Write a Python program that processes the Kaggle - SMS Spam Collection dataset(https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset) to classify tweets as spam or not spam using TF-IDF, Word2Vec, and GloVe embeddings for feature extraction and Preprocess text (tokenization, stop-word removal, lemmatization). Use a classical machine learning model (e.g., Logistic Regression, Naïve Bayes, SVM, or Random Forest) for classification.

```
import nltk
import shutil
import os

# Define the path where you want to store NLTK data
nltk_data_path = os.path.expanduser('~/nltk_data')

# Remove the directory if it exists (optional, for a clean installation)
if os.path.exists(nltk_data_path):
    shutil.rmtree(nltk_data_path)

# Create the directory if it doesn't exist
os.makedirs(nltk_data_path, exist_ok=True)

# Set the NLTK data path
nltk.data.path.append(nltk_data_path)

# Download the 'punkt_tab' resource to the specified path
nltk.download('punkt_tab', download_dir=nltk_data_path)

# Now you can use word_tokenize
from nltk.tokenize import word_tokenize
print(word_tokenize("Hello world!"))
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
['Hello', 'world', '!']
```

```
import nltk
# Download the 'wordnet' resource if not already downloaded
nltk.download('wordnet')

import numpy as np
import pandas as pd
import re
import gensim.downloader as api
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from tqdm import tqdm


def preprocess_text(text):
    text = re.sub(r'[^a-zA-Z]', ' ', text)
    text = text.lower()
    tokens = word_tokenize(text)
    # Download stopwords if not already downloaded

    tokens = [word for word in tokens if word not in stopwords.words('english')]
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens]
    return ' '.join(tokens)

# Load dataset
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/NLP/Exp-9/spam.csv", encoding='latin-1')
df = df[['v1', 'v2']]
df.columns = ['label', 'text']
df['label'] = df['label'].map({'ham': 0, 'spam': 1})
df['text'] = df['text'].apply(preprocess_text)

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(df['text'], df['label'], test_size=0.2, random_state=42)

# TF-IDF Feature Extraction
tfidf_vectorizer = TfidfVectorizer()
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

```
# Train classifiers

models = {
    "Naïve Bayes": MultinomialNB(),
    "Logistic Regression": LogisticRegression(max_iter=500),
    "SVM": SVC(kernel='linear'),
    "Random Forest": RandomForestClassifier(n_estimators=100)
}

for name, model in models.items():
    model.fit(X_train_tfidf, y_train)
    y_pred = model.predict(X_test_tfidf)
    print(f"{name} Accuracy: {accuracy_score(y_test, y_pred):.4f}")
    print(classification_report(y_test, y_pred))

# Word2Vec Embeddings
word2vec_model = api.load("word2vec-google-news-300")

def get_word2vec_embedding(text):
    words = text.split()
    word_vectors = [word2vec_model[word] for word in words if word in word2vec_model]
    return np.mean(word_vectors, axis=0) if word_vectors else np.zeros(300)

X_train_w2v = np.array([get_word2vec_embedding(text) for text in tqdm(X_train)])
X_test_w2v = np.array([get_word2vec_embedding(text) for text in tqdm(X_test)])

# Train Logistic Regression on Word2Vec
logreg = LogisticRegression(max_iter=500)
logreg.fit(X_train_w2v, y_train)
y_pred_w2v = logreg.predict(X_test_w2v)
print(f"Word2Vec Logistic Regression Accuracy: {accuracy_score(y_test, y_pred_w2v):.4f}")
print(classification_report(y_test, y_pred_w2v))

# GloVe Embeddings
glove_vectors = api.load("glove-wiki-gigaword-300")

def get_glove_embedding(text):
    words = text.split()
    word_vectors = [glove_vectors[word] for word in words if word in glove_vectors]
    return np.mean(word_vectors, axis=0) if word_vectors else np.zeros(300)

X_train_glove = np.array([get_glove_embedding(text) for text in tqdm(X_train)])
X_test_glove = np.array([get_glove_embedding(text) for text in tqdm(X_test)])

logreg_glove = LogisticRegression(max_iter=500)
logreg_glove.fit(X_train_glove, y_train)
y_pred_glove = logreg_glove.predict(X_test_glove)
print(f"GloVe Logistic Regression Accuracy: {accuracy_score(y_test, y_pred_glove):.4f}")
print(classification_report(y_test, y_pred_glove))
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
Naïve Bayes Accuracy: 0.9632
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       965
           1       1.00      0.73      0.84       150

    accuracy                           0.96      1115
   macro avg       0.98      0.86      0.91      1115
weighted avg       0.96      0.96      0.96      1115

Logistic Regression Accuracy: 0.9570
              precision    recall  f1-score   support

           0       0.96      0.99      0.98       965
           1       0.96      0.71      0.82       150

    accuracy                           0.96      1115
   macro avg       0.96      0.85      0.90      1115
weighted avg       0.96      0.96      0.95      1115

SVM Accuracy: 0.9830
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       965
           1       0.98      0.89      0.93       150

    accuracy                           0.98      1115
   macro avg       0.98      0.95      0.96      1115
weighted avg       0.98      0.98      0.98      1115

Random Forest Accuracy: 0.9758
              precision    recall  f1-score   support
```

```
           0       0.97      1.00      0.99       965
           1       1.00      0.82      0.90       150

    accuracy                           0.98      1115
   macro avg       0.99      0.91      0.94      1115
weighted avg       0.98      0.98      0.97      1115

[==================================================] 100.0% 1662.8/1662.8MB downloaded
100%|██████████| 4457/4457 [00:00<00:00, 15111.88it/s]
100%|██████████| 1115/1115 [00:00<00:00, 15671.10it/s]
Word2Vec Logistic Regression Accuracy: 0.9453
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       965
           1       0.83      0.75      0.79       150

    accuracy                           0.95      1115
   macro avg       0.90      0.86      0.88      1115
weighted avg       0.94      0.95      0.94      1115

[==================================================] 100.0% 376.1/376.1MB downloaded
100%|██████████| 4457/4457 [00:00<00:00, 11870.37it/s]
100%|██████████| 1115/1115 [00:00<00:00, 13543.92it/s]
GloVe Logistic Regression Accuracy: 0.9498
```