**TASK 3(ML):**

**SPAM SMS DETECTION**

**Build an AI model that can classify SMS messages as spam or**

**legitimate. Use techniques like TF-IDF or word embeddings with**

**classifiers like Naive Bayes, Logistic Regression, or Support Vector**

**Machines to identify spam messages**

**Building an AI model to classify SMS messages as spam or legitimate involves several steps, from data preprocessing to model evaluation. Here’s a detailed guide on how to create such a model using techniques like TF-IDF and word embeddings with classifiers like Naive Bayes, Logistic Regression, or Support Vector Machines (SVM).**

**1. Data Collection**

**You need a dataset of SMS messages labeled as spam or legitimate. A popular dataset for this task is the** [**SMS Spam Collection Dataset**](https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection)**. This dataset contains SMS messages classified into "ham" (legitimate) and "spam."**

**2. Data Preprocessing**

**a. Load the Dataset**

**python**

**Copy code**

**import pandas as pd**

**# Load the dataset**

**df = pd.read\_csv('spam.csv', encoding='latin1')**

**df = df[['v1', 'v2']] # Select relevant columns**

**df.columns = ['label', 'message'] # Rename columns**

**# Display the first few rows**

**print(df.head())**

**b. Preprocess the Text**

**You need to clean and preprocess the text data. This includes:**

* **Removing special characters and numbers.**
* **Converting text to lowercase.**
* **Tokenizing the text.**
* **Removing stop words.**

**python**

**Copy code**

**import re**

**from nltk.corpus import stopwords**

**from nltk.tokenize import word\_tokenize**

**stop\_words = set(stopwords.words('english'))**

**def preprocess\_text(text):**

**text = re.sub(r'\W', ' ', text) # Remove special characters**

**text = text.lower() # Convert to lowercase**

**text = re.sub(r'\d', '', text) # Remove numbers**

**tokens = word\_tokenize(text) # Tokenize**

**tokens = [word for word in tokens if word not in stop\_words] # Remove stopwords**

**return ' '.join(tokens)**

**df['processed\_message'] = df['message'].apply(preprocess\_text)**

**c. Encode Labels**

**Convert the labels ("ham" and "spam") into binary values.**

**python**

**Copy code**

**from sklearn.preprocessing import LabelEncoder**

**label\_encoder = LabelEncoder()**

**df['label\_encoded'] = label\_encoder.fit\_transform(df['label'])**

**3. Feature Extraction**

**Convert the text data into numerical features. You can use TF-IDF or word embeddings.**

**a. TF-IDF Vectorization**

**python**

**Copy code**

**from sklearn.feature\_extraction.text import TfidfVectorizer**

**# Initialize TF-IDF Vectorizer**

**tfidf\_vectorizer = TfidfVectorizer(max\_features=5000)**

**X = tfidf\_vectorizer.fit\_transform(df['processed\_message'])**

**y = df['label\_encoded']**

**b. Word Embeddings**

**If you choose to use word embeddings, you can use models like Word2Vec or GloVe. Here’s an example using pre-trained Word2Vec embeddings.**

**python**

**Copy code**

**from gensim.models import Word2Vec**

**from gensim.utils import simple\_preprocess**

**import numpy as np**

**# Tokenize the messages**

**tokenized\_messages = [simple\_preprocess(msg) for msg in df['message']]**

**# Train a Word2Vec model**

**word2vec\_model = Word2Vec(sentences=tokenized\_messages, vector\_size=100, window=5, min\_count=1, sg=1)**

**def vectorize\_message(message):**

**vectors = [word2vec\_model.wv[word] for word in simple\_preprocess(message) if word in word2vec\_model.wv]**

**if len(vectors) == 0:**

**return np.zeros(word2vec\_model.vector\_size)**

**return np.mean(vectors, axis=0)**

**X\_word2vec = np.array([vectorize\_message(msg) for msg in df['message']])**

**4. Model Training**

**Choose and train classifiers.**

**a. Naive Bayes**

**python**

**Copy code**

**from sklearn.naive\_bayes import MultinomialNB**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.metrics import classification\_report, confusion\_matrix**

**# Split the data**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**# Initialize and train the model**

**nb\_classifier = MultinomialNB()**

**nb\_classifier.fit(X\_train, y\_train)**

**# Predict and evaluate**

**y\_pred\_nb = nb\_classifier.predict(X\_test)**

**print("Naive Bayes Classification Report:\n", classification\_report(y\_test, y\_pred\_nb))**

**print("Naive Bayes Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred\_nb))**

**b. Logistic Regression**

**python**

**Copy code**

**from sklearn.linear\_model import LogisticRegression**

**# Initialize and train the model**

**lr\_classifier = LogisticRegression(max\_iter=1000)**

**lr\_classifier.fit(X\_train, y\_train)**

**# Predict and evaluate**

**y\_pred\_lr = lr\_classifier.predict(X\_test)**

**print("Logistic Regression Classification Report:\n", classification\_report(y\_test, y\_pred\_lr))**

**print("Logistic Regression Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred\_lr))**

**c. Support Vector Machines**

**python**

**Copy code**

**from sklearn.svm import SVC**

**# Initialize and train the model**

**svc\_classifier = SVC(kernel='linear')**

**svc\_classifier.fit(X\_train, y\_train)**

**# Predict and evaluate**

**y\_pred\_svc = svc\_classifier.predict(X\_test)**

**print("SVM Classification Report:\n", classification\_report(y\_test, y\_pred\_svc))**

**print("SVM Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred\_svc))**

**5. Model Evaluation**

**Evaluate the models using metrics like precision, recall, F1-score, and confusion matrix. This will help you understand the performance of each model in classifying SMS messages.**

**6. Hyperparameter Tuning**

**To improve the model’s performance, you can tune hyperparameters using techniques like GridSearchCV.**

**python**

**Copy code**

**from sklearn.model\_selection import GridSearchCV**

**# Example for Logistic Regression**

**param\_grid = {**

**'C': [0.1, 1, 10],**

**'solver': ['liblinear', 'saga']**

**}**

**grid\_search = GridSearchCV(estimator=lr\_classifier, param\_grid=param\_grid, cv=3, scoring='f1')**

**grid\_search.fit(X\_train, y\_train)**

**print("Best parameters:", grid\_search.best\_params\_)**

**print("Best score:", grid\_search.best\_score\_)**

**7. Deployment**

**Once you have a well-performing model, you can deploy it to classify incoming SMS messages as spam or legitimate in a real-time application.**

**Conclusion**

**This guide covers the essential steps to build a spam SMS detection model. Depending on the results and the specific requirements, you might need to adjust the preprocessing steps, try different feature extraction methods, or explore additional classifiers.**