

Confusion Matrix		Target			
		Positive	Negative		
Model	Positive	a	b	Positive Predictive Value	$a/(a+b)$
	Negative	c	d	Negative Predictive Value	$d/(c+d)$
		Sensitivity	Specificity	Accuracy = $(a+d)/(a+b+c+d)$	
		$a/(a+c)$	$d/(b+d)$		

Confusion Matrix

Accuracy:

- Defined by $\frac{TP+TN}{TP+TN+FP+FN}$
- Provides an overall assessment of the model's correctness.
- Can be misleading when dealing with imbalanced datasets, where one class significantly outnumbers the other.

Precision:

- Defined by $\frac{TP}{TP+FP}$
- Quantifies the proportion of positive predictions that were correct.
- Used in cases where false positives are costly or undesirable.

Recall:

- Defined by $\frac{TP}{TP+FN}$
- Measures the proportion of actual positive cases that were correctly identified by the model.
- Used when missing a positive can have serious consequences.

Micro and Macro Metrics

- Micro and macro evaluation metrics are two different approaches to aggregating and reporting performance measures, such as precision, recall, and F1-Score.

Micro Metrics:

- Aggregate the contributions of all classes to compute the average metric.
- Micro Precision:
 - Calculates the precision for each class individually, sums up the numerators (true positives), and divides by the sum of the denominators (true positives and false positives) across all classes.

- Formula:
$$\frac{TP_1 + TP_2 + \dots + TP_N}{TP_1 + TP_2 + \dots + TP_N + FP_1 + FP_2 + \dots + FP_N}$$

Macro Metrics:

- Evaluate the model's performance on each class independently and then average the results.
- Macro Precision:
 - Calculates the precision for each class individually and then takes the average of these precision scores.

- Formula:
$$\frac{Precision_1 + Precision_2 + \dots + Precision_N}{N}$$

Micro and Macro Metrics

- Micro metrics are used when overall classification performance needs to be emphasized, giving equal importance to all instances. They are particularly useful when class imbalance is present, as they consider all instances collectively.
- Macro metrics are used to evaluate the model's performance on each class independently and then average the results. They are suitable when each class is considered equally important and the model's ability to perform well on all classes is to be assessed.

Confusion Matrix

To apply confusion matrices, let's use last week's results on the IMDB dataset:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix
import seaborn as sns
#generates a confusion matrix between hand labelled data and model predictions
def getConfMatrix(pred_data, actual):
    conf_mat = confusion_matrix(actual, pred_data, labels=[0,1])
    accuracy = accuracy_score(actual, pred_data)
    precision = precision_score(actual, pred_data, average='micro')
    recall = recall_score(actual, pred_data, average='micro')
    sns.heatmap(conf_mat, annot = True, fmt=".0f", annot_kws={"size": 18})
    print('Accuracy: ' + str(accuracy))
    print('Precision: ' + str(precision))
    print('Recall: ' + str(recall))
```

Import the various metrics associated with confusion matrices

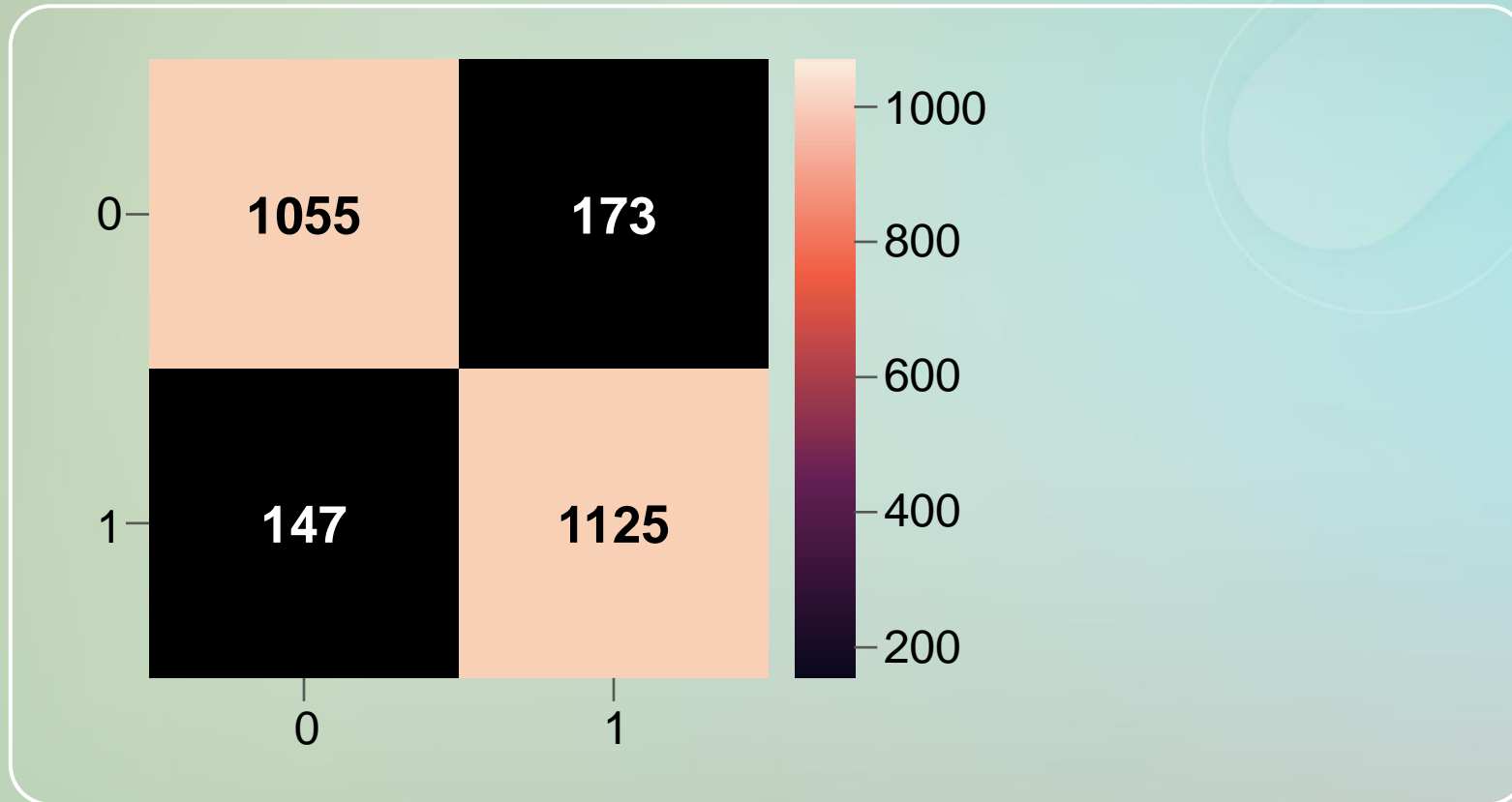
Import seaborn for visualisation

Generating confusion matrix

Visualising the matrix with heatmap

Confusion Matrix

To apply confusion matrices, let's use last week's results on the IMDB dataset:



F1 Score

- The harmonic mean of precision and recall values for a classification problem.
- Allows us to optimise both precision and recall values for a classification problem.
 - The formula is given by: $\left(\frac{recall^{-1}+precision^{-1}}{2}\right)^{-1} = 2 \cdot \frac{precision \cdot recall}{precision+recall}$
 - Harmonic mean is used in place of an arithmetic mean as it gives a more balanced measure when dealing with extreme values.
 - Consider an extreme scenario where we have precision of 0.9 and recall of 0.1
 - An arithmetic mean will yield 0.5, suggesting moderate performance.
 - The harmonic mean will yield 0.18, indicating a significant issue in balancing between precision and recall.

F1 Scores

- F-1 Scores can be implemented in SKLearn as such:

```
from sklearn.metrics import f1_score
micro = f1_score(test_sent, y_pred, average='micro')
macro = f1_score(test_sent, y_pred, average='macro')
print('F1 Micro: ' + str(micro))
print('F1 Macro: ' + str(macro))
```

F1 Micro: 0.872

F1 Macro: 0.8718995692623017

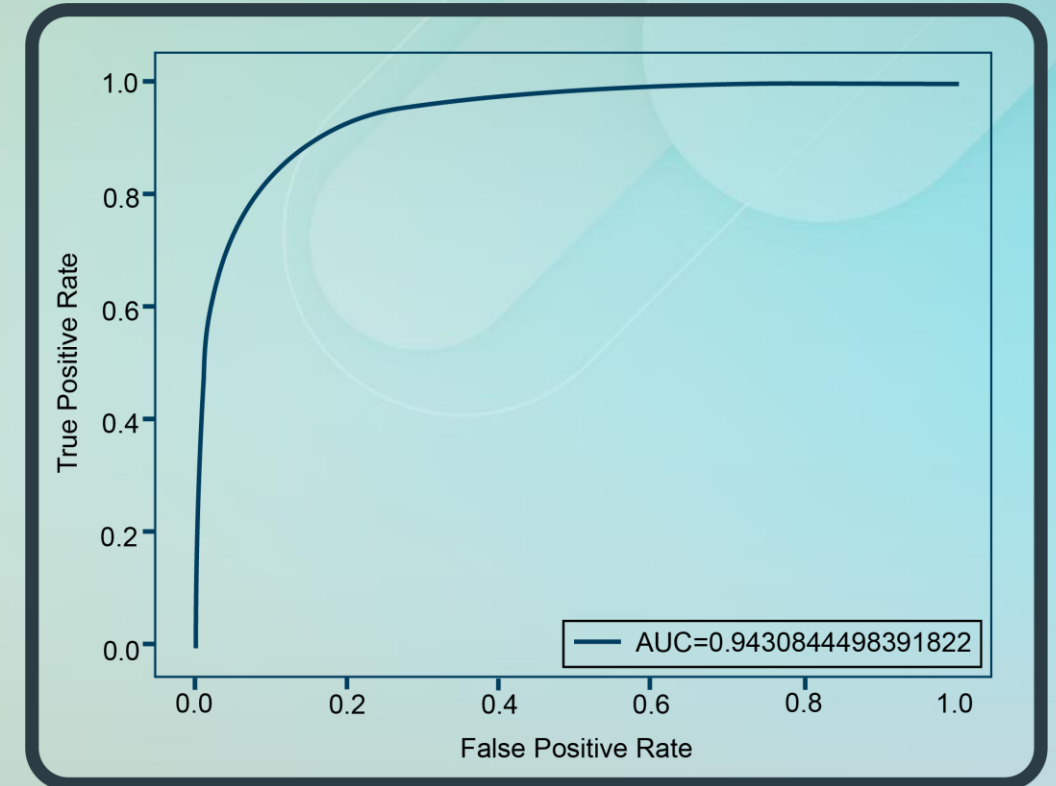
Note that our dataset has a balanced split in classes. Hence, the micro and macro F1 scores do not differ much.

Area Under Curve (AUC-ROC)

- The Receiver Operating Characteristic (ROC) curve is a graphical representation of a model's performance.
- The Area Under Curve of the ROC curve (AUC-ROC) is the most widely used metric in measuring the performance of binary classification models where the underlying model outputs are probabilistic values.
- For these models, classification is done by choosing a decision boundary (threshold value). For example, model outputs with a value ≥ 0.5 can be labeled to the class 1, and < 0.5 to the class 0.
- Different choices of decision boundary will give different True Positive Rate and False Positive Rate
- The ROC curve is created by plotting the True Positive Rate against the False Positive Rate as we vary the model decision boundary (threshold).
- The ROC curve reflects the trade-off between a model's ability to correctly identify positive instances (True Positives) and its tendency to incorrectly classify negative instances as positive (False Positives).

Area Under Curve (AUC-ROC)

- The Receiver Operating Characteristic (ROC) curve is a graphical representation of a model's performance.
- The AUC-ROC is a scalar value that represents the area **under the ROC curve**.
- Values range from **0-1**:
 - A model with an AUC-ROC value of **0.5** performs **no better than random chance**.
 - A model with an AUC-ROC value greater than **0.5** indicates a **better-than-random** classifier.
 - A model with an AUC-ROC value of **1** is a **perfect classifier**, meaning it has achieved perfect discrimination between the classes.



Area Under Curve (AUC-ROC)

- To generate a ROC curve, we can use the metrics library in SKLearn.

```
#define metrics
y_pred_proba = log_regression.predict_proba(test_review_tfidf)[::,1]
fpr, tpr, _ = metrics.roc_curve(test_sent, y_pred_proba)
auc = metrics.roc_auc_score(test_sent, y_pred_proba)

#create ROC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

BLEU

- Bilingual Evaluation Understudy (BLEU) evaluates the similarity between a target sentence and a generated sentence.
- BLEU counts the number of n-grams that appear in both the generated sentence and the target sentence.
- Common n-gram sizes used are 1, 2, 3, or 4.
- BLEU calculates the precision of matching n-grams.
- Mostly used in machine translation applications.

- Precision is defined as:

$$\textit{Precision} = \frac{\textit{Total number of correctly predicted } n\text{-grams in the target sentence}}{\textit{Total number of } n\text{-grams in the predicted sentence}}$$

We first focus on 1-grams (single words).

Predicted Sentence: He eats an apple.

Target Sentence: He ate an apple.

In this case, the precision for the predicted sentence is 3/4.

However, consider the following:

Predicted Sentence: He He He.

Target Sentence: He ate an apple.

Predicted Sentence: He He He eats tasty fruit.

Target Sentence: He ate an apple.

Target Sentence: He is eating a tasty apple.

To avoid repetitions, we use clipped precision instead of precision:

- We compare each word from the predicted sentence with all the target sentences. If the word has a match in any target sentence, it is considered correct.
- We limit the count for each correct word to the maximum number of times that that word occurs across all target sentences. This helps to avoid the repetition problem.

The clipped precision of our predicted sentence is $2/6$ (As the repeated “He”s are not counted.)

BLEU

BLEU calculated the precision n-gram scores for translated sentences.

Predicted Sentence: The guard arrived late because it was raining.

Target Sentence: The guard arrived late because of the rain.

Precision 1-gram:

Predicted Sentence: The guard arrived late because ~~it was raining.~~

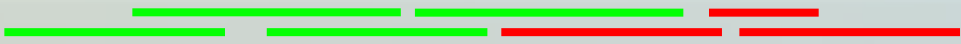


Target Sentence: The guard arrived late because of the rain.

Score: 5/8

Precision 2-gram:

Predicted Sentence: The guard arrived late because it was raining.



Target Sentence: The guard arrived late because of the rain.

Score: 4/7

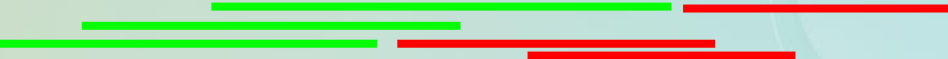
BLEU

BLEU calculated the precision n-gram scores for translated sentences.

Predicted Sentence: The guard arrived late because it was raining.

Target Sentence: The guard arrived late because of the rain.


Precision 3-gram:



Predicted Sentence: The guard arrived late because it was raining.

Target Sentence: The guard arrived late because of the rain.

Precision 4-gram (default):



Predicted Sentence: The guard arrived late because it was raining.

Target Sentence: The guard arrived late because of the rain.

The geometric average of the 4-gram is computed as such:

$$\begin{aligned} \textit{Geometric Average Precision} (N) &= \exp \left(\sum_{n=1}^N w_n \log p_n \right) \\ &= \prod_{n=1}^N p_n^{w_n} \\ &= (p_1)^{1/4} \cdot (p_2)^{1/4} \cdot (p_3)^{1/4} \cdot (p_4)^{1/4} \end{aligned}$$

A brevity penalty is then added to penalize sentences that are too short. (As shorter sentences can generate a misleading probability.)

$$\textit{Brevity Penalty} = \begin{cases} 1, & \text{if } c > r \\ e^{1-\frac{r}{c}}, & \text{if } c \leq r \end{cases}$$

Where: c = predicted length

r = target length

BLEU

The BLEU score is then given by:

$$BLEU = \text{Brevity Penalty} \cdot \text{Geometric Average Precision (N)}$$

Python implementation of BLEU:

```
import nltk
from nltk import word_tokenize
from nltk.translate.bleu_score import SmoothingFunction
ref = 'The guard arrived late because it was raining.'
cand = 'The guard arrived late because of the rain.'
smoother = SmoothingFunction().method1
reference = word_tokenize(ref)
candidate = word_tokenize(cand)
weights = (0.25, 0.25, 0.25, 0.25)
BLEUScore = nltk.translate.bleu_score.sentence_bleu([reference], candidate, weights, smoothing_function=smoother)
print(BLEUScore)
```

```
0.4671379777282001
```

ROUGE

- Recall-Oriented Understudy for Gisting Evaluation (ROUGE) is a set of metrics commonly used for text summarisation tasks.
- ROUGE scores are designed to assess the similarity and overlap between the words or phrases in the machine-generated text and the reference text.
- The general formula for ROUGE is given as:

$$ROUGE = \sum (Recall\ of\ n - grams)$$

- ROUGE is split into 3 types:
 - **ROUGE-N**: ROUGE-N measures the overlap of n-grams.
 - **ROUGE-L**: ROUGE-L measures the longest common subsequence (LCS) between the candidate text and the reference text.
 - **ROUGE-S**: ROUGE-S measures the skip-bigram (bi-gram with at most one intervening word) overlap between the candidate text and the reference text.

ROUGE-N

- ROUGE-N measures the overlap of n-grams (contiguous sequences of n words) between the candidate text and the reference text.
- Precision, recall, and F1-score are computed based on the n-gram overlap.
- Used to evaluate the grammatical correctness and fluency of generated text.
- Precision and Recall given by:

$$\text{Recall} = \frac{\text{Overlapping number of } n - \text{grams}}{\text{Number of } n - \text{grams in reference}}$$

$$\text{Precision} = \frac{\text{Overlapping number of } n - \text{grams}}{\text{Number of } n - \text{grams in candidate}}$$

ROUGE-L

- Measures the longest common subsequence (LCS) between the candidate text and the reference text.
- Precision, recall, and F1-score are computed based on the length of the LCS.
- Used to evaluate the semantic similarity and content coverage of generated text.
- Precision and Recall given by:

$$\textit{Recall} = \frac{\textit{Number of words in LCS}}{\textit{Number of words in reference}}$$

$$\textit{Precision} = \frac{\textit{Number of words in LCS}}{\textit{Number of words in candidate}}$$

ROUGE-S

- Measures the skip-bigram (bi-gram with at most one intervening word) overlap between the candidate text and the reference text.
- Precision, recall, and F1-score are computed based on the skip-bigram overlap.
- Used to evaluate the coherence and local cohesion of generated text.
- Precision and Recall given by:

$$\textit{Recall} = \frac{\textit{Number of skip – bigrams}}{\textit{Number of skip – bigrams in reference}}$$

$$\textit{Precision} = \frac{\textit{Number of skip – bigrams}}{\textit{Number of skip – bigrams in candidate}}$$

ROUGE

To implement the ROUGE score, we can use the evaluate library from HuggingFace.

```
import evaluate
rouge = evaluate.load('rouge')
predictions = ["Transformers Transformers are fast plus efficient",
               "Good Morning", "I am waiting for new Transformers"]
references = [
    ["HuggingFace Transformers are fast efficient plus awesome",
     "Transformers are awesome because they are fast to execute"],
    ["Good Morning Transformers", "Morning Transformers"],
    ["People are eagerly waiting for new Transformer models",
     "People are very excited about new Transformers"]
]
results = rouge.compute(predictions=predictions, references=references)
print(results)
```

```
{'rouge1': 0.6659340659340659, 'rouge2': 0.45454545454545453, 'rougeL': 0.6146520146520146, 'rougeLsum': 0.6146520146520146}
```

METEOR

- Metric for Evaluation of Translation with Explicit Ordering (METEOR) is used to assess the quality of machine translation systems.
- It complements other popular metrics like BLEU and ROUGE.
- METEOR takes into account the sequence of words in the output sentence.
- It considers the importance of word order in evaluating the translation quality.



METEOR

- To account for word order, a chunk penalty is included in the calculation of the METEOR metric.
- Intuitively, it represents the idea that a good translation should not only have words that are synonymous with the reference, but the words should also be in the correct order and grouped together in meaningful chunks.
- The chunk penalty is computed as:

$$p = 0.5 \left(\frac{c}{u_m} \right)^3$$

Where: c = the number of chunks in the candidate
 u_m = the number of unigrams in the candidate

- The final METEOR score is then given by: $M = F_{mean} (1 - p)$, where F_{mean} is a modified F1 score specifically used in METEOR.

METEOR

```
from nltk.translate import meteor
from nltk import word_tokenize
score=round(meteor([word_tokenize('The cat sat on the mat')],
                  word_tokenize('The cat was sat on the mat'))), 4)
print('The METEOR score is: '+str(score))
```

```
The METEOR score is: 0.9654
```

Word Embeddings



Word Embeddings

- To process textual data, we must convert raw text data into meaningful numerical representations.
- We have previously covered methods like BOW and TF-IDF.
- These methods however, have important drawbacks:
 - They have limited ability in capture semantic meaning of the words.
 - They are computationally inefficient and require high-dimensional presentation when the corpus is large.
- We will introduce two advanced embedding methods that significantly improves on these issues:
 - **Word2Vec**
 - A neural network approach to train word embeddings.
 - **GloVE**
 - A model using a global count-based matrix factorisation approach.

Word2Vec

- Word2Vec's key idea is that words that have similar meanings or are used in similar contexts should have similar vector representations.
- This enables Word2Vec to capture the semantic relationships between words. For example, it can represent that "king" is to "queen" as "man" is to "woman".
- Word2Vec includes two main models:
 - Continuous Bag of Words (CBOW):
 - Aims to predict a target word based on its context words.
 - Skip-gram:
 - Predicts context words given a target word.

Word2Vec - CBOW

Goal: Predict a target word based on context words surrounding the target word.

For example, to train the CBOW model, we can look at the 2 words before and after a target word:

Jay was hit by a _____ in...

by	a	red	bus	in
----	---	-----	-----	----

Here, the model aims to predict the word 'red' given the 4 surrounding context words:

Input 1	Input 2	Input 3	Input 4	output
by	a	bus	in	red

Word2Vec – Skip-gram

Goal: Predict context words from a target word.

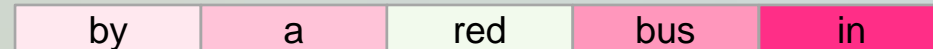
For example, train the Skip-gram model, we can look at the 2 words before and after a target word:

Jay was hit **by a red bus in** ...



Here, the model aims to predict the context words ‘by’, ‘a’, ‘bus’, ‘in’ given the word target ‘red’:

Jay was hit **by a red bus in** ...



input	output
red	by
red	a
red	bus
red	in

Word2Vec – Skip-gram

Thou shalt not make **a machine in the likeness** of a human mind

thou	shall	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

thou	shall	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

thou	shall	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

thou	shall	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

thou	shall	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

input word	output word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine
a	not
a	make
a	machine
a	in
machine	make
machine	a
machine	in
machine	the
in	a
in	machine
in	the
in	likeness

More data is generated using the same sliding window in the Skip-gram model.

CBOW

- CBOW is computationally more efficient and often trains faster than Skip-gram.
- A good choice for smaller datasets or when you want to quickly generate word embeddings.
- It can be more effective when the context window size is relatively small, as it directly predicts the target word based on nearby words.

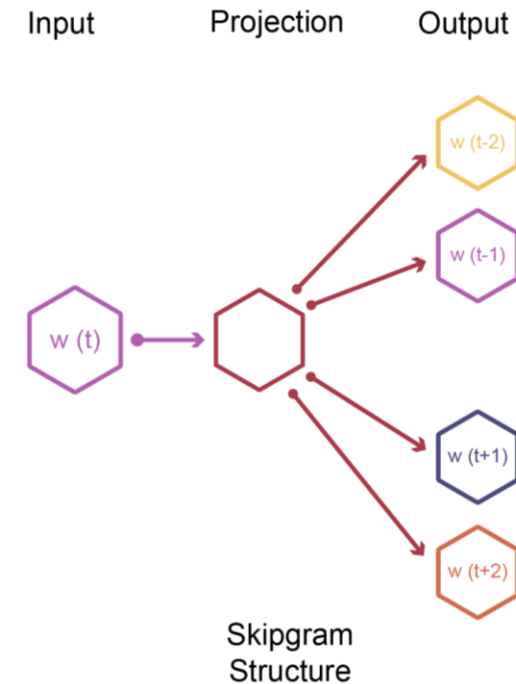
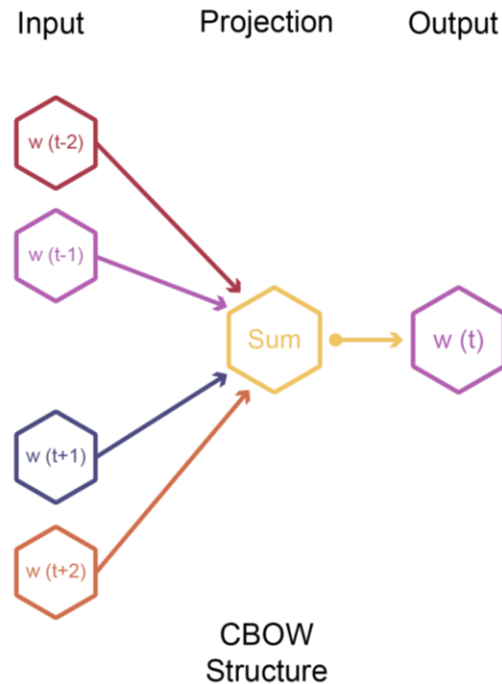
Skip-gram

- Skip-gram is often preferred when you have a large dataset with a rich vocabulary and you want to capture semantic relationships between words effectively.
- It can capture rare words and infrequent word associations better than CBOW.

The Skip-gram model is preferred when training on large datasets.

Word2Vec

- The Word2Vec model is based on a shallow neural network consisting of an input layer, a densely connected hidden layer and an output layer.

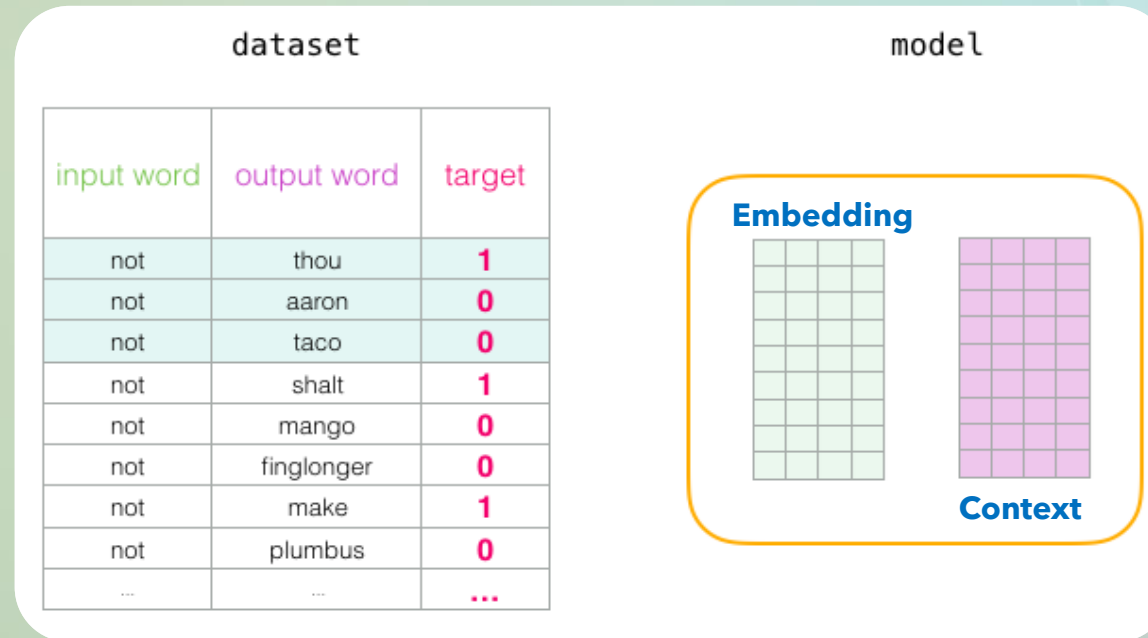


Word2Vec

- Word2Vec is trained iteratively as follows:
- Given a large text corpus;
- Go over the text with a sliding window, moving one word at a time.
- At each step, there is a target word and surrounding context words;
- In the Skip-gram setup, compute prediction probabilities of context words based on the target word;
- These predictions are updated through standard neural network iterations.

Word2Vec - Training

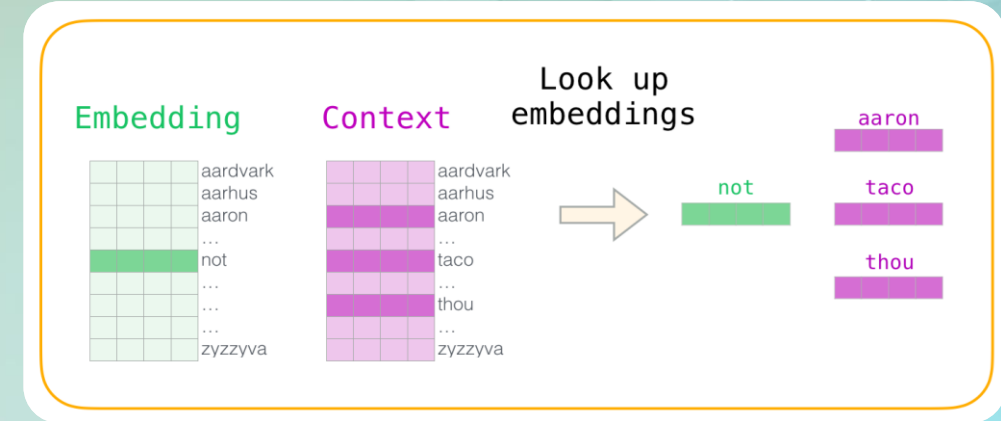
- Create 2 matrices, the Embedding and Context Matrix.
- Initialise matrix with random numbers.
- In each training step, we take one positive example and its associated negative examples.



Subtitle here

Word2Vec - Training

- Input words are found in the embedding matrix.
- We find the corresponding output words in the context matrix.
- Calculate similarity between input and output words using dot product.
- Pass it through a sigmoid function to convert it into probabilities.
- Calculate loss function.









input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	-1.11	0.25	-0.25
not	taco	0	0.74	0.68	-0.68

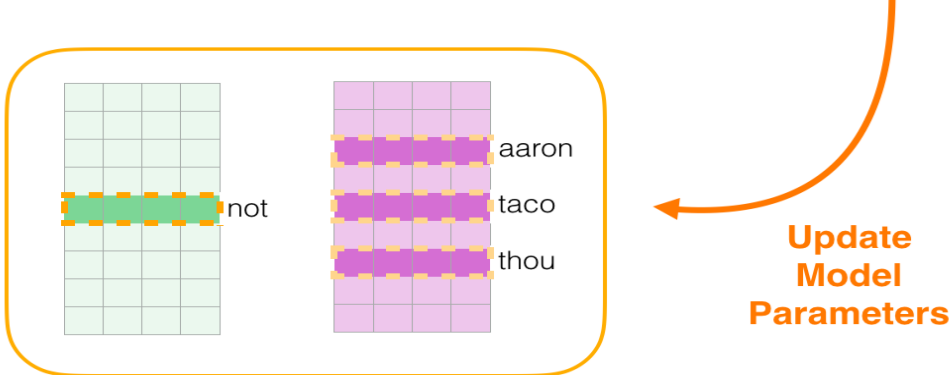
$$\text{error} = \text{target} - \text{sigmoid_scores}$$

Subtitle here

Word2Vec - Training

- The error score is used to adjust the embeddings of output words.
- The next time this calculation is made, the result would be closer to the target scores.
- Iteratively perform this through every input word.
- After training, the context embeddings are discarded, leaving us with the final embedding vector.

input word	output word	target	input • output	sigmoid()	Error
not 	thou 	1	0.2	0.55	0.45
not 	aaron 	0	-1.11	0.25	-0.25
not 	taco 	0	0.74	0.68	-0.68



Subtitle here

Word2Vec

```
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize

# Sample sentences for training Word2Vec
sentences = [
    "Word2Vec is a technique for word embedding.",
    "Embedding words in vector space is powerful for NLP.",
    "Gensim provides an easy way to work with Word2Vec.",
]

# Tokenize the sentences into words
tokenized_sentences = [word_tokenize(sentence.lower()) for sentence in sentences]

# Train a Word2Vec model
model = Word2Vec(tokenized_sentences, vector_size=100, window=5, min_count=1, sg=0)
# Adjust parameters as needed

# Save the trained model for future use
model.save("word2vec.model")
```

Word2Vec

```
# Get the word embeddings
word = "word"
if word in model.wv:
    embedding = model.wv[word]
    print(f"Embedding for '{word}': {embedding}")
else:
    print(f"'{word}' is not in the vocabulary.")

# Similarity between words
similarity = model.wv.similarity("word", "embedding")
print(f"Similarity between 'word' and 'embedding': {similarity}")
```

```
Embedding for 'word': [-9.5782708e-03  8.9433035e-03  4.1655651e-03  9.2360200e-03
 6.6434164e-03  2.9238823e-03  9.8045552e-03 -4.4228774e-03
-6.8031684e-03  4.2263791e-03  3.7309569e-03 -5.6647998e-03
 9.7051831e-03 -3.5571249e-03  9.5494138e-03  8.3399686e-04
-6.3375360e-03 -1.9765303e-03 -7.3783435e-03 -2.9803123e-03
 1.0425968e-03  9.4828764e-03  9.3571255e-03 -6.5945145e-03
 3.4750504e-03  2.2753996e-03 -2.4888995e-03 -9.2287343e-03
 1.0264782e-03 -8.1667695e-03  6.3202251e-03 -5.8006351e-03
 5.5357707e-03  9.8335072e-03 -1.6020649e-04  4.5292759e-03
-1.8079536e-03  7.3599042e-03  3.9389166e-03 -9.0105701e-03
-2.3983277e-03  3.6285119e-03 -9.9961828e-05 -1.2015507e-03
-1.0550468e-03 -1.6715144e-03  6.0320512e-04  4.1645542e-03
-4.2527826e-03 -3.8329726e-03 -5.2653751e-05  2.6851249e-04
-1.7041666e-04 -4.7857561e-03  4.3126042e-03 -2.1723476e-03
 2.1040821e-03  6.6577346e-04  5.9687104e-03 -6.8427366e-03
-6.8160188e-03 -4.4750450e-03  9.4369482e-03 -1.5926627e-03
-9.4300918e-03 -5.4425694e-04 -4.4496474e-03  6.0008741e-03
-9.5837293e-03  2.8578769e-03 -9.2512975e-03  1.2507273e-03
 5.9983991e-03  7.3978822e-03 -7.6199183e-03 -6.0532107e-03
-6.8376604e-03 -7.9174070e-03 -9.5002521e-03 -2.1253186e-03
-8.3641807e-04 -7.2558355e-03  6.7871362e-03  1.1204430e-03
 5.8272551e-03  1.4734893e-03  7.9107360e-04 -7.3686223e-03
-2.1775942e-03  4.3218113e-03 -5.0858771e-03  1.1309833e-03
 2.8836925e-03 -1.5359134e-03  9.9341124e-03  8.3504440e-03
 2.4163835e-03  7.1164975e-03  5.8903527e-03 -5.5806041e-03]
Similarity between 'word' and 'embedding': -0.06901167333126068
```

Word2Vec – Probability

- For the Word2Vec model, the objective is to maximise the average log-probability of the context words occurring around the input word over the entire vocabulary.

$$\frac{1}{T} \sum_{t=1}^T \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} | w_t)$$

Where T is all the words in the training data and c is the training context window

- One way to calculate the above probability is to use the SoftMax function:

$$p(w_o | w_I) = \frac{\exp(v'_{w_o} v_{w_I})}{\sum_{w=1}^W \exp(v'_{w_o} v_{w_I})}$$

Where v_w and v'_w are the vector representations of the word w as the input and output, respectively. Also, W is the number of words in the entire vocabulary.

Word2Vec – SoftMax

$$p(w_o|w_I) = \frac{\exp(v'_{w_o} v_{w_I})}{\sum_{w=1}^W \exp(v'_{w_o} v_{w_I})}$$

Where v_w and v'_w are the vector representations of the word w as the input and output, respectively. Also, W is the number of words in the entire vocabulary.

- The intuition is that words that appear in the same context will have similar vector representations.
- The numerator in the equation will show this by assigning a larger value for similar words through the dot product of the two vectors.
- However, the denominator, which is a normalizing factor that has to be computed over the entire vocabulary, is extremely difficult to compute for large vocabularies.

Word2Vec – Negative Sampling

- Negative sampling is a workaround that aims at maximising the similarity of the words in the same context and minimising it when they occur in different contexts.
- Instead of doing the minimisation for all the words in the dictionary except for the context words, it randomly selects a handful of words ($2 \leq k \leq 20$) depending on the training size and uses them to optimize the objective.
- A larger k is chosen for smaller datasets and vice versa

$$\log \sigma(v'_{w_O}^T v_{w_I}) + \sum_{i=1}^k \mathbb{E}_{w_i \sim P_n(w)} [\log \sigma(-v'_{w_i}^T v_{w_I})]$$

Where σ is the sigmoid function and $P_n(w)$ is the noise distribution with the negative samples drawn from it. It's calculated as the unigram distribution of the words to the power of $3/4$.

$$P_n(w) = \frac{U(w)^{\frac{3}{4}}}{Z}$$

Where Z is a normalisation constant.

- Global Vectors for Word Representation (GloVe) is an unsupervised machine learning algorithm used for generating word embeddings.
- Designed to capture the global co-occurrence statistics of words from a large corpus of text.
- It counts the frequency of each word appearing in the context of every other word in a fixed window size.
- A co-occurrence matrix X is constructed, where a cell X_{ij} is a “strength” which represents how often the word i appears in the context of the word j .
- In some sense, GloVe goes beyond Word2Vec, but not only considering local context, but also aggregating the results to a global count.

- For each pair of words, i, j , we can build a cost (loss) term as follows:

$$w_i^T w_j + b_i + b_j - \log(X_{ij})$$

where b_i and b_j are scalar bias terms associated with words i and j , respectively.

- To generate these vectors, we minimise an objective function, J which evaluates the sum of all squared errors based on the above equation, weighted with a function f :

$$J = \sum_{i,j=1}^V f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

Where V is the size of the vocabulary.

- The function f is used to prevent the model from being overly influenced by very common word pairs.
- A typical form of the function f is as follows.

$$f(x) = \begin{cases} \left(\frac{x}{x_{max}}\right)^\alpha & \text{if } x < x_{max} \\ 1 & \text{otherwise} \end{cases}$$

```
import gensim.downloader as api

# Load the pre-trained GloVe model (you may need to download it first)
glove_model = api.load("glove-wiki-gigaword-100")

# Find the embedding for a specific word
word = "nero"
try:
    embedding = glove_model[word]
    print(f"Embedding for '{word}':")
    print(embedding)
except KeyError:
    print(f"'{word}' is not in the vocabulary.")

# Find the most similar words to a given word
similar_words = glove_model.most_similar(word)
print(f"\nWords most similar to '{word}':")
for similar_word, score in similar_words:
    print(similar_word, score)
```

Embedding for 'nero':

```
[ 0.12703 -1.2517 -0.30333 -0.24548 0.068448 1.1062
 0.30107 -0.33641 0.067003 -0.71713 -0.045462 0.38737
-0.53082 0.19166 -0.13454 -0.69842 0.43061 -0.1036
 0.33654 0.74012 0.026213 -0.52291 -0.3713 1.0698
 0.61742 0.27974 0.66584 -0.11241 0.0066852 0.1599
 0.16144 0.29823 -0.079172 0.1576 0.20478 1.1741
 0.24823 -0.60665 0.83795 0.44143 -0.31444 0.0040575
 0.19608 0.36953 0.16881 -0.081036 -0.23726 -0.82465
-0.025315 0.44143 -0.18847 0.2841 0.034934 0.41033
-0.73196 -0.41349 -0.72422 0.31519 -1.13 -0.2272
 0.080723 0.50456 0.014441 0.14606 -0.15919 -0.54273
-0.43092 -0.37986 -0.35882 -0.037606 0.51308 -0.2918
 0.46491 0.41077 -0.20835 -1.2477 -0.77411 0.053342
-0.57954 0.58317 0.56071 -0.16284 -0.3974 0.77043
-0.58452 0.7157 -0.89662 -0.60718 -0.26547 0.11487
 0.48213 -0.66232 0.23707 -0.38942 0.14209 -0.90944
-0.061628 -0.58932 -0.76396 0.10286 ]
```

Words most similar to 'nero':

```
claudius 0.7039013504981995
caligula 0.6931164860725403
tiberius 0.624136209487915
caesar 0.6094770431518555
domitian 0.5942543745040894
vespasian 0.5886495113372803
germanicus 0.5798532962799072
drusus 0.5789514780044556
cleopatra 0.5602390766143799
galba 0.5509929060935974
```

Evaluation Metrics:

Confusion Matrix:

A table used in machine learning that summarises the performance of a classification model by comparing actual and predicted values, showing counts of true positives, true negatives, false positives, and false negatives.

F1 Scores:

A single metric that combines precision and recall into a single value, providing a balanced measure of a model's accuracy in binary classification tasks.

AUC-ROC:

A metric used to evaluate the performance of a binary classification model by measuring the area under the Receiver Operating Characteristic (ROC) curve, reflecting the model's ability to distinguish between classes.

Evaluation Metrics:

BLEU:

A metric for evaluating the quality of machine-generated text by comparing it to human-generated reference text based on n-gram overlap and precision.

ROUGE:

A set of metrics used for evaluating the quality of machine-generated text by measuring the overlap of n-grams and other text units between the generated text and reference text.

METEOR:

A metric that emphasizes on the importance of word order in machine translations.

Word Embeddings:

Word2Vec:

A popular word embedding technique that learns dense vector representations of words by predicting words in their context, capturing semantic relationships between words in a continuous vector space.

GloVe:

An unsupervised word embedding algorithm that captures global co-occurrence statistics of words from a large text corpus to create vector representations that encode semantic meaning and relationships between words.

No part of this video shall be filmed, recorded, downloaded, reproduced, distributed, republished or transmitted in any form or by any means without written approval from the University.