

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

Getting Started

Prerequisites

- Python 3
- Libraries: pandas, scikit-learn

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

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 sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
import graphviz
from sklearn.tree import export_graphviz
from imblearn.under_sampling import RandomUnderSampler

# Step 1: Load the dataset
df = pd.read_csv('spam.csv', encoding='latin-1')

# Step 2: Check the Loaded dataset
print(df.head())
```

```

print("Columns:", df.columns)
print("Missing values:\n", df.isnull().sum())
print("Class distribution:\n", df['v1'].value_counts())

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

# Step 4: Feature Extraction
# Using CountVectorizer to convert text data to a format suitable for machine
learning
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df['v2'])

# Step 5: Undersample the Majority Class
rus = RandomUnderSampler(random_state=42)
X_resampled, y_resampled = rus.fit_resample(X, df['v1'])

# Step 6.1: 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)

# Step 6.2: 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)

# Step 7: Implement the ID3 Algorithm
# Step 8: Train the Model
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

# Step 9: Evaluate the Model on Validation Set
y_val_pred = clf.predict(X_val)

# Corrected part: Use vectorizer.get_feature_names_out() for feature names
plt.figure(figsize=(18, 12))
plot_tree(clf, filled=True, feature_names=vectorizer.get_feature_names_out(),
class_names=['non-spam', 'spam'], rounded=True)
# plt.show()

# Metrics for Validation Set
print("Accuracy on Validation Set:", accuracy_score(y_val, y_val_pred))
print("Classification Report on Validation Set:\n", classification_report(y_val,
y_val_pred))

# Step 10: Evaluate the Model on Test Set
y_test_pred = clf.predict(X_test)

# Metrics for Test Set
print("Accuracy on Test Set:", accuracy_score(y_test, y_test_pred))
print("Classification Report on Test Set:\n", classification_report(y_test,
y_test_pred))

```