Machine Learning
Tema practică

Ciudin Ștefana, Său Ionela

ianuarie 2024

Înțegerea setului de date și procesarea

Pentru a ușura utilizarea setului de date, am creat un script în Python care să transforme dataset-ul într-un fișier format .csv, cu antetul: **subject,message,label,folder**. Astfel, subject reprezintă subiectul email-ului, message conținutul efectiv al email-ului, label cu valoarea 0 sau 1 pentru a marca dacă email-ul este spam sau nu (1 pentru email spam) și folder care conține folderul în care email-ul se află (**lemn, bare, stop, sau lemn-stop**), urmat de **part1-part10**, pentru a marca subfolderul în care se află (pentru a separa datele de test de cele de antrenament).

Atributele sunt **subject** (subiectul mesajului, extras din primul rând al fișierului de email), **message** (extras de după subiect, din fișierul de email) și **folder** (extras din numele folderului și subfolderului în care se află fișierul). Eticheta **label**, cu valoarea 0 sau 1, este extrasă în funcție de numele email-ului (fișierele spam au în nume prefixul spm).

Funcțiile Python utilizate pentru a transforma dataset-ul în fișier .csv sunt următoarele:

```
def read_partitioned_dataset(root_folder):
 data = {'subject': [], 'message': [], 'label': [], 'folder': []}

 for folder_name in os.listdir(root_folder):
  folder_path = os.path.join(root_folder, folder_name)
   if os.path.isdir(folder_path):
    for subfolder_name in os.listdir(folder_path):
     subfolder_path = os.path.join(folder_path, subfolder_name)
     if os.path.isdir(subfolder_path):
      for file_name in os.listdir(subfolder_path):
       file_path = os.path.join(subfolder_path, file_name)
       with open(file_path, 'r', encoding='latin1') as file:
        match = re.search(r"Subject:(.+?)\n\n(.+)", content, re.DOTALL)
        if match:
         subject, message = match.group(1), match.group(2)
         label = 1 if 'spm' in file_name else 0
         folder_label = f"{folder_name}-{subfolder_name}"
         data['subject'].append(subject)
         data['message'].append(message)
         data['label'].append(label)
         data['folder'].append(folder_label)

return data


def create_csv(data, output_file):
 df = pd.DataFrame(data)
 df.to_csv(output_file, index=False)
```

Testarea practică a algoritmilor de Învățare Automată

Pentru a determina în mod corect ce algoritm de Învățare Automată să utilizăm în rezolvarea problemei, am făcut o analiză teoretică și practică a tuturor algoritmilor studiați. Pentru a le analiza acuratețea, am folosit implementările din **sklearn**, pentru algoritmii ID3, Naive Bayes, KNN și AdaBoost. Aceste mici scripturi de test au rulat folosind fișierul .csv generat anterior, și folosind drept date de antrenare fișierele din folderele **part1-part9**, iar ca date de testate fișierele din folderele **part10**. De exemplu, pentru KNN, am folosit următoarele funcții Python:

```
def read_dataset_from_csv(csv_file):
    df = pd.read_csv(csv_file)
    return df


def train_knn(x_train, y_train, n_neighbors=5):
    model = KNeighborsClassifier(n_neighbors=n_neighbors)
    model.fit(x_train, y_train)
    return model


def main():
    csv_file = 'lingspam_dataset.csv'

    df = read_dataset_from_csv(csv_file)
    train_data = df[df['folder'].str.contains('-part[1-9]')]

    test_data = df[df['folder'].str.contains('-part10')]

    X_train, y_train = train_data['message'], train_data['label']
    X_test, y_test = test_data['message'], test_data['label']

    vectorizer = TfidfVectorizer()
    X_train_tfidf = vectorizer.fit_transform(X_train)
    X_test_tfidf = vectorizer.transform(X_test)

    for n_neighbors in range(10, 500, 10):
        model = train_knn(X_train_tfidf.toarray(), y_train, n_neighbors)
        y_pred = model.predict(X_test_tfidf.toarray())
        accuracy = accuracy_score(y_test, y_pred)
        print(f"Accuracy for K={n_neighbors}: {accuracy}")
```

Diferențele dintre ID3, Naive Bayes, KNN și AdaBoost

În lumea diversă a învățării automate, algoritmii ID3, Naive Bayes, k-Nearest Neighbors (KNN) și AdaBoost ies în evidență, fiecare aducând propria sa serie de caracteristici. ID3, un algoritm de arbore de decizie, se deosebește prin selecția recursivă a atributelor care oferă cea mai mare cantitate de informație pentru a despărți datele. Abilitatea sa naturală de a gestiona datele categorice îl face o opțiune puternică pentru probleme în care caracteristicile sunt discrete. Cu toate acestea, ID3 se confruntă cu provocări – sensibilitatea la datele zgomotoase și tendința de a face suprapuneri.

Pe partea probabilistică, Naive Bayes se bazează pe teorema lui Bayes și pe prespupunerea independenței caracteristicilor. Simplitatea sa se pretează bine la probabilități ușor de interpretat, făcându-l o alegere excelentă pentru clasificarea textelor sau filtrarea spam-ului. Rezistența sa la datele zgomotoase și viteza relativă de antrenare îl fac o opțiune de încredere pentru rezolvarea anumitor probleme.

Contrastând acestea, KNN operează pe principiul proximității, clasificând punctele de date pe baza clasei

majoritare a celor mai apropiați vecini ai lor. În ciuda versatilității sale și capacității de a gestiona atât date categorice, cât și numerice, interpretarea sa suferă datorită naturii sale de învățare leneșă. Viteza mică la antrenare și sensibilitatea la datele atipice impun o analiză atentă în aplicarea sa.

AdaBoost, pe de altă parte, adoptă o abordare de învățare prin ansamblu. Prin combinarea unor clasificatori slabi într-un clasificator puternic, AdaBoost abordează limitările modelelor individuale. Excelează în scenarii în care clasificatorii slabi pot fi îmbunătățiți iterativ pentru a îmbunătăți performanța generală. Cu toate acestea, interpretarea sa scade pe măsură ce ansamblul crește în complexitate.

Gestionarea valorilor lipsă adaugă un alt nivel de distincție. ID3 și KNN pot gestiona valorile lipsă, dar pot introduce distorsiuni sau pot necesita preprocesare. Naive Bayes cere strategii de imputare, în timp ce AdaBoost, în general, necesită o abordare proactivă pentru a aborda valorile lipsă.

Luând în considerare viteza de antrenare, ID3 și Naive Bayes ies în evidență ca învățători rapizi datorită simplității lor intrinseci. KNN, un învățător leneș, schimbă viteza de antrenare pentru predicții mai rapide. AdaBoost, datorită naturii sale iterative, întârzie în antrenare, dar compensează cu forța colectivă a unui ansamblu.

În domeniul datelor zgomotoase, Naive Bayes strălucește cu fundația sa probabilistică robustă. ID3 și KNN manifestă sensibilitate la zgomot, în timp ce AdaBoost, bazat pe învățători slabi, se poate confrunta cu dificultăți în prezența valorilor extreme.

În concluzie, diferențele dintre ID3, Naive Bayes, KNN și AdaBoost subliniază importanța selectării instrumentului potrivit pentru sarcina în desfășurare. Fiecare algoritm aduce propriile sale puncte forte și slăbiciuni, iar o considerație atentă a acestor nuanțe este crucială pentru aplicații reușite de învățare automată.

Rezultate în urma testării algoritmilor de Învățare Automată

	ID3	**Naive Bayes**	**K-NN**	**AdaBoost**
Acuratețe	0.98223	0.99823	0.98793$^{(*)}$	0.99779

Table 1: Acuratețea algoritmilor de învățare supervizată

$^{(*)}$Acuratețea de 0.98793 a algoritmului KNN a fost obținută prin K=10. Prin testare, am obținut și următoarele seturi de valori:

- Accuracy for K=30: 0.9844827586206897
- Accuracy for K=40: 0.9844827586206897
- Accuracy for K=50: 0.9853448275862069
- Accuracy for K=60: 0.9844827586206897
- Accuracy for K=70: 0.9844827586206897
- Accuracy for K=80: 0.9827586206896551
- Accuracy for K=90: 0.9827586206896551
- Accuracy for K=100: 0.9818965517241379
- Accuracy for K=110: 0.9818965517241379
- Accuracy for K=120: 0.9810344827586207

Alegerea algoritmului Bayes pentru filtrarea spamului în email-uri

În peisajul intricat al filtrării spamului în e-mail, selectarea celui mai eficient algoritm este crucială. Evaluând acuratețea diferitelor algoritmi - Bayes cu un impresionant 99.823%, ID3 cu 98.223%, KNN cu 98.793%, și AdaBoost cu 99.779% - am optat pentru algoritmul Naive Bayes datorită performanței sale remarcabile.

Naive Bayes, ancorat în fundamentele probabilităților, s-a dovedit a fi excepțional de robust în gestionarea subtilităților clasificării e-mailului. Acuratețea sa de 99.823% semnifică o capacitate remarcabilă de a discerne între spam și mesaje legitime. Simplitatea și viteza algoritmului îl fac potrivit în mod deosebit pentru natura diversă și dinamică a conținutului de e-mail.

Comparând cu alți concurenți, precum ID3, deși un algoritm eficient de arbore de decizie, rămâne în urmă cu o acuratețe de 98.223%. KNN și AdaBoost, cu acuratețe de 98.793% și, respectiv, 99.779%, demonstrează o performanță lăudabilă, dar superioritatea marginală a acurateții Naive Bayes a înclinat balanța în favoarea sa.

Avantajele care susțin alegerea acestui algoritm sunt:

Eficiență computațională: Naive Bayes se remarcă prin eficiența sa computațională, fiind potrivit pentru manipularea seturilor de date de dimensiuni mari.

Performanțe în clasificarea textului: Datorită caracteristicilor specifice, Naive Bayes obține rezultate deosebite în clasificarea textului, ceea ce este esențial pentru filtrarea email-urilor.

Manevrabilitatea datelor lipsă: Naive Bayes gestionează cu succes datele lipsă, o caracteristică deosebit de utilă atunci când unele caracteristici ale e-mail-urilor sunt incomplete sau indisponibile.

Rezistența lui Naive Bayes la datele zgomotoase, viteza sa rapidă de antrenare și interpretarea probabilităților au contribuit la alegerea sa. În contextul filtrării spamului, unde precizia este crucială, Naive Bayes se impune ca algoritmul care trebuie ales, oferind o soluție puternică și fiabilă pentru a deosebi între comunicarea autentică și spamul nedorit.

Python Code

```
class NaiveBayes:
    def __init__(self, alpha=1.0):
        self.alpha = alpha
        self.class_probs = {}
        self.word_probs = {}
        self.classes = []

    def fit(self, X, y):
        self.classes = np.unique(y)

        for c in self.classes:
            self.class_probs[c] = (y == c).sum() / len(y)

        vectorizer = CountVectorizer()
        X_count = vectorizer.fit_transform(X)
        word_counts = X_count.toarray()
        feature_names = vectorizer.get_feature_names_out()

        for i, c in enumerate(self.classes):
            class_word_counts = word_counts[y == c].sum(axis=0)
            total_words_in_class = class_word_counts.sum()

            self.word_probs[c] = {}
            for j, word in enumerate(feature_names):
                word_prob = (class_word_counts[j] + self.alpha) / (
                        total_words_in_class + self.alpha * len(feature_names))
                self.word_probs[c][word] = word_prob
```

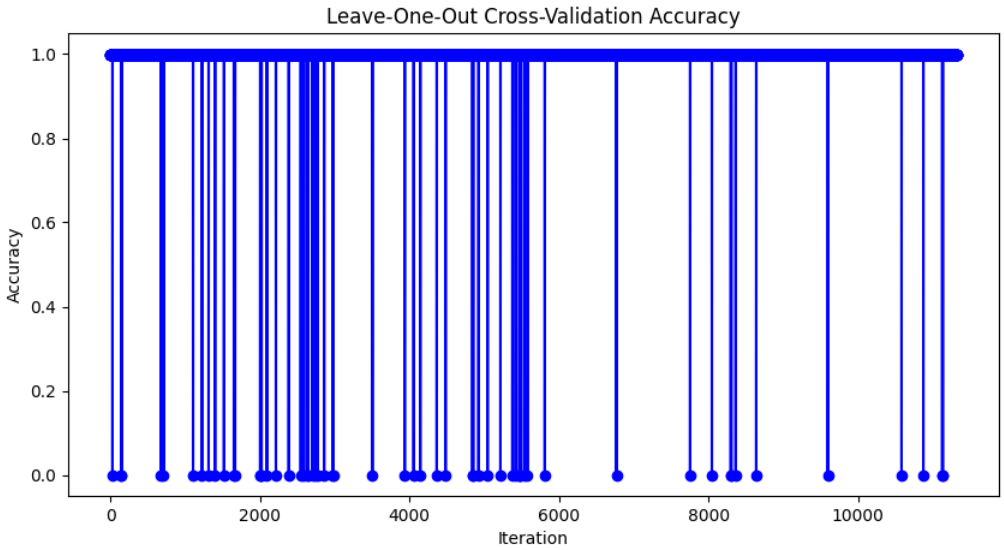
```
def predict(self, X):
    predictions = []

    for x in X:
        probs = {c: np.log(self.class_probs[c]) for c in self.classes}

        for word in x.split():
            for c in self.classes:
                if word in self.word_probs[c]:
                    probs[c] += np.log(self.word_probs[c][word])

        predicted_class = max(probs, key=probs.get)
        predictions.append(predicted_class)

    return predictions
```

Rezultatele Clasificatorului Naive Bayes

	Precision	**Recall**	**F1 Score**	**Support**
0	1.00	1.00	1.00	968
1	1.00	0.98	0.99	192
Accuracy			1.00	1160
Macro avg	1.00	0.99	0.99	1160
Weighted avg	1.00	1.00	1.00	1160

Rezultate Cross-Validare Leave-One-Out

Am atașat un grafic pentru a ilustra rezultatele obținute la Cross-Validare Leave-One-Out.



Rezultate acuratețe reprezentate grafic

Am atașat un grafic pentru a ilustra acuratețea algoritmului Naive Bayes pe setul de date.