SMS Spam Detection Model

Students

- Hussein AbdElkader
- Ahmed Hesham
- Elsherif Shaban

Overview

This repository houses a machine learning model designed to detect spam messages in SMS (Short Message Service) data. The model is built using the ID3 algorithm and implemented using the scikit-learn library.

Spam SMS Dataset

- **Description:** Classifies SMS messages as spam or ham.
- **Records:** 5573
- **Target variable:** Spam classification (spam or ham)
- **Python libraries:** `pandas`, `scikit-learn`, `id3`

Getting Started

Prerequisites

- Python 3
- Libraries: pandas, scikit-learn, and id3

Install the required libraries using:

```
pip install pandas scikit-learn
```

Usage

1. Clone this repository:

```
git clone [repository_url]
cd spam-detection-model
```

2. Download the SMS Spam Collection Dataset (e.g., 'spam.csv').

3. Run the model:

```
python main.py
```

4. Explore the results in the console. The accuracy and classification report will be displayed.

Files and Directory Structure

- `main.py` : Main script containing the implementation of the ID3 algorithm and model evaluation.
- `spam.csv` : SMS Spam Collection Dataset (not included, download and place in the same directory).

- `README.md` : Documentation file.

Model Details

- **ID3 Algorithm**: The model uses the Iterative Dichotomiser 3 (ID3) algorithm for decision tree-based classification.
- **Feature Extraction**: Text data is transformed using the CountVectorizer to convert messages into a format suitable for machine learning.
- **Training and Evaluation**: The model is trained on a subset of the dataset, and its performance is evaluated on another subset.

Results

> *Class distribution:*
> *ham 4825*
> *spam 747*
> *Class distribution after Undersample:*
> *ham 747*
> *spam 747*
> *Accuracy on Validation Set: 0.9285714285714286*
> *Accuracy on Test Set: 0.8977777777777778*
> *tree command :*

```
dot -Tpdf tree.dot -o tree.pdf
```

1. **Undersampling (Downsampling):**
    - *Pros:*
        - Reduces the computational cost.
        - May improve model training time.
    - *Cons:*
        - Potential loss of information from the majority class.
2. **Oversampling (Upsampling):**
    - *Pros:*
        - Provides more examples of the minority class for the model to learn from.
        - Reduces the risk of ignoring the minority class.
    - *Cons:*
        - May increase the risk of overfitting, especially if not carefully implemented.

> *In such imbalanced scenarios, oversampling the minority class (spam) or undersampling the majority class (ham) are common techniques to address the imbalance.*
> *We will go with undersampling the majority class (ham)*

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from id3 import Id3Estimator
from id3 import export_graphviz
import matplotlib.pyplot as plt
```

```
from sklearn.metrics import accuracy_score, classification_report
from imblearn.under_sampling import RandomUnderSampler

df = pd.read_csv('spam.csv', encoding='latin-1')

print(df.head())
print("Columns:", df.columns)
print("Class distribution:\n", df['v1'].value_counts()) # values to see the spam and
hum sum

# Preprocess the Data
df = df.drop(['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], axis=1)

# Using CountVectorizer to convert text data to a format suitable for machine
learning
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df['v2']) # fit transform to messages v2 (X)

# Undersample the majority class (ham)
rus = RandomUnderSampler(random_state=42)
X_resampled, y_resampled = rus.fit_resample(X, df['v1'])
print("Class distribution after Undersample:\n", y_resampled.value_counts())

# Split into Training (70%) and Temporary Data (30%)
X_train, X_temp, y_train, y_temp = train_test_split(X_resampled, y_resampled,
test_size=0.3, random_state=42)

# Split Temporary Data into Validation (50%) and Test (50%)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,
random_state=42)

# Implement the ID3 Algorithm and Train the Model
clf = Id3Estimator()
clf.fit(X_train.toarray(), y_train, check_input=True)

# Visualize the Decision Tree
export_graphviz(clf.tree_, 'tree.dot',
feature_names=vectorizer.get_feature_names_out())

# Metrics for Validation Set
X_val_dense = X_val.toarray()
y_val_pred = clf.predict(X_val_dense)

print("Accuracy on Validation Set:", accuracy_score(y_val, y_val_pred))

# Evaluate the Model on Test Set
X_test_dense = X_test.toarray()
y_test_pred = clf.predict(X_test_dense)

print("Accuracy on Test Set:", accuracy_score(y_test, y_test_pred))
```