OM CHOKSI

🎯 Learning Objectives

> - Document Classification
> - Text Pre-processing
> - Feature extraction
> - Vocabulary creation

📋 Vocabulary & Feature Extraction

> Given a document, you can represent it as a vector of dimension V, where V corresponds to your vocabulary size. As V gets larger, the vector becomes more sparse. Furthermore, we end up having many more features and end up training lot of parameters. This could result in larger training time, and large prediction time.

🔨 Preprocessing

When preprocessing, you have to perform the following:

> 1. Eliminate handles and URLs
> 2. Tokenize the string into words
> 3. Remove stop words like "and, is, a, on, etc."
> 4. Stemming - or convert every word to its stem. Like dancer, dancing, danced, becomes 'danc'.
> 5. Convert all your words to lower case.

📌 Cell 1: Imports & Setup

In [1]:
```
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.datasets import fetch_20newsgroups
from sklearn.decomposition import TruncatedSVD
```

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score, classification_report, confusion_m
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import Normalizer
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

import tensorflow as tf
from tensorflow.keras import layers
import tensorflow_datasets as tfds
```

```
2025-11-17 19:04:48.094720: E external/local_xla/xla/stream_executor/cuda/cud
a_fft.cc:477] Unable to register cuFFT factory: Attempting to register factory
for plugin cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
E0000 00:00:1763406288.621020      48 cuda_dnn.cc:8310] Unable to register cuDN
N factory: Attempting to register factory for plugin cuDNN when one has already
been registered
E0000 00:00:1763406288.734810      48 cuda_blas.cc:1418] Unable to register cuB
LAS factory: Attempting to register factory for plugin cuBLAS when one has alre
ady been registered
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
AttributeError: 'MessageFactory' object has no attribute 'GetPrototype'
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
AttributeError: 'MessageFactory' object has no attribute 'GetPrototype'
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
AttributeError: 'MessageFactory' object has no attribute 'GetPrototype'
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
AttributeError: 'MessageFactory' object has no attribute 'GetPrototype'
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
AttributeError: 'MessageFactory' object has no attribute 'GetPrototype'
```

Cell 2: Load Data

In [2]:

```python
categories = ['alt.atheism', 'talk.religion.misc', 'comp.graphics', 'sci.space

data_train = fetch_20newsgroups(subset='train', categories=categories, shuffle
data_test = fetch_20newsgroups(subset='test', categories=categories, shuffle=T

print("Train samples:", len(data_train.data))
print("Test samples:", len(data_test.data))

y_train, y_test = data_train.target, data_test.target
target_names = data_train.target_names
```

```
Train samples: 2034
Test samples: 1353
```

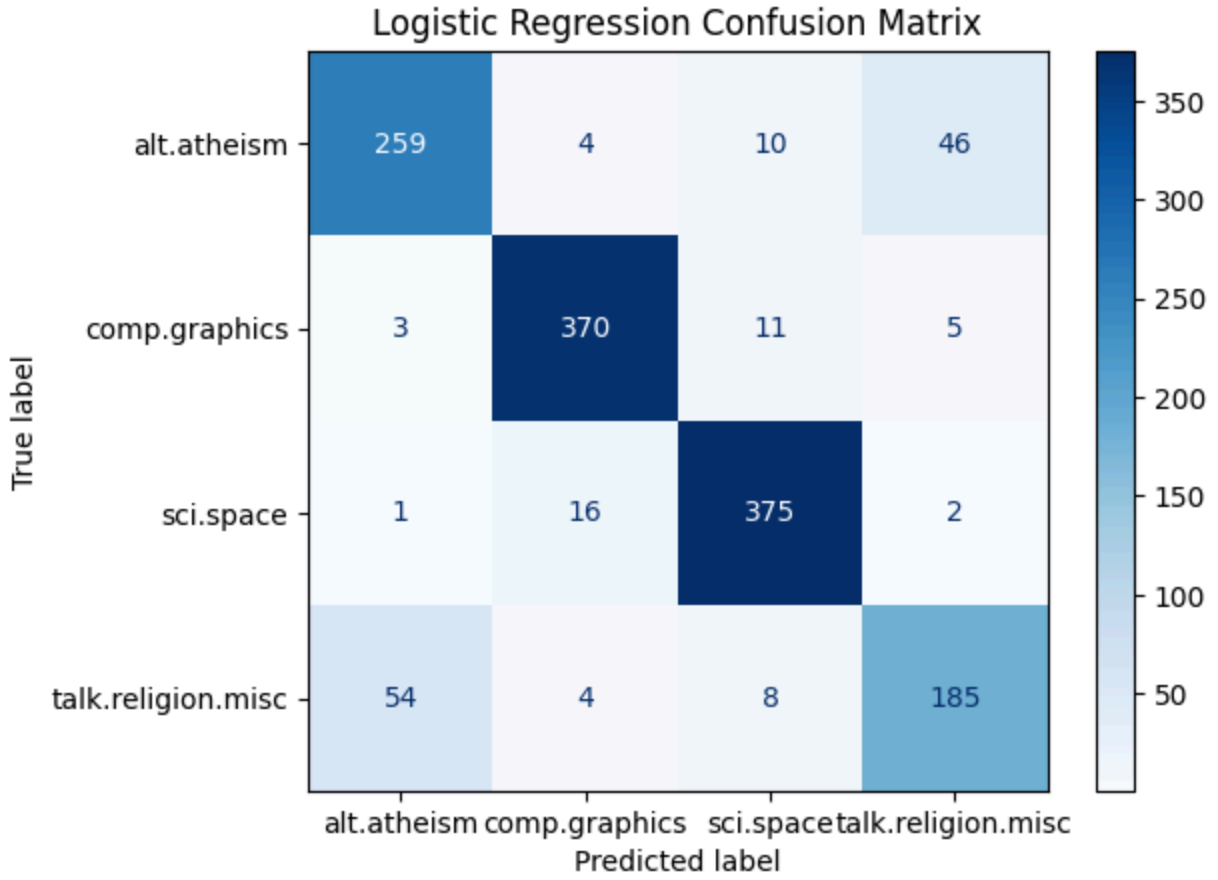
Cell 3: Feature Extraction

In [3]:
```
vectorizer = TfidfVectorizer(max_df=0.5, min_df=2, stop_words='english')
X_train = vectorizer.fit_transform(data_train.data)
X_test = vectorizer.transform(data_test.data)

# Optional dimensionality reduction
n_components = 100
svd = TruncatedSVD(n_components)
normalizer = Normalizer(copy=False)
lsa = make_pipeline(svd, normalizer)

X_train = lsa.fit_transform(X_train)
X_test = lsa.transform(X_test)
```

Cell 4: Logistic Regression

In [4]:
```
lr_clf = LogisticRegression(max_iter=1000)
lr_clf.fit(X_train, y_train)

lr_train_pred = lr_clf.predict(X_train)
lr_test_pred = lr_clf.predict(X_test)

print("Logistic Regression Train Accuracy:", accuracy_score(y_train, lr_train_
print("Logistic Regression Test Accuracy:", accuracy_score(y_test, lr_test_pre

ConfusionMatrixDisplay(confusion_matrix(y_test, lr_test_pred), display_labels=
plt.title("Logistic Regression Confusion Matrix")
plt.show()

print(pd.DataFrame(classification_report(y_test, lr_test_pred, output_dict=Tru
```

```
Logistic Regression Train Accuracy: 0.95968534906588
Logistic Regression Test Accuracy: 0.8787878787878788
```

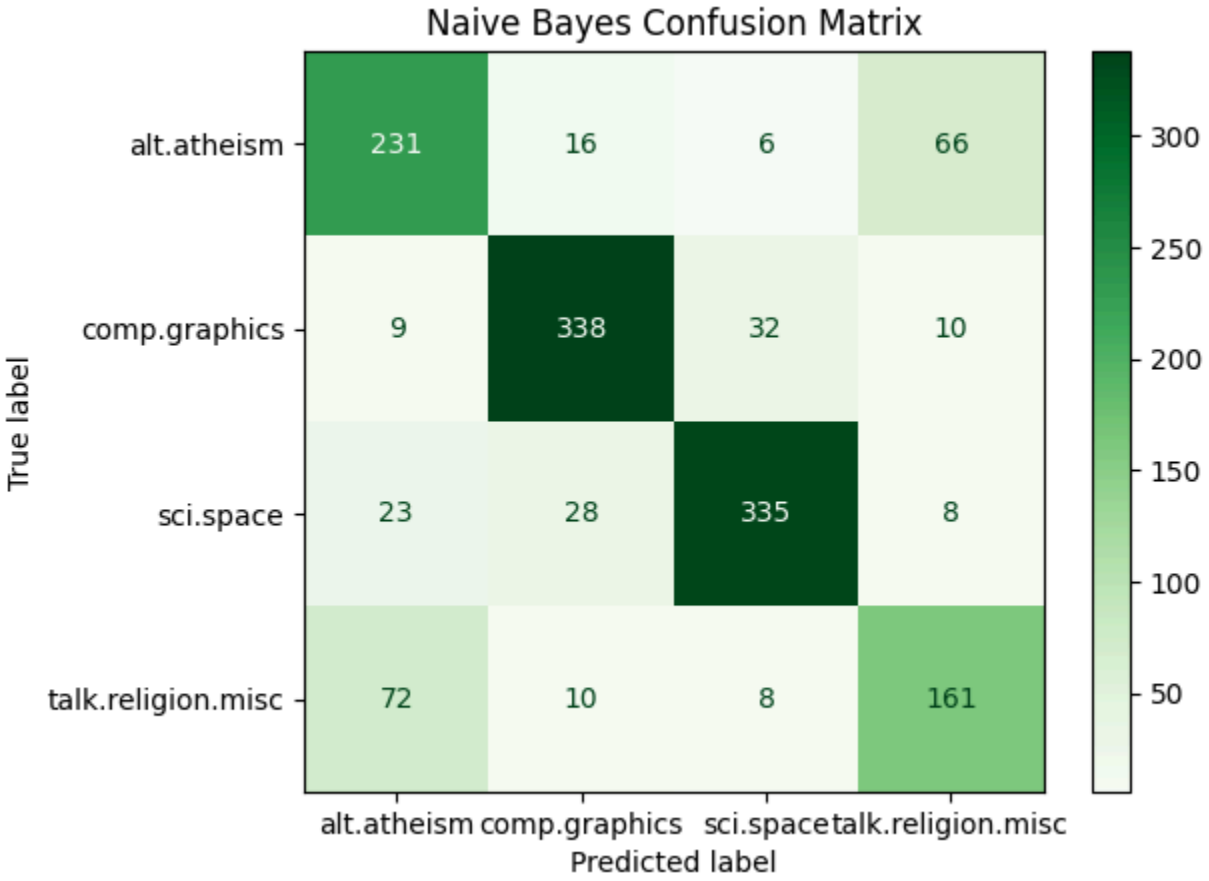


	precision	recall	f1-score	support
0	0.817035	0.811912	0.814465	319.000000
1	0.939086	0.951157	0.945083	389.000000
2	0.928218	0.951777	0.939850	394.000000
3	0.777311	0.737052	0.756646	251.000000
accuracy	0.878788	0.878788	0.878788	0.878788
macro avg	0.865412	0.862974	0.864011	1353.000000
weighted avg	0.877133	0.878788	0.877805	1353.000000

Cell 5: Naive Bayes

In [5]:
```
nb_clf = GaussianNB()
nb_clf.fit(X_train, y_train)

nb_train_pred = nb_clf.predict(X_train)
nb_test_pred = nb_clf.predict(X_test)

print("Naive Bayes Train Accuracy:", accuracy_score(y_train, nb_train_pred))
print("Naive Bayes Test Accuracy:", accuracy_score(y_test, nb_test_pred))

ConfusionMatrixDisplay(confusion_matrix(y_test, nb_test_pred), display_labels=
plt.title("Naive Bayes Confusion Matrix")
plt.show()

print(pd.DataFrame(classification_report(y_test, nb_test_pred, output_dict=Tru
```

```
Naive Bayes Train Accuracy: 0.8593903638151426
Naive Bayes Test Accuracy: 0.7871396895787139
```



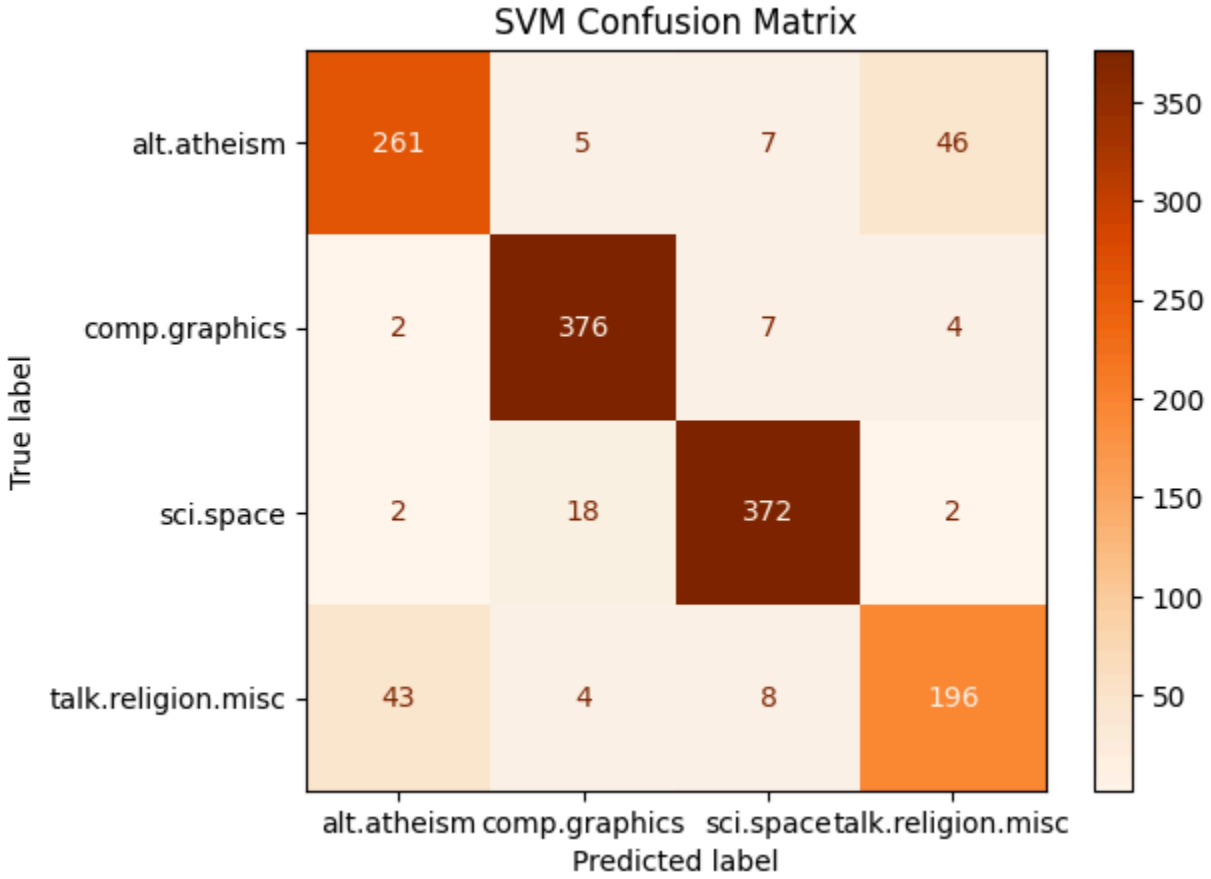
	precision	recall	f1-score	support
0	0.689552	0.724138	0.706422	319.00000
1	0.862245	0.868895	0.865557	389.00000
2	0.879265	0.850254	0.864516	394.00000
3	0.657143	0.641434	0.649194	251.00000
accuracy	0.787140	0.787140	0.787140	0.78714
macro avg	0.772051	0.771180	0.771422	1353.00000
weighted avg	0.788436	0.787140	0.787596	1353.00000

Cell 6: Support Vector Machine

In [6]:
```
svm_clf = SVC()
svm_clf.fit(X_train, y_train)

svm_train_pred = svm_clf.predict(X_train)
svm_test_pred = svm_clf.predict(X_test)

print("SVM Train Accuracy:", accuracy_score(y_train, svm_train_pred))
print("SVM Test Accuracy:", accuracy_score(y_test, svm_test_pred))

ConfusionMatrixDisplay(confusion_matrix(y_test, svm_test_pred), display_labels
plt.title("SVM Confusion Matrix")
plt.show()
```

```
print(pd.DataFrame(classification_report(y_test, svm_test_pred, output_dict=Tr
```

```
SVM Train Accuracy: 0.9837758112094396
SVM Test Accuracy: 0.8906134515890614
```



	precision	recall	f1-score	support
0	0.847403	0.818182	0.832536	319.000000
1	0.933002	0.966581	0.949495	389.000000
2	0.944162	0.944162	0.944162	394.000000
3	0.790323	0.780876	0.785571	251.000000
accuracy	0.890613	0.890613	0.890613	0.890613
macro avg	0.878723	0.877450	0.877941	1353.000000
weighted avg	0.889601	0.890613	0.889956	1353.000000

Cell 7: Deep Learning CNN

In [7]:

```
tokenizer = tfds.deprecated.text.SubwordTextEncoder.build_from_corpus(
    data_train.data, target_vocab_size=2**15
)

train_inputs = [tokenizer.encode(text) for text in data_train.data]
test_inputs = [tokenizer.encode(text) for text in data_test.data]

MAX_LEN = max(len(x) for x in train_inputs)

train_inputs = tf.keras.preprocessing.sequence.pad_sequences(train_inputs, val
```

```
test_inputs = tf.keras.preprocessing.sequence.pad_sequences(test_inputs, value

class DCNN(tf.keras.Model):
    def __init__(self, vocab_size, emb_dim=128, nb_filters=50, FFN_units=512,
        super(DCNN, self).__init__()
        self.embedding = layers.Embedding(vocab_size, emb_dim)
        self.bigram = layers.Conv1D(filters=nb_filters, kernel_size=2, activat
        self.trigram = layers.Conv1D(filters=nb_filters, kernel_size=3, activa
        self.fourgram = layers.Conv1D(filters=nb_filters, kernel_size=4, activ
        self.pool = layers.GlobalMaxPool1D()
        self.dense_1 = layers.Dense(units=FFN_units, activation="relu")
        self.dropout = layers.Dropout(rate=dropout_rate)
        self.last_dense = layers.Dense(units=nb_classes, activation="softmax")

    def call(self, inputs, training=False):
        x = self.embedding(inputs)
        x_1 = self.pool(self.bigram(x))
        x_2 = self.pool(self.trigram(x))
        x_3 = self.pool(self.fourgram(x))
        merged = tf.concat([x_1, x_2, x_3], axis=-1)
        merged = self.dense_1(merged)
        merged = self.dropout(merged, training=training)
        return self.last_dense(merged)

VOCAB_SIZE = tokenizer.vocab_size
Dcnn = DCNN(vocab_size=VOCAB_SIZE, emb_dim=128, nb_filters=100, FFN_units=256,

Dcnn.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics

history = Dcnn.fit(train_inputs, y_train, batch_size=16, epochs=10, validation
```

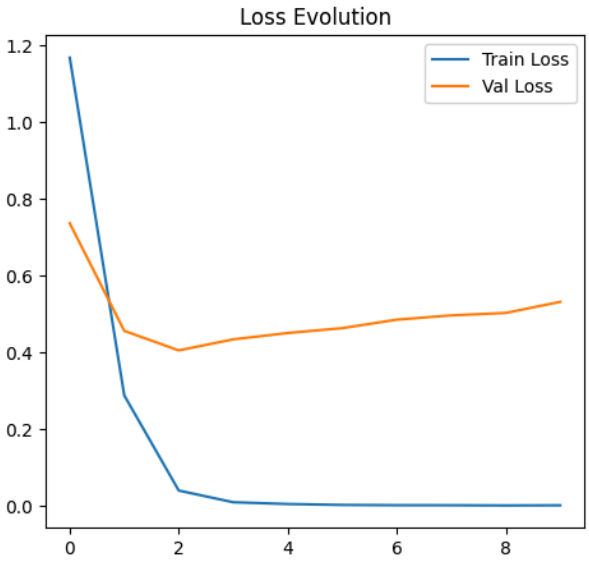
```
I0000 00:00:1763406543.139391      48 gpu_device.cc:2022] Created device /job:l
ocalhost/replica:0/task:0/device:GPU:0 with 13942 MB memory:  -> device: 0, nam
e: Tesla T4, pci bus id: 0000:00:04.0, compute capability: 7.5
I0000 00:00:1763406543.140068      48 gpu_device.cc:2022] Created device /job:l
ocalhost/replica:0/task:0/device:GPU:1 with 13942 MB memory:  -> device: 1, nam
e: Tesla T4, pci bus id: 0000:00:05.0, compute capability: 7.5
Epoch 1/10
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
I0000 00:00:1763406547.051772     194 service.cc:148] XLA service 0x7c071000954
0 initialized for platform CUDA (this does not guarantee that XLA will be use
d). Devices:
I0000 00:00:1763406547.053060     194 service.cc:156]   StreamExecutor device
(0): Tesla T4, Compute Capability 7.5
I0000 00:00:1763406547.053076     194 service.cc:156]   StreamExecutor device
(1): Tesla T4, Compute Capability 7.5
I0000 00:00:1763406547.446870     194 cuda_dnn.cc:529] Loaded cuDNN version 903
00
  1/128 ━━━━━━━━━━━━━━━━━━━━ 20:51 10s/step - loss: 1.4164 - sparse_categorica
l_accuracy: 0.1250
I0000 00:00:1763406553.156719     194 device_compiler.h:188] Compiled cluster u
sing XLA!  This line is logged at most once for the lifetime of the process.
```

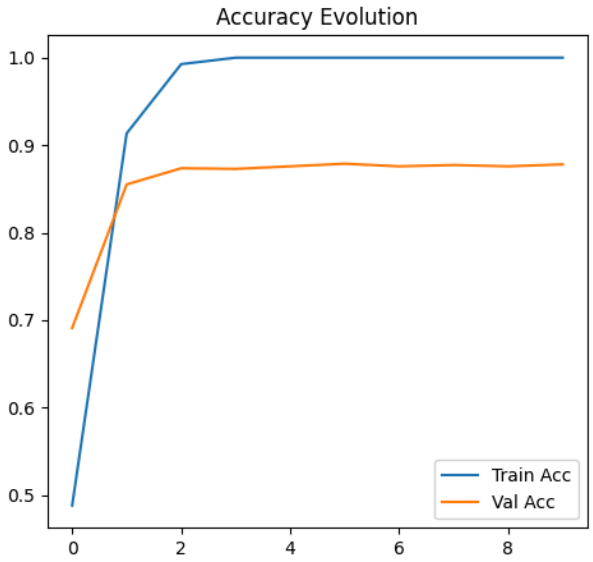
```
128/128 ━━━━━━━━━━━━━━━━━━━━ 32s 173ms/step - loss: 1.3169 - sparse_categorica
l_accuracy: 0.3599 - val_loss: 0.7359 - val_sparse_categorical_accuracy: 0.6911
Epoch 2/10
128/128 ━━━━━━━━━━━━━━━━━━━━ 17s 134ms/step - loss: 0.3814 - sparse_categorica
l_accuracy: 0.8751 - val_loss: 0.4556 - val_sparse_categorical_accuracy: 0.8551
Epoch 3/10
128/128 ━━━━━━━━━━━━━━━━━━━━ 17s 136ms/step - loss: 0.0463 - sparse_categorica
l_accuracy: 0.9949 - val_loss: 0.4047 - val_sparse_categorical_accuracy: 0.8736
Epoch 4/10
128/128 ━━━━━━━━━━━━━━━━━━━━ 18s 139ms/step - loss: 0.0108 - sparse_categorica
l_accuracy: 1.0000 - val_loss: 0.4335 - val_sparse_categorical_accuracy: 0.8729
Epoch 5/10
128/128 ━━━━━━━━━━━━━━━━━━━━ 19s 145ms/step - loss: 0.0050 - sparse_categorica
l_accuracy: 1.0000 - val_loss: 0.4500 - val_sparse_categorical_accuracy: 0.8758
Epoch 6/10
128/128 ━━━━━━━━━━━━━━━━━━━━ 19s 147ms/step - loss: 0.0026 - sparse_categorica
l_accuracy: 1.0000 - val_loss: 0.4627 - val_sparse_categorical_accuracy: 0.8788
Epoch 7/10
128/128 ━━━━━━━━━━━━━━━━━━━━ 18s 143ms/step - loss: 0.0015 - sparse_categorica
l_accuracy: 1.0000 - val_loss: 0.4848 - val_sparse_categorical_accuracy: 0.8758
Epoch 8/10
128/128 ━━━━━━━━━━━━━━━━━━━━ 18s 142ms/step - loss: 0.0015 - sparse_categorica
l_accuracy: 1.0000 - val_loss: 0.4959 - val_sparse_categorical_accuracy: 0.8773
Epoch 9/10
128/128 ━━━━━━━━━━━━━━━━━━━━ 18s 143ms/step - loss: 7.1925e-04 - sparse_categor
ical_accuracy: 1.0000 - val_loss: 0.5023 - val_sparse_categorical_accuracy: 0.8
758
Epoch 10/10
128/128 ━━━━━━━━━━━━━━━━━━━━ 18s 144ms/step - loss: 0.0012 - sparse_categorica
l_accuracy: 1.0000 - val_loss: 0.5312 - val_sparse_categorical_accuracy: 0.8780
```

Cell 8: Plot Training Curves

In [8]:

```
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.legend(); plt.title("Loss Evolution")

plt.subplot(1,2,2)
plt.plot(history.history['sparse_categorical_accuracy'], label='Train Acc')
plt.plot(history.history['val_sparse_categorical_accuracy'], label='Val Acc')
plt.legend(); plt.title("Accuracy Evolution")
plt.show()
```





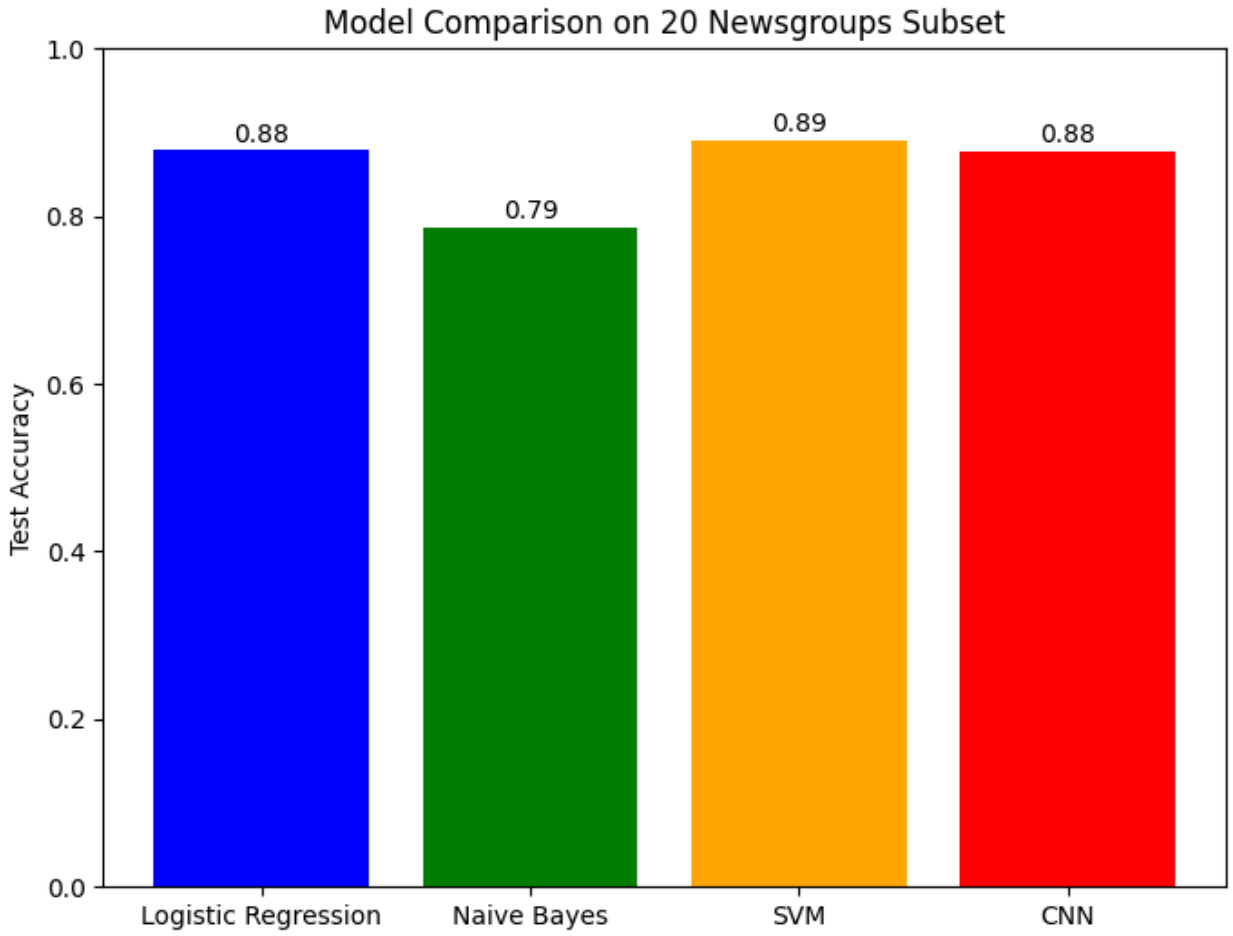
Cell 9: Final Evaluation

In [9]:
```
results = Dcnn.evaluate(test_inputs, y_test, batch_size=16)
print("Final Test Loss:", results[0])
print("Final Test Accuracy:", results[1])
```

```
85/85 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step - loss: 0.4986 - sparse_categorical_acc
uracy: 0.8763
Final Test Loss: 0.5311814546585083
Final Test Accuracy: 0.8780487775802612
```

Cell 10: Comparison Bar Chart

In [10]:
```
results_dict = {
    "Logistic Regression": accuracy_score(y_test, lr_test_pred),
    "Naive Bayes": accuracy_score(y_test, nb_test_pred),
    "SVM": accuracy_score(y_test, svm_test_pred),
    "CNN": results[1]
}

plt.figure(figsize=(8,6))
plt.bar(results_dict.keys(), results_dict.values(), color=['blue','green','ora
plt.ylabel("Test Accuracy")
plt.title("Model Comparison on 20 Newsgroups Subset")
plt.ylim(0,1)
for i, v in enumerate(results_dict.values()):
    plt.text(i, v+0.01, f"{v:.2f}", ha='center')
plt.show()
```


Model Comparison on 20 Newsgroups Subset
Test Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
0.88
0.79
0.89
0.88
Logistic Regression
Naive Bayes
SVM
CNN

In [ ]: