School of Engineering and Sciences: Computer Science and Engineering

SUMMER RESEARCH INTERNSHIP PROJECT

Supervised Machine Learning and its Application

Supriya Manna: AP21110010560,
Sahitya Akula: AP21110011110

Supervisor: **Dr. Niladri Sett**

August 29, 2023
SRM University, AP

A REPORT

ON

Supervised Machine Learning and its Application

By

Name (s) of the student (s)	Registration No.
Supriya Manna	AP21110010560
Sahitya Akula	AP21110011110
--------------------------------	--------------------------

Prepared in the partial fulfillment of the
Summer Internship Course

SRM University, AP

Mangalagiri -Mandal, Neeru Konda, Amaravati, Andhra Pradesh 522502

SRM UNIVERSITY, AP

(July, 2023)

Acknowledgement

Acknowledged with appreciation to Vice Chancellor Prof Manoj K. Arora, HOD of CSE Dr Jatindra Kumar Dash, and Mentor Dr. Niladri Sett.

Chapter 1

Abstract

The research internship was centered on supervised machine learning, with a focus on Bayesian statistical methodologies. The internship encompassed diverse models and techniques, including linear regression, logistic regression, Binomial and Multinomial Naive Bayes, as well as feature engineering and exploratory data analysis (EDA). Additionally, the internship culminated in the implementation of a spam filter using Naive Bayes variants, demonstrating practical application. This experience deepened my understanding of these models and their real-world relevance, providing a solid foundation for addressing contemporary data-driven challenges.

Contents

Overview of the Internship, Brief Introduction and Objective

During this 8-week research internship, we embarked on an immersive journey encompassing various aspects of machine learning and data analysis. The primary focus of our research was on the mathematical underpinnings of Bayesian statistics and supervised machine learning. We dived deep into understanding the theoretical foundations, including probabilistic models and algorithms. Furthermore, we delved into the realm of feature engineering, an essential component for enhancing model performance. Exploratory Data Analysis (EDA) and data preprocessing techniques were explored to glean insights from raw data, a critical step in building robust machine learning models.

Our day-to-day activities and progress were meticulously documented in a Diary, which served as a comprehensive record of our experiences and challenges. The internship culminated in the development of a sophisticated email spam filter. Leveraging the skills and knowledge gained, we successfully designed and implemented the filter to effectively classify emails as spam or non-spam.

Guided by Dr. Niladri Sett, an expert with a diverse background in machine learning paradigms, this internship provided us with a holistic understanding of the intricate interplay between theory and application in the field of machine learning. We are grateful for the support and guidance we received throughout the internship, and we are confident that the skills and insights gained will be invaluable in our future endeavors.

CHAPTER 2

Machine Learning: Introduction, Modesls and Application

Machine Learning has evolved from early concepts in pattern recognition and artificial intelligence. The foundations were laid in the 20th century, with Alan Turing's work and the development of perceptrons. The advent of digital computing and data availability led to the growth of ML in the latter half of the century. Notable milestones include the perceptron convergence theorem, decision trees, and the emergence of neural networks.

2.1 Introductory Problem Space Discussion and Approaches

Problem Definition: In supervised learning, we work with a dataset of input-output pairs: $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i$ represents the input feature vector and y_i is the corresponding output label. Our objective is to find a function $f(\mathbf{x})$ that minimizes a defined loss function.

Loss Function: The loss function $L(y, f(\mathbf{x}))$ quantifies the disparity between predictions and actuals. In the context of regression, the mean squared error (MSE) is commonly employed:

$$L(y, \hat{y}) = (y - \hat{y})^2$$

Gradient Descent: To minimize the loss function, we utilize gradient descent, an iterative optimization algorithm. Model parameters β are updated iteratively in the opposite direction of the gradient:

$$\beta_{t+1} = \beta_t - \eta \nabla L(y, f(\mathbf{x}))$$

Here, η denotes the learning rate.

Linear Regression: Linear regression assumes a linear relationship between inputs and outputs:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$$

Proof of Linear Regression Solution: The solution is derived by setting the gradient of the loss function with respect to β to zero:

$$\frac{\partial}{\partial \beta} \sum_i (y_i - \hat{y}_i)^2 = 0$$

Solving for β yields the optimal coefficients.

Logistic Regression: Logistic regression models binary outcomes using the logistic function:

$$P(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n)}}$$

Proof of Logistic Regression Solution: Coefficients in logistic regression are obtained by maximizing the likelihood of observed data using the logistic function's inverse logit transformation.

Convergence of Gradient Descent: Convergence proofs rely on the Lipschitz continuity and convexity properties of the loss function.

Gradient Descent: To minimize the loss function, we utilize gradient descent, an iterative optimization algorithm. Model parameters β are updated iteratively in the opposite direction of the gradient:

$$\beta_{t+1} = \beta_t - \eta \nabla L(y, f(\mathbf{x}))$$

Here, η denotes the learning rate.

Proof of Convergence: The convergence of gradient descent can be established by considering the properties of the loss function and the learning rate.

Theorem 1 (Convergence of Gradient Descent): Let the loss function L be differentiable and convex, and let the learning rate η be chosen such that $0 < \eta \leq \frac{2}{L}$, where L is the Lipschitz constant of the gradient of L. Then, gradient descent converges to the global minimum.

Proof: The update step of gradient descent is given by:

$$\beta_{t+1} = \beta_t - \eta \nabla L(y, f(\mathbf{x}))$$

Applying the convexity property of L and using Taylor's theorem, we have:

$$L(y, f(\mathbf{x}_{t+1})) \leq L(y, f(\mathbf{x}_t)) + \nabla L(y, f(\mathbf{x}_t))^T (\mathbf{x}_{t+1} - \mathbf{x}_t) + \frac{L}{2} \|\mathbf{x}_{t+1} - \mathbf{x}_t\|_2^2$$

Substituting the gradient descent update:

$$L(y, f(\mathbf{x}_{t+1})) \leq L(y, f(\mathbf{x}_t)) - \eta \|\nabla L(y, f(\mathbf{x}_t))\|_2^2 + \frac{L}{2} \|\eta \nabla L(y, f(\mathbf{x}_t))\|_2^2$$

Using the Cauchy-Schwarz inequality and simplifying:

$$L(y, f(\mathbf{x}_{t+1})) \leq L(y, f(\mathbf{x}_t)) - \frac{\eta}{2} \|\nabla L(y, f(\mathbf{x}_t))\|_2^2$$

This implies that the loss decreases at each step, ensuring convergence. With a suitable choice of learning rate, the algorithm converges to the global minimum. CONVERGENCE AND DIVERGENCE ANALYSIS OF GRADIENT DESCENT

Consider a convex loss function $L(\beta)$ and the gradient descent update rule:

$$\beta_{t+1} = \beta_t - \eta \nabla L(\beta_t)$$

Convergence can be guaranteed if the learning rate η satisfies the following conditions: 1. $0 < \eta \leq \frac{2}{L}$, where L is the Lipschitz constant of the gradient of $L(\beta)$. 2. $\sum_{t=1}^{\infty} \eta_t = \infty$, where η_t is the learning rate at iteration t. 3. $\sum_{t=1}^{\infty} \eta_t^2 < \infty$.

Divergence Conditions: Conversely, if the learning rate is chosen too large, the algorithm might diverge. If $\eta > \frac{2}{L}$, the sequence $\{\beta_t\}$ diverges.

Optimal Learning Rate in Linear Regression: In the case of linear regression, an optimal learning rate can be determined using matrix manipulations. The closed-form solution for linear regression is given by:

$$\beta^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Here, $\mathbf{X}$ is the design matrix and $\mathbf{y}$ is the target vector. The optimal learning rate η^* can be found by considering the eigenvalues of $\mathbf{X}^T\mathbf{X}$. If all eigenvalues are positive, then η^* is the reciprocal of the largest eigenvalue for stable convergence.

In conclusion, the learning rate plays a crucial role in the convergence or divergence of the gradient descent algorithm. Proper selection, guided by theoretical considerations and experimentation, is essential for achieving efficient optimization in machine learning models.

Variants of Gradient Descent

Batch Gradient Descent:

For a convex loss function $L(\beta)$, update the parameter vector β at iteration t using the entire dataset:

$$\beta_{t+1} = \beta_t - \eta \nabla L(\beta_t)$$

Significance: Accurate gradients due to full dataset, suitable for convex functions, computationally intensive for large datasets.

Stochastic Gradient Descent (SGD):

Update β at iteration t using a single randomly selected example:

$$\beta_{t+1} = \beta_t - \eta \nabla L(y_i, f(\mathbf{x}_i, \beta_t))$$

Significance: Efficient for large datasets, may converge faster due to frequent updates, susceptible to more noise.

Algorithm: Gradient Descent

Input: Training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, learning rate η, maximum iterations T, initial parameter values β_0.

Output: Optimal parameter values β^*.

Procedure:

1. Initialize $\beta = \beta_0$.
2. For $t = 1$ to T:
   (a) Compute the gradient: $\nabla L(y, f(\mathbf{x}_i)) = \frac{\partial L(y, f(\mathbf{x}_i))}{\partial \beta}$ for each $(\mathbf{x}_i, y_i)$ in the training set.
   (b) Update β using the gradient descent formula:
$$\beta = \beta - \eta \nabla L(y, f(\mathbf{x}))$$
3. Return $\beta^* = \beta$.

Algorithm: Linear Regression

Input: Training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, learning rate η, maximum iterations T, initial parameter values β_0.

Output: Optimal parameter values β^*.

Procedure:

1. Run the Gradient Descent Algorithm with the mean squared error loss function:
$$L(y, \hat{y}) = (y - \hat{y})^2$$
2. Return the final parameter values β^*.

Algorithm: Logistic Regression

Input: Training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, learning rate η, maximum iterations T, initial parameter values β_0.

Output: Optimal parameter values β^*.

Procedure:

1. Run the Gradient Descent Algorithm with the logistic loss function:
$$L(y, f(\mathbf{x})) = -y \log(f(\mathbf{x})) - (1 - y) \log(1 - f(\mathbf{x}))$$
2. Return the final parameter values β^*.

2.2 Bayesian Statistical Frameworks: Naive Bayes

Bayesian statistics provides a probabilistic framework for reasoning about uncertainty by incorporating prior knowledge and observed data. In the context of machine learning, it offers a principled approach to modeling and inference.

Bayes' Theorem: At the core of Bayesian inference is Bayes' theorem, which relates the posterior probability $P(\theta|D)$ of a hypothesis θ given observed data D to the prior probability $P(\theta)$ and the likelihood $P(D|\theta)$:

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$$

Bayesian Machine Learning Models

Bayesian principles can be applied to various machine learning models, allowing us to capture uncertainty and make predictions in a coherent manner.

Probabilistic Models: In Bayesian machine learning, models are treated as probabilistic frameworks. A probabilistic model defines a joint distribution $P(\mathbf{X}, \mathbf{Y}, \theta)$ over inputs $\mathbf{X}$, outputs $\mathbf{Y}$, and model parameters θ.

Parameter Estimation: Bayesian parameter estimation involves computing the posterior distribution $P(\theta|\mathbf{X}, \mathbf{Y})$ given data $\mathbf{X}$ and $\mathbf{Y}$. This distribution characterizes the uncertainty in parameter values.

Predictive Inference: Given new input data $\mathbf{X}^*$, Bayesian inference provides the predictive distribution $P(\mathbf{Y}^*|\mathbf{X}^*, \mathbf{X}, \mathbf{Y})$, which gives the uncertainty in predictions $\mathbf{Y}^*$.

Benefits and Challenges

Benefits: - Bayesian models naturally handle uncertainty and provide credible intervals for predictions. - Incorporating prior knowledge leads to robust and interpretable models. - Bayesian methods can mitigate overfitting through regularization.

Challenges: - Analytical solutions may be intractable, requiring approximations like Markov Chain Monte Carlo (MCMC). - Bayesian methods can be computationally expensive for large datasets.

Bayesian statistics enriches machine learning with a rigorous framework to quantify uncertainty, make informed decisions, and build models that align with our understanding of the data.

Naive Bayes Classifier

Problem Space

Given a feature vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, the task is to classify it into one of the classes C_k, where $k = 1, 2, \ldots, K$. Bayes' theorem provides a probabilistic framework for this:

$$P(C_k|\mathbf{x}) = \frac{P(C_k)P(\mathbf{x}|C_k)}{P(\mathbf{x})}$$

Exact Probabilistic Algorithm

To solve the classification problem, we compute the posterior probability $P(C_k|\mathbf{x})$ for each class C_k. The class with the highest probability is chosen as the final classification.

Proof of the Algorithm

By applying Bayes' theorem and the assumption that the features are conditionally independent given the class, we derive the Naive Bayes formula:

$$P(C_k|\mathbf{x}) \propto P(C_k) \prod_{i=1}^{n} P(x_i|C_k)$$

The class with the highest posterior probability is the maximum likelihood estimate.

Limitations of Exact Algorithm

The exact algorithm requires computing $P(\mathbf{x})$, which involves marginalizing over all classes. This computation is computationally expensive and impractical for large datasets.

Naive Bayes Approach

To address the limitations of the exact algorithm, the Naive Bayes approach makes a conditional independence assumption among features given the class. This simplifies the computation of $P(\mathbf{x}|C_k)$:

$$P(\mathbf{x}|C_k) = P(x_1|C_k) \cdot P(x_2|C_k) \cdot \ldots \cdot P(x_n|C_k)$$

Variants

Binomial Naive Bayes

For binary features $x_i \in \{0, 1\}$, we use Bernoulli distributions for $P(x_i|C_k)$.

Algorithm for Binomial Naive Bayes:

For each class C_k:

1. Estimate $P(C_k)$ from training data.
2. Estimate $P(x_i = 1|C_k)$ from training data.
3. Compute posterior probabilities and classify based on the highest probability.

Multinomial Naive Bayes

For categorical features x_i with N possible values, we use multinomial distributions for $P(x_i|C_k)$.

Algorithm for Multinomial Naive Bayes:

For each class C_k:

1. Estimate $P(C_k)$ from training data.
2. Estimate $P(x_i|C_k)$ from training data.
3. Compute posterior probabilities and classify based on the highest probability.

CHAPTER 3

Feature Selection, EDA and Various Methods

In the realm of data-driven problem solving, Feature Selection, Exploratory Data Analysis (EDA), and Feature Engineering stand as essential pillars. Feature Selection optimizes model performance by identifying the most informative attributes, reducing complexity, and enhancing interpretability. EDA unveils hidden insights, relationships, and anomalies within data, guiding subsequent decisions. Feature Engineering transforms raw data into meaningful representations, capturing domain knowledge and improving model generalization. This chapter delves into the mathematical foundations of these techniques, unraveling their significance, problem spaces, objectives, and mathematical underpinnings, to empower robust data-driven decision-making.

3.1 Introductory Discussion of the Problem Space and Approaches

3.1.1 Data Prepossessing

Data preprocessing is a critical stage in refining raw data for subsequent analysis. Given a dataset $\mathbf{D}$ with N samples and M features, the preprocessing objective is to apply transformations $\mathcal{T}$ to generate the preprocessed dataset $\mathbf{D}'$.

Problem Space and Objective

Problem Space: $\mathbf{D} = \{(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)\}, \mathbf{x}_i \in \mathbb{R}^M$

Objective: $\mathbf{D}' = \{\mathcal{T}(\mathbf{x}_1), \mathcal{T}(\mathbf{x}_2), \ldots, \mathcal{T}(\mathbf{x}_N)\}$

Preprocessing Operations

Data Cleaning: Impute missing values: $x'_{ij} = \text{mean}(x_j)$ if x_{ij} is missing.

Data Transformation: Normalize features: $x'_{ij} = \frac{x_{ij} - \text{mean}(x_j)}{\text{std}(x_j)}$.

Data Reduction: Apply PCA for dimensionality reduction: $\mathbf{X}' = \mathbf{XW}$.

Outlier Detection: Identify outliers: x'_{ij} is an outlier if $\frac{|x_{ij} - \text{mean}(x_j)|}{\text{std}(x_j)}$ exceeds a threshold.

Categorical Encoding: One-hot encode categorical variables: cat $\rightarrow [0, 1, 0]$.

Feature Extraction: Extract features: Text or image data $\rightarrow$ numerical features.

Normalization: Rescale data to [0, 1]: $x'_{ij} = \frac{x_{ij} - \min(x_j)}{\max(x_j) - \min(x_j)}$.

Significance

accuracy in data preprocessing optimizes data quality, enhances uniformity, and prepares the dataset for accurate analysis and modeling.

3.1.2 Feature Selection, EDA, and Feature Engineering

Feature Selection: Feature selection is crucial in simplifying models and enhancing interpretability, reducing dimensionality, and mitigating overfitting by retaining only relevant features.

Exploratory Data Analysis (EDA): EDA is a foundational step that uncovers data patterns, identifies outliers, and guides feature engineering by providing insights into relationships among variables.

Feature Engineering: Feature engineering transforms raw data into informative features, enhancing model performance, capturing domain knowledge, and improving generalization.

Feature Selection

Problem Space: Given a feature matrix $\mathbf{X}$ and corresponding target vector $\mathbf{y}$, select a subset of features to optimize model performance.

Objective:

$$\min_{\mathbf{S}\subseteq\{1,2,\ldots,n\}} \mathrm{Loss}(\mathbf{X}_{\mathbf{S}}, \mathbf{y})$$

Exploratory Data Analysis (EDA)

Problem Space: Given a dataset $\mathbf{D}$, discover underlying patterns, relationships, and anomalies to inform subsequent modeling steps.

Objective: Explore the data's distribution, central tendencies, dispersion, and relationships among variables.

Feature Engineering

Problem Space: Given a feature matrix $\mathbf{X}$, engineer new features or transform existing ones to improve model performance and capture domain knowledge.

Objective: Create informative features using operations like scaling, transformation, aggregation, interaction, and domain-specific knowledge.

3.2 Feature Selection Methods

3.2.1 Basic Methods

Correlation-based Feature Selection (CFS):

Objective: Select features that maximize the relevance with the target and minimize redundancy among the selected features.

Algorithm: 1. Calculate feature-target correlation: $C(\text{feature}, \text{target}) = \frac{\text{cov(feature,target)}}{\sigma_{\text{feature}}\sigma_{\text{target}}}$. 2. Calculate feature-feature correlation: $C(\text{feature}_i, \text{feature}_j)$. 3. Define a merit function that combines relevance and redundancy measures. 4. Select features with the highest merit.

Mutual Information-based Feature Selection:

Objective: Choose features that have high mutual information with the target.

Algorithm: 1. Calculate mutual information between feature X_i and target Y: $I(X_i;Y) = \sum_{x_i \in X_i} \sum_{y \in Y} p(x_i, y) \log \frac{p(x_i,y)}{p(x_i)p(y)}$. 2. Select features with the highest mutual information values.

ANOVA F-test Feature Selection:

Objective: Select features that exhibit significant variation across different classes.

Algorithm: 1. Compute the ANOVA F-statistic for each feature: $F = \frac{\text{between-group variance}}{\text{within-group variance}}$. 2. Select features with the highest F-statistic values.

Entropy-based Feature Selection:

Objective: Choose features that provide maximal information gain with respect to the target.

Algorithm: 1. Calculate the entropy of the target: $H(Y) = -\sum_{y \in Y} p(y) \log p(y)$. 2. For each feature X_i, calculate the conditional entropy: $H(Y|X_i) = -\sum_{x_i \in X_i} \sum_{y \in Y} p(x_i, y) \log p(y|x_i)$. 3. Compute information gain: $IG(X_i) = H(Y) - H(Y|X_i)$. 4. Select features with the highest information gain.

Advanced Methods

Least Absolute Shrinkage and Selection Operator (LASSO):

Objective: Regularize coefficients to encourage feature selection.

Algorithm: Solve the optimization problem with L1 penalty:

$$\min_{\beta} \frac{1}{2N} \sum_{i=1}^{N} (y_i - \mathbf{x}_i^T \beta)^2 + \lambda \sum_{j=1}^{M} |\beta_j|$$

Elastic Net: Combines L1 and L2 regularization for a balance between LASSO and Ridge regression.

3.3 Smoothing

3.3.1 Introduction to Smoothing

Smoothing is a technique commonly used in Natural Language Processing to address the problem of zero probabilities in probabilistic models. It involves redistributing probability mass from observed events to unseen events, preventing the complete discounting of unseen events.

3.3.2 Laplace Smoothing: Mathematical Derivation

Laplace smoothing (add-one smoothing) is a simple and effective method to handle zero probabilities by adding a small constant k to both the numerator and denominator of the probability calculation. This ensures non-zero probabilities even for unseen events.

Objective: Smooth probability estimates to avoid zero probabilities: $P(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i)+k}{C(w_{i-1})+k\cdot V}$.

Overview:

1. Define V as the vocabulary size.
2. $C(w_{i-1}w_i)$ is the count of bigram $w_{i-1}w_i$ occurrences.
3. $C(w_{i-1})$ is the count of unigram w_{i-1} occurrences.
4. Applying Laplace smoothing: $P(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i)+k}{C(w_{i-1})+k\cdot V}$.
5. The added constant k ensures non-zero probabilities for unseen events.

Significance: Laplace smoothing prevents zero probabilities, providing a more robust foundation for probabilistic models.

CHAPTER 4

Project: Designing Spam Filter Using Naive Bayes

4.1 Introduction to the Problem Space and Approach

Introduction

The task of designing a spam filter involves classifying incoming messages as either spam or legitimate based on their content. This problem is a fundamental application of text classification, where the goal is to develop a model that accurately discerns between these two classes.

Problem Space and Objective

Given a set of labeled messages $\mathbf{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}$, where $\mathbf{x}_i$ represents a message and y_i is its label (spam or legitimate), the objective is to build a classifier $f(\mathbf{x})$ that accurately predicts the label of new, unseen messages.

Effectiveness of Naive Bayes

Bayesian Framework:

The Naive Bayes classifier operates within the Bayesian framework, utilizing Bayes' theorem to compute posterior probabilities. For spam classification:

$$P(\text{spam}|\mathbf{x}) = \frac{P(\mathbf{x}|\text{spam})P(\text{spam})}{P(\mathbf{x})}$$

Naive Independence Assumption:

The Naive Bayes assumption of feature independence, though simplistic, facilitates computation:

$$P(\mathbf{x}|\text{spam}) = P(x_1|\text{spam}) \cdot P(x_2|\text{spam}) \cdot \ldots \cdot P(x_n|\text{spam})$$

Advantages:

1. *Efficiency:* Naive Bayes requires fewer parameters for estimation, reducing the risk of overfitting. 2. *Robustness:* It handles irrelevant features gracefully due to its independence assumption. 3. *Interpretability:* Probabilistic interpretation allows for insights into classification decisions.

Limitations:

1. *Feature Independence:* The assumption of feature independence might not hold in complex scenarios. 2. *Data Balance:* Imbalanced data can lead to skewed predictions.

The Naive Bayes framework's simplicity, efficiency, and interpretable nature make it a well-suited choice for tackling the spam filter problem.

4.2 Full Fledged Python Implementation:

```python
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import KFold
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import nltk

# Download NLTK resources
nltk.download('punkt')
nltk.download('stopwords')

class NaiveBayes:
    def __init__(self, alpha=1):
        """
        Naive Bayes classifier for spam filtering using both multinomial

        Args:
            alpha (float): Smoothing parameter. Default is 1.
        """
        self.alpha = alpha
        self.vectorizer = TfidfVectorizer()
        self.kf = KFold(n_splits=10, shuffle=True, random_state=42)

    def preprocess_text(self, text):
        """
        Preprocesses input text for feature extraction.

        Args:
            text (str): Input text.

        Returns:
            str: Preprocessed text.
        """
        text = text.lower()
        tokens = word_tokenize(text)
        tokens = [word for word in tokens if word.isalnum()]
        stop_words = set(stopwords.words('english'))
        tokens = [word for word in tokens if word not in stop_words]
        stemmer = PorterStemmer()
        stemmed_tokens = [stemmer.stem(word) for word in tokens]
        return ' '.join(stemmed_tokens)

    def fit_predict(self, data):
        """
        Fits the Naive Bayes models, predicts and evaluates their perform

        Args:
            data (pd.DataFrame): Input data containing 'Message' and 'Cat

        Returns:
            float, float: Mean accuracy of multinomial Naive Bayes and Be
        """
        X = self.vectorizer.fit_transform(data['Message'])
        y = data['Category']

        mnb_scores = []
        bnb_scores = []

        for train_index, test_index in self.kf.split(X):
            X_train, X_test = X[train_index], X[test_index]
            y_train, y_test = y[train_index], y[test_index]

            mnb_y_pred = self.multinomial_naive_bayes(X_train, y_train, X
            mnb_score = np.mean(mnb_y_pred == y_test)
            mnb_scores.append(mnb_score)

            bnb_y_pred = self.bernoulli_naive_bayes(X_train, y_train, X_t
            bnb_score = np.mean(bnb_y_pred == y_test)
            bnb_scores.append(bnb_score)

            print("Debug Mode:")
            print("Multinomial Naive Bayes - Predicted:", mnb_y_pred)
            print("Bernoulli Naive Bayes - Predicted:", bnb_y_pred)
            print("True Labels:", y_test)
            print()

        mean_mnb_accuracy = np.mean(mnb_scores)
        mean_bnb_accuracy = np.mean(bnb_scores)

        return mean_mnb_accuracy, mean_bnb_accuracy

    def multinomial_naive_bayes(self, X_train, y_train, X_test):
        """
        Applies multinomial Naive Bayes model for classification.

        Args:
            X_train (sparse matrix): Training features.
            y_train (pd.Series): Training labels.
            X_test (sparse matrix): Testing features.

        Returns:
            np.array: Predicted labels.
        """
        X_spam = X_train[y_train == 'spam']
        X_ham = X_train[y_train == 'ham']

        # class priors
        spam_prior = X_spam.shape[0] / X_train.shape[0]
        ham_prior = X_ham.shape[0] / X_train.shape[0]

        # likelihoods in vocabulary
        spam_likelihood = np.array(X_spam.sum(axis=0))[0]
        ham_likelihood = np.array(X_ham.sum(axis=0))[0]

        # some smoothing ig
        spam_likelihood += self.alpha
        ham_likelihood += self.alpha

        # conditional probabilities
        spam_probs = np.log(spam_likelihood / (np.sum(spam_likelihood) +
        ham_probs = np.log(ham_likelihood / (np.sum(ham_likelihood) + len

        spam_log_likelihoods = X_test.dot(spam_probs)
        ham_log_likelihoods = X_test.dot(ham_probs)

        spam_log_posterior = np.log(spam_prior) + spam_log_likelihoods
        ham_log_posterior = np.log(ham_prior) + ham_log_likelihoods

        y_pred = np.where(spam_log_posterior > ham_log_posterior, 'spam',

        return y_pred

    def bernoulli_naive_bayes(self, X_train, y_train, X_test):
        """
        Applies Bernoulli Naive Bayes model for classification.

        Args:
            X_train (sparse matrix): Training features.
            y_train (pd.Series): Training labels.
            X_test (sparse matrix): Testing features.

        Returns:
            np.array: Predicted labels.
        """
        X_spam = X_train[y_train == 'spam']
        X_ham = X_train[y_train == 'ham']

```

```python
        spam_prior = X_spam.shape[0] / X_train.shape[0]
        ham_prior = X_ham.shape[0] / X_train.shape[0]

        spam_likelihood = np.array(X_spam.sum(axis=0))[0]
        ham_likelihood = np.array(X_ham.sum(axis=0))[0]

        spam_likelihood += self.alpha
        ham_likelihood += self.alpha

        spam_probs = np.log(spam_likelihood / (np.sum(spam_likelihood)))
        ham_probs = np.log(ham_likelihood / (np.sum(ham_likelihood)))

        spam_log_likelihoods = X_test.dot(spam_probs)
        ham_log_likelihoods = X_test.dot(ham_probs)

        spam_log_posterior = np.log(spam_prior) + spam_log_likelihoods
        ham_log_posterior = np.log(ham_prior) + ham_log_likelihoods

        y_pred = np.where(spam_log_posterior > ham_log_posterior, 'spam',

        return y_pred

def main():
    data = pd.read_csv('/content/sample_data/spam.csv')

    spam_filter = NaiveBayes(alpha=1)
    mean_mnb_accuracy, mean_bnb_accuracy = spam_filter.fit_predict(data)

    print("Multinomial Naive Bayes - Mean Accuracy:", mean_mnb_accuracy)
    print("Bernoulli Naive Bayes - Mean Accuracy:", mean_bnb_accuracy)

if __name__ == "__main__":
    main()
```

Analysis and Key-points

4.3 Preprocessing

Preprocessing is the initial step to prepare the text data for analysis.

4.3.1 Lowercasing and Tokenization

The algorithm converts text to lowercase and tokenizes it into individual words using the NLTK library.

4.3.2 Removing Special Characters and Stop Words

Special characters are removed, leaving only alphanumeric words. Common stop words are eliminated to focus on meaningful words.

4.3.3 Stemming

Words are stemmed using Porter stemming to reduce them to their root form.

4.4 TF-IDF Vectorization

TF-IDF (Term Frequency-Inverse Document Frequency) transforms text data into numerical vectors that capture the importance of words relative to the entire corpus.

4.4.1 Term Frequency (TF)

The term frequency of a word in a document is calculated using:

$$\text{TF}(t,d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

4.4.2 Inverse Document Frequency (IDF)

The inverse document frequency of a term is calculated using:

$$\text{IDF}(t, D) = \log \frac{\text{Total number of documents in corpus } D}{\text{Number of documents with term } t}$$

4.4.3 TF-IDF Weight

The TF-IDF weight of a term in a document is the product of TF and IDF:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D)$$

4.5 Multinomial Naive Bayes

Multinomial Naive Bayes calculates the probability that a document belongs to a class (spam or ham).

4.5.1 Prior Probabilities

The prior probability of a class is calculated as the ratio of documents in that class to the total number of documents.

4.5.2 Likelihood Probabilities

The likelihood probability of a word given a class is calculated as the frequency of that word in documents of that class.

4.5.3 Smoothing

Smoothing is applied to handle words with zero frequency. A smoothing parameter α is added to the word frequencies.

4.5.4 Prediction

The probability of a document belonging to each class is calculated. The class with the highest probability is assigned as the predicted class.

4.6 Bernoulli Naive Bayes

Bernoulli Naive Bayes is suited for binary data and presence/absence of words.

4.6.1 Prior and Likelihood Probabilities

Similar to Multinomial, but calculated based on presence/absence of words.

4.6.2 Prediction

Similar to Multinomial, but with presence/absence probabilities.

4.7 Efficiency and Advantages

4.7.1 Component Efficiency

- Preprocessing standardizes and reduces data dimensions. - TF-IDF focuses on important words, discards common words, and creates meaningful vectors. - Naive Bayes calculations are simple and memory-efficient.

4.7.2 Algorithm Efficiency

- Scalable to large datasets due to simple counting operations. - Effective for text classification tasks. - Good performance when many features and small data.

4.8 Conclusion

The Naive Bayes algorithm is a simple yet effective method for spam filtering. In this study, we used the Naive Bayes algorithm along with preprocessing and TF-IDF to filter spam emails. We evaluated the performance of the algorithm using 10-fold cross-validation.

The results showed that the multinomial Naive Bayes algorithm achieved a mean accuracy of 94.50 percent, while the Bernoulli Naive Bayes algorithm achieved a mean accuracy of 96.2 percent.

This suggests that the Bernoulli Naive Bayes algorithm is slightly more accurate than the multinomial Naive Bayes algorithm for spam filtering.

CHAPTER 5

Conclusions

This internship project has been a great learning experience. We started by taking a bottom-up approach, first observing the problem space (regression, classification etc), then doing a deep probabilistic and mathematical analysis, followed by learning the mathematical algorithms. We then learned feature engineering and the different models, their significance, and the maths behind their optimality. Finally, we took the project of email spam detection and implemented all the stuff we learned, which proved to get excellent results.

We are confident that the skills and knowledge we have gained during this internship will be valuable in our future careers. We are grateful to our mentors for their guidance and support.

Appendix A

Implementation of Other Algorithms Discussed Earlier

For scalability and upgradeability, we have uploaded other implementations on the GitHub. It's publicity accessible from anywhere in the world.

Link: `https://github.com/humblef-oo-l/Code_Repo`