BUILDING A SMARTER AI-POWERED SPAM CLASSIFIER

Aut104304-D. EBENEZER

Project Title: Building a Smarter AI-Powered Spam Classifier.

Phase 4: Development Part-2.

TOPIC:

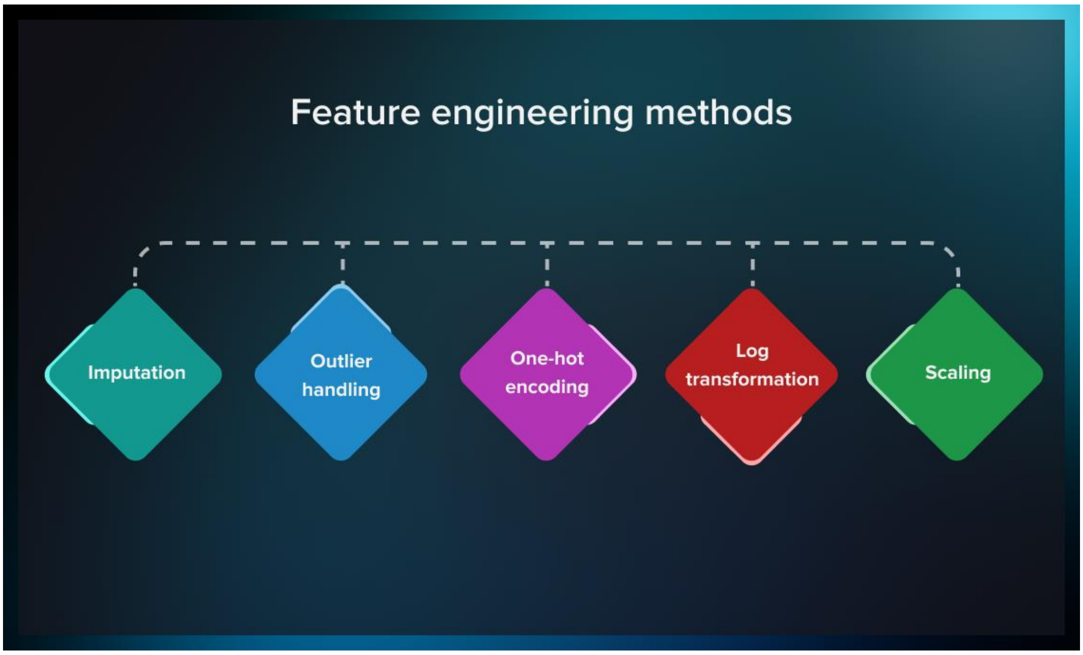
Continue building the project by performing different activities like feature engineering, model training, evaluation etc as per the instructions in the project.

INTRODUCTION:

In today's interconnected world, where digital communication plays a pivotal role in our daily lives, the relentless onslaught of spam messages has become a ubiquitous nuisance. Email inboxes, social media feeds, and messaging platforms are inundated with unwanted and often malicious content, making the task of managing digital clutter and maintaining online security an ever-growing challenge. This is where the development of a smarter AI-based spam classifier takes center stage.

In this introduction, we will explore the pressing need for an intelligent, AI-driven solution to combat the proliferation of spam. We will discuss the potential impact such a system can have on improving online communication and ensuring a safer digital environment. With the advent of artificial intelligence, machine learning, and advanced data analysis techniques, the prospect of building a smarter AI-based spam classifier is not only feasible but also essential for the well-being of individuals, businesses, and the broader online ecosystem. This introduction serves as a gateway to understanding the significance, capabilities, and potential benefits of a more intelligent approach to spam detection and prevention.

FEATURE ENGINEERING:



IMPUTATION:

Imputation is the process of managing missing values, which is one of the most common problems when it comes to preparing data for machine learning. By missing values, we mean places where information is missing in some cells of a respective row.

There may be different causes for missing values, including human error, data flow interruptions, cross-datasets errors, etc. Since data completeness impacts how well machine learning models perform, imputation is quite important.

Here are some ways how you can solve the issue of missing values:

- If a row is less than 20-30% complete, it’s recommended to dismiss such a record.
- A standard approach to assigning values to the missing cells is to calculate a mode, mean, or median for a column and replace the missing values with it.
- In other cases, there are possibilities to reconstruct the value based on other entries. For example, we can find out the name of a country if we have the name of a city and an administrative unit. Conversely, we can often determine the country/city by a postal code.

```
# Missing Value Transactions

# There are no missing observations in the data set, but Glucose, Insulin etc.

# It is known that variable values other than Pregnancies and Outcome cannot be 0 in a human.

# Therefore, we replace the missing 0 values with NaN.

na_cols = ["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI", "Age"]

df[na_cols] = df[na_cols].replace(0, np.nan)

# Let's assign a variable to our table where we checked for missing values.

na_columns = missing_values_table(df, na_name=True)

# Examining the Relationship of Missing Values with the Dependent Variable

# The function flags each observation that contains missing.

# returns a DataFrame that gives the average of the target.

# In this way, the effects of the missing observation situation .

def missing_vs_target(dataframe, target, na_columns):
    temp_df = dataframe.copy()
    for col in na_columns:
        temp_df[col + '_NA_FLAG'] = np.where(temp_df[col].isnull(), 1, 0)
        na_flags = temp_df.loc[:, temp_df.columns.str.contains("_NA_")].columns
        for col in na_flags:
        print(pd.DataFrame({"TARGET_MEAN": temp_df.groupby(col)[target].mean(), "Count": temp_df.groupby(col)[target].count()}), end="\n\n\n")
missing_vs_target(df, "Outcome", na_columns)
```

```
for col in na_cols:
   df.loc[df[col].isnull(), col] = df[col].median()
```

n_miss ratio

Insulin	374 48.700
Skin Thickness	227 29.560
Blood Pressure	35 4.560
BMI	11 1.430
Glucose	5 0.650

TARGET_MEAN Count

Glucose_NA_FLAG

0	0.349	763
1	0.400	5

TARGET_MEAN Count

BloodPressure_NA_FLAG

0	0.344	733
1	0.457	35

TARGET_MEAN Count

SkinThickness_NA_FLAG

0	0.333	541
1	0.388	227

TARGET_MEAN Count

Insulin_NA_FLAG

0	0.330	394
1	0.369	374

TARGET_MEAN Count

BMI_NA_FLAG

0	0.351	757
1	0.182	11

OUTLIER HANDLING:

Outlier handling is another way to increase the accuracy of data representation. Outliers are data points that are significantly different from other observations.

It can be done by removing or replacing outliers. Check out this post for an overview of the five most popular approaches to handling outliers.

```
# We have already identified the outliers by analysis above.
# Again using the replace_with_thresholds function below, we determined
# replaces those with a value below the lower value with the lower value
def replace_with_thresholds(dataframe, variable, q1=0.10, q3=0.90):
    low_limit, up_limit = outlier_thresholds(dataframe, variable, q1, q3)
    dataframe.loc[(dataframe[variable] < low_limit), variable] = low_limit
    dataframe.loc[(dataframe[variable] > up_limit), variable] = up_limit

# Run the function for all variables.
for col in df.columns:
    print(col, check_outlier(df, col))
    if check_outlier(df, col):
        replace_with_thresholds(df, col)

# Checking outlier result for Insulin variable
grab_outliers(df, "Insulin", index=True)
```

```
Pregnancies False
Glucose False
```

```
BloodPressure False
SkinThickness True
Insulin True
BMI False
DiabetesPedigreeFunction True
Age False
Outcome False
Empty DataFrame
Columns: [Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age, Outcome]
Index: []
```

MODEL TRAINING

Your dataset had too many variables to wrap your head around, or even to print out nicely. How can you pare down this overwhelming amount of data to something you can understand?

We'll start by picking a few variables using our intuition. Later courses will show you statistical techniques to automatically prioritize variables.

To choose variables/columns, we'll need to see a list of all columns in the dataset. That is done with the columns property of the DataFrame (the bottom line of code below).

```
import pandas as pd
melbourne_file_path = '../input/melbourne-housing-snapshot/melb_data.csv'
melbourne_data = pd.read_csv(melbourne_file_path)
```

```
melbourne_data.columns

Out[1]:

Index(['Suburb', 'Address', 'Rooms', 'Type', 'Price', 'Method', 'SellerG',

       'Date', 'Distance', 'Postcode', 'Bedroom2', 'Bathroom', 'Car',

       'Landsize', 'BuildingArea', 'YearBuilt', 'CouncilArea', 'Lattitude',

       'Longtitude', 'Regionname', 'Propertycount'],

       dtype='object')
```

```
# The Melbourne data has some missing values (some houses for which some variables weren't recorded.)

# We'll learn to handle missing values in a later tutorial.

# Your Iowa data doesn't have missing values in the columns you use.

# So we will take the simplest option for now, and drop houses from our data.

# Don't worry about this much for now, though the code is:


# dropna drops missing values (think of na as "not available")

melbourne_data = melbourne_data.dropna(axis=0)
```

There are many ways to select a subset of your data. The Pandas Course covers these in more depth, but we will focus on two approaches for now.

- Dot notation, which we use to select the "prediction target"
- Selecting with a column list, which we use to select the

SELECTING THE PREDICTION TARGET

You can pull out a variable with dot-notation. This single column is stored in a Series, which is broadly like a DataFrame with only a single column of data.

We'll use the dot notation to select the column we want to predict, which is called the prediction target. By convention, the prediction target is called y. So the code we need to save the house prices in the Melbourne data is

```
y = melbourne_data.Price
```

CHOOSING "FEATURES"

The columns that are inputted into our model (and later used to make predictions) are called "features." In our case, those would be the columns used to determine the home price. Sometimes, you will use all columns except the target as features. Other times you'll be better off with fewer features.

For now, we'll build a model with only a few features. Later on you'll see how to iterate and compare models built with different features.

We select multiple features by providing a list of column names inside brackets. Each item in that list should be a string (with quotes).

Here is an example:

In [4]:

```
melbourne_features = ['Rooms', 'Bathroom', 'Landsize', 'Lattitude', 'Longtitude']
```

By convention, this data is called X.

In [5]:

```
X = melbourne_data[melbourne_features]
```

Let's quickly review the data we'll be using to predict house prices using the describe method and the head method, which shows the top few rows.

In [6]:

```
X.describe()
```

Out[6]:

	Rooms	Bathroom	Landsize	Lattitude	Longtitude
count	6196.000000	6196.000000	6196.000000	6196.000000	6196.000000
mean	2.931407	1.576340	471.006940	-37.807904	144.990201
std	0.971079	0.711362	897.449881	0.075850	0.099165
min	1.000000	1.000000	0.000000	-38.164920	144.542370
25%	2.000000	1.000000	152.000000	-37.855438	144.926198
50%	3.000000	1.000000	373.000000	-37.802250	144.995800
75%	4.000000	2.000000	628.000000	-37.758200	145.052700
max	8.000000	8.000000	37000.000000	-37.457090	145.526350

```
X.head()
```

Evaluation

The evaluation metric for this competition is Root Mean Squared Error (RMSE). The RMSE is a frequently used measure of the differences between values predicted by a model and the values observed in existing data. It is the square root of the average of squared differences between prediction and actual observation and is given by:

Taking the square root of the average squared errors has some interesting implications for RMSE. Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. This means the RMSE should be more useful when large errors are particularly undesirable.

Submission Format

For every customer ID there should be exactly one predicted revenue amount.

The file should contain a header and have the following format:

1, 250

2, 75

3, 100

4, 109

etc.

1. Define Objectives and Goals:

- Clearly define the objectives and goals of making the system smarter. What specific improvements or enhancements are you aiming for?

2. Identify Key Metrics:

- Determine the key performance indicators (KPIs) or metrics that will be used to measure the smartness of the system. These metrics may vary depending on the context, such as energy efficiency, cost savings, user satisfaction, or environmental impact.

3. Baseline Assessment:

- Assess the current state of the system. This includes understanding its existing capabilities, technologies, and processes.

4. Benchmarking:

- Compare the system's current performance against industry standards or best practices. Benchmarking can help identify areas where improvements are needed.

5. Technology Assessment:

- Evaluate the available technologies that can be integrated into the system to make it smarter. This might involve IoT (Internet of Things) devices, AI and machine learning, data analytics, automation, and more.

6. Cost-Benefit Analysis:

- Analyze the costs associated with implementing smarter technologies and processes, and compare them to the potential benefits, such as increased efficiency, reduced operational costs, or enhanced user experience.

7. Risk Assessment:

- Identify potential risks and challenges associated with implementing smarter solutions. Consider factors like data security, privacy concerns, and technology obsolescence.

8. Stakeholder Engagement:

- Involve all relevant stakeholders in the evaluation process, including end-users, employees, and decision-makers. Their input and feedback can provide valuable insights.

9. Pilot Testing:

- If applicable, conduct pilot tests to assess the feasibility and effectiveness of the proposed smart solutions on a smaller scale before full implementation.

10. Data Collection and Analysis:

- Collect relevant data to measure the impact of smart solutions. This may include data on energy consumption, user behavior, or system performance. Analyze the data to draw insights.

11. Iterative Improvement:

- Continuous improvement is key to building a smarter system. Regularly review the system's performance against the defined metrics and adjust strategies as needed.

12. Scalability:

- Consider how easily the smart system can be scaled up to accommodate future growth or increased demand.

13. Security and Privacy:

- Ensure that adequate security and privacy measures are in place to protect sensitive data and prevent unauthorized access.

14. Sustainability:

- Assess the environmental impact of the smart system and explore ways to make it more sustainable and eco-friendly.

15. User Feedback:

- Gather feedback from users and stakeholders to understand their experience with the smarter system and make necessary improvements based on their input.

16. Documentation and Reporting:

- Maintain comprehensive documentation of the evaluation process, findings, and implementation. Regularly report progress to stakeholders.

17. **Long-term Planning:**

- Develop a long-term plan for the continued evolution and maintenance of the smart system.

METRICS AND THE NAIVE PREDICTOR

Because of we are particularly interested in predicting which sms spam accurately. It would seem that using accuracy as a metric for evaluating a particular model's performace would be appropriate. Therefore, a model's ability to precisely predict those which are spam as is more important than the model's ability to recall those not related. We can use F-beta score as a metric that considers both precision and recall:

SUPERVISED LEARNING MODELS

The following are some of the supervised learning models that are currently available in scikit-learn that i will use:

GAUSSIAN NAIVE BAYES

A classical use case is document classification: Determining whether a given (text) document corresponds to one or more categories and for classifying and filtering spam emails based on the likelihood of certain words appearing on an spam email as compared to a non-spam email

Strengths:

- Easy to implement , Simple to run
- If the NB conditional independence assumption holds, then it will converge quicker than discriminative models like logistic regression
- Even if the NB assumption doesn’t hold, it works great in practice.
- Less training data than others
- Can be used for both binary and mult-iclass classification problems.

Weaknesses:

- It can’t learn interactions between features (e.g., it can’t learn that although you love movies with Brad Pitt and Tom Cruise, you hate movies where they’re together).
- Data scarcity For any possible value of a feature, you need to estimate a likelihood value by a frequentist approach. This can result in probabilities going towards 0 or 1, which in turn leads to numerical instabilities and worse results. In this case, you need to smooth in some way your probabilities (e.g. as in sklearn), or to impose some prior on your data, however you may argue that the resulting classifier is not naive anymore.
- This particular model could be a good approach to solving the problem as we have a large data set with few features.

References:

- wikipedia
- quora - real world applications for Naive Bayes classifier
- youtube- udacity - Naive Bayes Strengths and Weaknesses
- quora - the advantages of using a naive Bayes for classification
- quora - disadvantages of using a naive bayes for classification
- AdaBoost:
- A classic use case where AdaBoost algorithms is in the problem of Face Detection.

ADABOOST

Strengths:

- AdaBoost very simple to implement we can achieve similar classification results with much less tweaking of parameters or settings The user only needs to choose:
- which weak classifier might work best to solve their given classification problem.
- the number of boosting rounds that should be used during the training phase.
- Feature selection on very large sets of features.

Weaknesses

- AdaBoost can be sensitive to noisy data and outliers. In some problems, however, it can be less susceptible to the overfitting problem than most learning algorithms. AdaBoost algorithm does not currently support null rejection, although this will be added at some point in the near future.
- This particular model could be a good approach to solving the problem as we have a large data set so we can we can perform multiple quick trainining iterations to maximize our overall accuracy.

References:

- analyticsvidhya
- nickgillian
- slideshare - Machine learning with ADA Boost
- linkcode

SUPPORT VECTOR MACHINES (SVC)

- SVMs can be used to solve various real world problems:
- SVMs are helpful in text and hypertext categorization as their application can significantly reduce the need for labeled training instances in both the standard inductive and transductive settings.
- Face Detection.

It classifies the parts of the image as face and non-face. It contains training data of n x n pixels with a two-class face (+1) and non-face (-1). Then it extracts features from each pixel as face or non-face. Creates a square boundary around faces on the basis of pixel brightness and classifies each image by using the same process.

- Classification of images.

Experimental results show that SVMs achieve significantly higher search accuracy than traditional query refinement schemes after just three to four rounds of relevance feedback.

- Hand-written characters can be recognized Bioinformatics.
- Strengths:

- High accuracy
- Nice theoretical guarantees regarding overfitting
- With an appropriate kernel they can work well even if you’re data isn’t linearly separable in the base feature space.

Weaknesses:

- Don't perform so well in very large data set because the training time happens to be cubic in size of data set.
- Don't work very well in lots and lots of noise do when class are very overlapping you have to count independent evidence that's where the naive bayes classifier would be better.
- The theory only really covers the determination of the parameters for a given value of the regularisation and kernel parameters and choice of kernel. In a way the SVM moves the problem of over-fitting from optimising the parameters to model selection. Sadly kernel models can be quite sensitive to over-fitting the model selection criterion.
- The model could be still be a good candidate as there seems to be some features in the data that can more clearly define the income level boundary and dataset not very large so it won't take so long training time

References:

- wikipedia
- data-flair - Real-Life Applications of SVM
- blog.echen.me
- youtube - udacity - SVM Strengths and Weaknesses
- stackexchange - Advantages and disadvantages of SVM

CONCLUSION

Building a smarter future is an aspiration that carries immense significance and potential for our world. In conclusion, the pursuit of a smarter future involves harnessing the power of technology, fostering innovation, and prioritizing sustainability. It requires a collective effort from individuals, communities, governments, and industries. By focusing on education, research, and responsible use of technology, we can unlock the vast opportunities presented by artificial intelligence, automation, and data analytics.

This journey towards a smarter future holds the promise of improving the quality of life, addressing pressing global challenges, and creating a more equitable and sustainable world for future generations. As we navigate this path, it is essential to consider ethical and social implications and to ensure that our quest for intelligence is balanced with wisdom, compassion, and a commitment to the well-being of all. In building a smarter future, we must be guided by the principles of responsible innovation, inclusivity, and the preservation of our planet, while continually striving to enhance the collective intelligence of humanity.