UNIVERZITET U NOVOM SADU
PRIRODNO-MATEMATIČKI FAKULTET
DEPARTMAN ZA MATEMATIKU I INFORMATIKU

Projekat ML3

Analiza sentimenta i klasifikacija komentara na filmove

- Praktični projekat iz predmeta Veštačka inteligencija -

Milan Balabanović 347/21

Novi Sad 2024.

Sadržaj

1. Uvod

Projekat se bavi analizom sentimenta i klasifikacijom komentara ostavljenih na filmove u pozitivne i negativne. Razvijen je u Python-u uz korišćenje naprednih biblioteka kao što su **pandas (2.2.2)**, **scikit-learn (1.5.1)**, **numpy (2.1.0)**, **nltk (3.9.1)** i **transformers (4.32.0)**.

U okviru ovog projekta, implementirani su i testirani različiti modeli za klasifikaciju teksta, uključujući **Naive Bayes**, **SVM** (Support Vector Machines), **Logistic Regression**, i **Random Forest**. Svaki od ovih modela je primenjen na zadatak prepoznavanja pozitivnih i negativnih komentara, pri čemu su korišćene različite tehnike ekstrakcije karakteristika iz teksta, kao što su **Bag of Words**, **TF-IDF**, **Hashing**, kao i napredniji pristupi poput **DistilBERT** reprezentacije. Pre ekstrakcije karakteristika i treniranja modela, odradjen je preprocessing podataka kako bi se tekstovi očistili i doveli u stanje da budu spremni za dalju obradu.

Cilj ovog rada je bio da se uporede performanse ovih modela u kontekstu analize sentimenta.

2. Učitavanje i preprocessing podataka

Podaci u ovom projektu smešteni su u direktorijumu `**data**`, koji je dalje podeljen na dva poddirektorijuma: `**2800**` i `**50000**`, u skladu sa brojem komentara koje sadrže. Učitavanje podataka izvršeno je čitanjem svakog tekstualnog dokumenta i dodavanjem pročitane sadržine u odgovarajuće liste (slika 2.1). Nakon toga, sproveden je preprocessing podataka koji obuhvata prebacivanje svih slova u mala, uklanjanje karaktera koji nisu alfanumerički, kao i brisanje stop reči (slika 2.2).

```
def load_data(data_dir):
    global instanceCount
    texts, labels = [], []

    for split in ['train', 'test']:
        for label_dir in ['pos', 'neg']:
            folder_path = os.path.join(data_dir, split, label_dir)
            label = 1 if label_dir == 'pos' else 0

            for filename in os.listdir(folder_path):
                file_path = os.path.join(folder_path, filename)
                with open(file_path, 'r', encoding='utf-8', errors='ignore') as file:
                    instanceCount += 1
                    texts.append(file.read())
                    labels.append(label)

    return pd.Series(texts), pd.Series(labels)
```

slika 2.1

```
def preprocess_text(text):
    tokens = word_tokenize(text.lower())
    tokens = [word for word in tokens if word.isalpha()]
    tokens = [word for word in tokens if word not in stopwords.words('english')]
    return ' '.join(tokens)
```

slika 2.2

3. Reprezentacija teksta

Reprezentacija teksta je ključni korak u procesu obrade prirodnog jezika (NLP) i predstavlja način na koji se tekstualni podaci pretvaraju u numerički format koji može biti korišćen za analizu i modeliranje. U osnovi, cilj je da se tekstu dodeli numerički format koji očuvava njegovu semantiku i značenje. Postoji nekoliko metoda za reprezentaciju teksta:

1. **Bag of Words (BoW)**: Bag of Words (BoW) je jedna od najosnovnijih metoda za reprezentaciju teksta. Ova tehnika pretvara tekst u vektor gde svaka pozicija odgovara određenoj reči u rečniku, a vrednost na toj poziciji predstavlja broj pojavljivanja te reči u dokumentu. BoW ne uzima u obzir redosled reči i ne pruža informacije o kontekstu reči, ali je jednostavna za implementaciju i često se koristi kao osnovna metoda za analizu teksta.
2. **Term Frequency-Inverse Document Frequency (TF-IDF)**: TF-IDF je naprednija metoda koja kombinuje frekvenciju reči u dokumentu sa brojem pojavljivanja te reči u celom skupu dokumenata. Frekvencija reči u dokumentu (TF) meri koliko često se reč pojavljuje u dokumentu, dok Inverse Document Frequency (IDF) meri koliko je reč retka u celokupnom skupu dokumenata. Kombinacija ovih faktora pomaže u identifikaciji reči koje su specifične za određeni dokument i koje imaju veću važnost za analizu.
3. **Kontekstualni Embedding**: Savremeni modeli kao što su BERT (Bidirectional Encoder Representations from Transformers) i DistilBERT pružaju kontekstualni embedding. Ovi modeli uzimaju u obzir kontekst reči u rečenici, omogućavajući generisanje dinamičnih vektora koji se menjaju u zavisnosti od konteksta u kojem se reč pojavljuje.

```python
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from transformers import DistilBertTokenizer, DistilBertModel
import numpy as np

distilbert_tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
distilbert_model = DistilBertModel.from_pretrained('distilbert-base-uncased')

def extract_features(texts, method='tfidf'):
    if method == 'tfidf':
        vectorizer = TfidfVectorizer()
        return vectorizer.fit_transform(texts)
    elif method == 'bow':
        vectorizer = CountVectorizer()
        return vectorizer.fit_transform(texts)
    elif method == 'bert':
        def vectorize_text(text):
            inputs = distilbert_tokenizer(text, return_tensors='pt', truncation=True, padding=True)
            outputs = distilbert_model(**inputs)
            return outputs.last_hidden_state.mean(dim=1).squeeze().detach().numpy()

        feature_matrix = np.array([vectorize_text(text) for text in texts])
        return feature_matrix

    else:
        raise ValueError(f"Unknown feature extraction method: {method}")
```

4. Modeli

1. Naive Bayes

Naive Bayes klasifikatori koriste Bayesov-u teoremu kako bi izračunali verovatnoću da jedan tekst pripada određenoj klasi na osnovu njegovih karakteristika (reči, u ovom slučaju).

Prednosti

- **Brzina**: Obuka i predikcija su veoma brzi, čak i za velike skupove podataka.
- **Efikasnost sa Malim Skupovima Podataka**: Dobro funkcioniše čak i kada je količina obučavajućih podataka mala.
- **Robustnost**: Osetljiv je na šum i greške u podacima, ali često i dalje daje dobre rezultate.

Mane

- **Pretpostavka Nezavisnosti**: Pretpostavka da su karakteristike nezavisne nije uvek tačna, što može dovesti do lošijih performansi u nekim slučajevima.
- **Ograničena Fleksibilnost**: Može biti manje efikasan u slučajevima kada su veze između karakteristika složene ili kada karakteristike nisu nezavisne.

```
class NaiveBayesModel:
    def __init__(self):
        self.model = Pipeline([
            ('nb', MultinomialNB())
        ])

    def train(self, X_train, y_train):
        """Train the Naive Bayes model."""
        self.model.fit(X_train, y_train)

    def predict(self, X_test):
        """Make predictions using the trained model."""
        return self.model.predict(X_test)

    def predict_proba(self, X_test):
        """Predict class probabilities using the trained model."""
        return self.model.predict_proba(X_test)
```

2. SVM (Support Vector Machine)

SVM je zasnovan na konceptu pronalaženja optimalne hiperravnine koja razdvaja podatke u različite klase. Optimalna hiperravnina je ona koja maksimizuje marginu između klasa, tj. rastojanje između najbližih tačaka sa obe strane hiperravnine (poznate kao *support vectors*).

Prednosti

- **Efikasnost sa Visokodimenzionalnim Podacima**: SVM je vrlo efikasan u radu sa podacima visokih dimenzija i dobro funkcioniše čak i kada je broj karakteristika veći od broja uzoraka.
- **Robustnost na Overfitting**: Korišćenje pravila regularizacije i marginu omogućava SVM-u da se efikasno nosi sa prekomernim prilagođavanjem (overfitting).
- **Fleksibilnost**: Upotrebom različitih kernel funkcija, SVM može modelovati složene nelinearne odnose između karakteristika.

Mane

- **Skaliranje**: SVM može biti računski intenzivan i spor za veoma velike skupove podataka.
- **Odabir Parametara**: Odabir optimalnih vrednosti za hiperparametre, kao što su C (regularizacija) i gamma (za RBF kernel), može biti izazovan i zahteva pažljivo podešavanje.
- **Interpretabilnost**: SVM modeli mogu biti teži za interpretaciju u poređenju sa jednostavnim modelima poput logističke regresije.

```python
class SVMModel:
    def __init__(self):
        self.model = Pipeline([
            ('svm', SVC(kernel='linear', probability=True))
        ])

    def train(self, X_train, y_train):
        """Train the SVM model."""
        self.model.fit(X_train, y_train)

    def predict(self, X_test):
        """Make predictions using the trained model."""
        return self.model.predict(X_test)

    def predict_proba(self, X_test):
        """Predict class probabilities using the trained model."""
        return self.model.predict_proba(X_test)
```

3. Logistic regression

Logistička regresija je metoda koja predviđa verovatnoću pripadnosti uzorka određenoj klasi koristeći funkciju logističke funkcije (ili sigmoidnu funkciju). Ova funkcija pretvara bilo koji realan broj u vrednost između 0 i 1, što je idealno za modelovanje verovatnoće da uzorak pripada jednoj od dve klase.

Prednosti

- **Jednostavnost**: Logističku regresiju je lako implementirati i interpretirati. Dobro funkcioniše za probleme sa linearnim granicama između klasa.
- **Brza i Efikasna**: Ima brze algoritme za obuku, što je korisno za velike skupove podataka.
- **Probabilistička Predikcija**: Obezbeđuje verovatnoće za predikcije, što može biti korisno u mnogim aplikacijama gde je potrebna interpretacija neizvesnosti.

Mane

- **Ograničena na Linearne Granice**: Ako su odnosi između karakteristika i ciljne promenljive nelinearni, model može imati loše performanse bez dodatnih transformacija.
- **Osetljivost na Neuravnotežene Klase**: Može biti osetljiv na neuravnotežene skupove podataka, gde jedna klasa može biti znatno zastupljenija od druge.
- **Ograničeno za Višeklasnu Klasifikaciju**: Iako se može proširiti na višeklasnu klasifikaciju koristeći metode kao što je "one-vs-rest", logistička regresija je prirodno dizajnirana za binarnu klasifikaciju.

```
class LogisticRegressionModel:
    def __init__(self):
        self.model = Pipeline([
            ('lr', LogisticRegression(max_iter=1000))
        ])

    def train(self, X_train, y_train):
        """Train the Logistic Regression model."""
        self.model.fit(X_train, y_train)

    def predict(self, X_test):
        """Make predictions using the trained model."""
        return self.model.predict(X_test)

    def predict_proba(self, X_test):
        """Predict class probabilities using the trained model."""
        return self.model.predict_proba(X_test)
```

4. Random forest

Random Forest kombinuje rezultate više odlučujućih stabala kako bi se dobila konačna predikcija. Svako stablo u šumi daje svoju predikciju, a konačni rezultat se dobija glasanjem (za klasifikaciju) ili prosečnim vrednostima (za regresiju) svih stabala.

Prednosti

- **Robusnost:** Random Forest je manje osetljiv na overfitting u poređenju sa pojedinačnim odlučujućim stablima jer koristi ensemble tehniku.
- **Preciznost:** Obično postiže visoku tačnost i može se nositi sa velikim i složenim skupovima podataka.
- **Izdržljivost:** Može da se nosi sa nedostajućim vrednostima i šumom u podacima.
- **Automatska Selekcija Karakteristika:** Može automatski da oceni važnost karakteristika, što može biti korisno za selekciju atributa.

Mane

- **Složenost:** Model može postati složen i teško interpretirati zbog velikog broja stabala u šumi.
- **Resursi:** Može zahtevati dosta memorije i procesorske snage, posebno za velike skupove podataka i brojne karakteristike.
- **Potreba za Tuning-om:** Iako generalno robusno, Random Forest može zahtevati podešavanje hiperparametara (kao što su broj stabala i maksimalna dubina stabala) za postizanje optimalnih performansi.

```
class RandomForestModel:
    def __init__(self):
        self.model = Pipeline([
            ('rf', RandomForestClassifier())
        ])

    def train(self, X_train, y_train):
        """Train the Random Forest model."""
        self.model.fit(X_train, y_train)

    def predict(self, X_test):
        """Make predictions using the trained model."""
        return self.model.predict(X_test)

    def predict_proba(self, X_test):
        """Predict class probabilities using the trained model."""
        return self.model.predict_proba(X_test)
```

5. Rezultati testiranja modela

Svo treniranje i testiranje bilo je odrađeno sa data set-om od 2800 instanci. Podela podataka za treniranje i testiranje bila je odradjena metodom **train_test_split(test_size=0.2, random_state=42)** iz biblioteke **Scikit-learn**.

Za evaluaciju modela koričćena je metoda **evaluate_model** (slika 5.1) koja vraća 4 metrike:

- **Tačnost** - U klasifikaciji, ova funkcija računa tačnost skupa tako što podeli broj tačno klasifikovanih instanci sa ukupnim brojem instanci.
- **Preciznost** - Preciznost intuitivno predstavlja sposobnost klasifikatora da ne označi uzorak kao pozitivan kada je on zapravo negativan.
- **Odziv** - Odziv (recall) se računa kao odnos između broja pravih pozitivnih i zbir broja pravih pozitivnih i broja lažnih negativnih. Odziv meri sposobnost klasifikatora da pronađe sve pozitivne uzorke.
- **F1 skor** - F1 skor, poznat i kao balansirani F1 skor ili F-merit, može se interpretirati kao harmonijska sredina preciznosti i odziva.

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    return accuracy, precision, recall, f1
```

slika 5.1

Nakon testiranja svih modela sa svim tehnika reprezentacije teksta došli smo do sledećih rezultata:

Naziv model	Tehnika reprezentacije teksta	Tačnost	Preciznost	Odziv	F1
Naive bayes	Bag of words	0.9315	0.9275	0.9343	0.9309
	TF-IDF	0.9279	0.9209	0.9343	0.9275
	DistilBert	0.9281	0.9153	0.9250	0.9201
SVM	Bag of words	0.9784	0.9852	0.9708	0.9779
	TF-IDF	0.9640	0.9568	0.9708	0.9638
	DistilBert	0.9850	0.9916	0.9795	0.9855
Logistic regression	Bag of words	0.9820	0.9925	0.9708	0.9815
	TF-IDF	0.9441	0.9451	0.9416	0.9433
	DistilBert	0.9818	0.9850	0.9751	0.9795
Random forest	Bag of words	0.9423	0.9481	0.9343	0.9412
	TF-IDF	0.9495	0.9489	0.9489	0.9489
	DistilBert	0.9550	0.9602	0.9505	0.9553

6. Zaključak

Ovaj praktični zadatak ističe značaj izbora odgovarajućih modela i tehnika reprezentacije teksta u analizi sentimenta komentara na filmove. Analizom performansi različitih modela, uključujući Naive Bayes, SVM, Logistic Regression i Random Forest, u kombinaciji sa različitim tehnikama reprezentacije teksta kao što su Bag of Words, TF-IDF i DistilBert, dobili smo uvid u njihove prednosti i nedostatke.

SVM sa DistilBert pokazuje se kao najefikasniji model, pružajući najbolje rezultate u svim ključnim metrikama: tačnosti, preciznosti, odzivu i F1 skoru. Ovaj model se izdvaja zbog svoje sposobnosti da precizno identifikuje pozitivne i negativne komentare, što je od velikog značaja za analizu sentimenta. S druge strane, Logistic Regression sa Bag of Words takođe pokazuje visoke performanse, naročito u preciznosti, dok Random Forest sa DistilBert nudi stabilne rezultate sa visokom tačnošću i F1 skorom.

Naive Bayes, iako daje solidne rezultate, pokazuje nešto slabije performanse u poređenju sa SVM i Logistic Regression, ali i dalje predstavlja pouzdan izbor za određene primene. Korišćenje DistilBert kao tehnike reprezentacije teksta često dovodi do poboljšanja u rezultatima, što ukazuje na prednost u korišćenju naprednih modela za obuku u analizi teksta.

Dok SVM sa DistilBert pruža vrhunske rezultate, druge kombinacije kao što su Logistic Regression sa Bag of Words i Random Forest sa DistilBert takođe nude značajne prednosti i mogu biti korisne u zavisnosti od konteksta i zahteva projekta.