Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek

Računarstvo usluga i analiza podataka

SEMINARSKI RAD

Klasifikacija sentimenta teksta

Adrijan Malinović

Osijek, 2024.

Sadržaj

1. Uvod

Klasifikacija sentimenta teksta, također poznata kao rudarenje mišljenja, jedno je od ključnih područja **strojnog učenja** (engl. *Machine Learning, ML*) i **obrade prirodnog jezika** (engl. *Natural Language Processing, NLP*). Ovaj zadatak ima za cilj automatsko određivanje emocionalnog tona izraženog u tekstualnim podacima, što omogućava razumijevanje stavova, mišljenja i raspoloženja korisnika prema određenim temama, proizvodima ili događajima.

Sentimenti se najčešće kategoriziraju kao negativni, neutralni ili pozitivni. Negativni sentimenti mogu ukazivati na nezadovoljstvo ili kritiku, dok pozitivni sentimenti izražavaju zadovoljstvo ili podršku. Neutralni sentimenti predstavljaju stavove koji nisu ni pozitivni ni negativni, već su neutralni u svom tonu. Međutim, sentimenti se također mogu stupnjevati na skali od negativnog do pozitivnog, gdje se različiti stupnjevi negativnosti i pozitivnosti mogu kvantificirati kako bi se dobila detaljnija analiza osjećaja.

Cilj ovog rada je implementirati nekoliko jednostavnih modela za klasifikaciju sentimenta teksta, koji će klasificirati tekstualne podatke u dvije kategorije: **negativni** i **pozitivni** sentiment. Ovaj jednostavni binarni pristup (0 za negativan i 1 za pozitivan sentiment) omogućava fokusiranje na temeljne tehnike strojnog učenja koje se koriste u procesu klasifikacije, čineći ga idealnim za početak istraživanja u ovom području.

2. Opis problema

Klasifikacija sentimenta teksta predstavlja izazovan problem u području obrade prirodnog jezika i strojnog učenja te ima široku primjenu u mnogim industrijama, uključujući marketing, korisničku podršku, politiku i mnoge druge. Ovaj problem je od velike važnosti zbog sve većeg volumena tekstualnih podataka generiranih na internetu, kao što su korisničke recenzije proizvoda, komentari na društvenim mrežama, blogovi, forumi, vijesti i drugi oblici pisanih sadržaja. Ručno analiziranje tih podataka nije praktično zbog njihovog volumena, stoga automatizirani sustavi za klasifikaciju sentimenta nude vrijedno rješenje.

Područja primjene

1. **Marketing i brendiranje**: Analiza sentimenta omogućava tvrtkama da prate stavove i mišljenja potrošača o svojim proizvodima i uslugama. To im pomaže u oblikovanju marketinških strategija i poboljšanju proizvoda.
2. **Korisnička podrška**: Automatsko prepoznavanje negativnih komentara može omogućiti brzu reakciju na probleme korisnika i poboljšanje njihovog iskustva.
3. **Politika**: Analiza političkih diskursa, komentara i reakcija na društvenim mrežama može pružiti uvid u javno mišljenje i pomoći u oblikovanju kampanja.
4. **Mediji i novinarstvo**: Praćenje sentimenta u vijestima i medijima može pomoći u razumijevanju javne percepcije određenih događaja ili osoba.

Postojeći pristupi

1. **Ručno kodirani pravila**: Rani sustavi za klasifikaciju sentimenta oslanjali su se na ručno kodirana pravila i leksikone koji su određivali pozitivne i negativne riječi. Ovi sustavi su imali ograničenu fleksibilnost i često nisu bili dovoljno precizni.
2. **Metode strojnog učenja**: S razvojem strojnog učenja, korišteni su algoritmi kao što su *Naive Bayes*, logistička regresija, i podržavajući vektorski strojevi (engl. *Support Vector Machine*, *SVM*) za klasifikaciju sentimenta. Ovi modeli koriste značajke izvučene iz teksta, kao što su frekvencija riječi, *n-grami* i druge statističke mjere.
3. **Duboko učenje**: Naprednije tehnike koriste duboke neuronske mreže, poput konvolucijskih neuronskih mreža (engl. *Convolutional Neural Network*, *CNN*) i rekurentnih neuronskih mreža (engl. *Recurrent Neural Network*, *RNN*), uključujući *LSTM* (engl. *Long Short-Term Memory)* mreže. Ovi modeli mogu automatski izvući značajke iz tekstualnih podataka i postići visoku točnost u klasifikaciji sentimenta.
4. **Transformerski modeli**: Nedavni napredak uključuje korištenje transformerskih modela, kao što su ***BERT*** (engl. *Bidirectional Encoder Representations from Transformers*) i ***GPT*** (engl. *Generative Pre-trained Transformer*), koji postižu vrhunske rezultate u mnogim zadacima obrade prirodnog jezika, uključujući klasifikaciju sentimenta.

2.1. Korišteni podaci

Za ovaj projekt koristimo ***SST-2,*** koji je dio većeg ***Stanford Sentiment Treebank*** skupa podataka. *SST-2* je široko korišten u istraživanjima klasifikacije sentimenta i sastoji se od tekstualnih recenzija filmova koje su označene kao pozitivne ili negativne. Ovaj podatkovni skup omogućava jednostavan binarni zadatak klasifikacije sentimenta. Podatci su preuzeti s platforme *Hugging Face.*

Metode predprocesiranja

1. **Konverzija teksta u ASCII**: Tekstualni podaci se konvertiraju u ASCII format kako bi se uklonili posebni znakovi koji nisu dio osnovnog skupa znakova.
2. **Konverzija teksta u mala slova**: Svi znakovi u tekstu se konvertiraju u mala slova kako bi se osigurala konzistentnost i smanjila složenost modela.
3. **Uklanjanje vodećih i završnih interpunkcijskih znakova**: Interpunkcijski znakovi na početku i kraju rečenica se uklanjaju kako bi se smanjio utjecaj nepotrebnih znakova na model.
4. **Uklanjanje riječi koje sadrže posebne znakove**: Riječi koje sadrže posebne znakove (npr. @, #, $, itd.) se uklanjaju iz teksta.
5. **Uklanjanje riječi koje sadrže samo jedno slovo**: Riječi koje se sastoje samo od jednog slova se uklanjaju jer obično nemaju značajno semantičko značenje.
6. **Uklanjanje *stop* riječi**: *Stop* riječi koje se često pojavljuju u engleskom jeziku, a nemaju puno semantičkog značenja (npr. "*the*", "*is*", "*at*"), se uklanjaju iz teksta.
7. **Lematizacija**: Riječi se skraćuju na njihov osnovni oblik kako bi se različite morfološke varijante riječi smanjile na isti oblik. Na primjer, riječi "*running*" i "*ran*" bi se lematizirale na "*run*".

Format zapisa podataka

1. **Tokenizacija**: Tekst se dijeli na manje jedinice zvane tokeni. Token može biti riječ, dio riječi ili čak pojedinačni znak.
2. **Pretvaranje tokena u *ID*-ove**: Svakom tokenu se dodjeljuje jedinstveni identifikacijski broj (*ID*) iz vokabulara koji model koristi.
3. **Dodavanje posebnih tokena**: Na početak i kraj svake rečenice dodaju se posebni tokeni kako bi model mogao označiti početak i kraj sekvencije.
4. ***Padding***: Kraće sekvence se popunjavaju posebnim tokenima za popunjavanje kako bi sve sekvence imale istu duljinu.
5. **Maskiranje**: Koriste se maskirajući tokeni kako bi se modelu signaliziralo koji dijelovi sekvence su stvarni podaci, a koji su dodani za popunjavanje.

2.2. Korišteni postupci strojnog učenja

Za zadatak klasifikacije sentimenta teksta u ovom radu koristili smo razne modele temeljene na ***BERT*** arhitekturi. *BERT* je napredni model obrade prirodnog jezika koji koristi transformere za razumijevanje konteksta u tekstu kroz dvosmjernu analizu. Ovo omogućava modelu da uzme u obzir kako prethodni tako i naredni kontekst za svaki token u tekstu, što ga čini vrlo učinkovitim za zadatke kao što je klasifikacija sentimenta.

Pre-trained BERT modeli

Kao početnu točku koristili smo *pre-trained BERT* modele. Ovi modeli su prethodno **trenirani na velikim korpusima tekstualnih podataka**, što im omogućava da razumiju složene jezične strukture i kontekste. *Pre-trained* modeli pružaju osnovu na kojoj se može dalje trenirati za specifične zadatke putem procesa poznatog kao *fine-tuning*.

Fine-tuning za klasifikaciju sentimenta

Fine-tuning je proces dodatnog treniranja *pre-trained* modela na specifičnom podatkovnom skupu za određeni zadatak. U našem slučaju, *fine-tuning* smo izvršili na *SST-2* skupu podataka kako bismo prilagodili *BERT* modele za zadatak binarne klasifikacije sentimenta (negativno/pozitivno).

Koraci fine-tuninga uključuju:

1. **Priprema podataka**:
    - Predprocesirani tekstualni podaci iz *SST-2* podatkovnog skupa podijeljeni su na trening, validacijski i testni skup.
    - Podaci su tokenizirani i pretvoreni u format koji BERT može koristiti (token ID-ovi, maskirajući tokeni, itd.).
2. **Postavljanje modela**:
    - *Pre-trained BERT* model je učitan i prilagođen za klasifikacijski zadatak dodavanjem dodatnog sloja (*fully connected layer*) koji će predviđati sentiment (negativan ili pozitivan).
3. **Trening modela**:
    - Model je treniran na trening skupi, koristeći optimizacijske tehnike kao što je *Adam optimizer*, i funkciju gubitka prilagođenu za binarnu klasifikaciju (engl. *binary cross-entropy loss*).
    - Tijekom treninga, hiperparametri kao što su *learning rate*, *batch size* i broj epoha su optimizirani kako bi se postigla najbolja moguća izvedba.
4. **Validacija i testiranje**:
    - Model je validiran na validacijskom skupu kako bi se pratila njegova izvedba tijekom treninga i spriječio problem prenaučenosti (engl. *overfitting*).
    - Nakon završetka treninga, model je testiran na neviđenom testnom skupu kako bi se procijenila njegova sposobnost generalizacije na nove podatke.

Evaluacija modela

Konačni model evaluiran je pomoću standardnih metrika za klasifikaciju, uključujući točnost (engl. *accuracy)*, preciznost (engl. *precision*), odziv (engl. *recall*), *F1*-mjeru, *ROC-AUC* (engl. *Area under the ROC Curve*) te empirijsku pogrešku (engl. *Loss*). Ove metrike pružaju detaljan uvid u performanse modela u klasifikaciji sentimenta i pomažu u identificiranju područja za daljnje poboljšanje.

3. Opis programskog rješenja

Cjelokupno programsko rješenje za binarnu klasifikaciju sentimenta teksta implementirano je koristeći *PyTorch* i *BERT* arhitekturu. Kroz nekoliko ključnih koraka prikazat ćemo način implementacije, od učitavanja podataka, obrade podataka, definicije modela, do treniranja, validacije i testiranja.

Učitavanje podataka

Podaci se učitavaju iz datoteka `train.tsv`, `validation.tsv` i `test.tsv` koje se nalaze u *DATA_PATH* direktoriju na *Google Drive*-u. Podaci su organizirani u obliku tablica gdje svaka datoteka sadrži rečenice i njihove pripadajuće oznake sentimenta.

```
train_dataframe = pd.read_csv(os.path.join(DATA_PATH,'train.tsv'), sep='\t', header=None, names=['labels','sentences'])
validation_dataframe = pd.read_csv(os.path.join(DATA_PATH,'validation.tsv'), sep='\t', header=None, names=['labels','sentences'])
test_dataframe = pd.read_csv(os.path.join(DATA_PATH,'test.tsv'), sep='\t', header=None, names=['labels','sentences'])
```

Obrada podataka

Podaci se pripremaju za unos u obliku ***PyTorch tensor***-a kroz klasu ***`DatasetProcessor`*** koja koristi ***tokenizer*** za transformaciju rečenica u *BERT*-ov format.

```
class DatasetProcessor(Dataset):
    def __init__(self, tokenizer, dataframe, max_sequence_length=50):
        super(DatasetProcessor, self).__init__()
        self.tokenizer = tokenizer
        self.max_sequence_length = max_sequence_length
        self.input_ids, self.attention_mask, self.token_type_ids, self.labels = self.to_tensor(dataframe)

    def to_tensor(self, dataframe):
        sentences = dataframe['sentences'].values
        labels = dataframe['labels'].values
        tokens_seq = list(map(self.tokenizer.tokenize, sentences))
        result = list(map(self.truncate_and_pad, tokens_seq))
        input_ids = [i[0] for i in result]
        attention_mask = [i[1] for i in result]
        token_type_ids = [i[2] for i in result]

        return (
            torch.Tensor(input_ids).type(torch.long),
            torch.Tensor(attention_mask).type(torch.long),
            torch.Tensor(token_type_ids).type(torch.long),
            torch.Tensor(labels).type(torch.long)
        )

    def truncate_and_pad(self, tokens_seq):
        tokens_seq = ['[CLS]'] + tokens_seq
        if len(tokens_seq) > self.max_sequence_length:
            tokens_seq = tokens_seq[:self.max_sequence_length]
        padding = [0] * (self.max_sequence_length - len(tokens_seq))
        input_ids = self.tokenizer.convert_tokens_to_ids(tokens_seq)
        input_ids += padding
        attention_mask = [1] * len(tokens_seq) + padding
        token_type_ids = [0] * self.max_sequence_length

        return input_ids, attention_mask, token_type_ids
```

Definicija modela

Model je definiran u klasi ***TransformerBinarySequenceClassificator*** koja nasljeđuje *PyTorch* klasu ***nn.Module*** te implementira njezinu metodu ***forward*** za unaprijedni prolazak podatkovnog primjera kroz mrežu. Koriste se dvije klase za automatsko izabiranje objekata modela i tokenizatora na temelju odabranog imena *pre-trained* arhitekture (***BERT*** ili slične arhitekture kao što su ***AlBERT*** i ***RoBERTa*** dostupne u *Hugging Face* biblioteci ***transformers)***.

```
class TransformerBinarySequenceClassificator(nn.Module):
    def __init__(self, model_name: str, requires_grad: bool = True) -> None:
        super(TransformerBinarySequenceClassificator, self).__init__()
        self.model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)
        self.tokenizer = AutoTokenizer.from_pretrained(model_name, do_lower_case=True)
        self.requires_grad = requires_grad
        self.device = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")

        for parameter in self.model.parameters():
            parameter.requires_grad = requires_grad

    def forward(self, batch_seqs, batch_seq_masks, batch_seq_segments, labels=None):
        if labels is not None:
            outputs = self.model(
                input_ids=batch_seqs,
                attention_mask=batch_seq_masks,
                token_type_ids=batch_seq_segments,
                labels=labels
            )
            loss, logits = outputs.loss, outputs.logits
        else:
            outputs = self.model(
                input_ids=batch_seqs,
                attention_mask=batch_seq_masks,
                token_type_ids=batch_seq_segments
            )
            loss = None
            logits = outputs.logits
        probabilities = nn.functional.softmax(logits, dim=-1)
        return loss, logits, probabilities
```

Funkcije za treniranje, validaciju i testiranje modela

```python
def train(model, dataloader, optimizer, epoch_number, max_gradient_norm):
    model.train()
    device = model.device
    epoch_start = time.time()
    batch_time_avg = 0.0
    running_loss = 0.0
    correct_preds = 0
    tqdm_batch_iterator = tqdm(dataloader)
    for batch_index, (batch_seqs, batch_seq_masks, batch_seq_segments, batch_labels) in enumerate(tqdm_batch_iterator):
        batch_start = time.time()
        seqs, masks, segments, labels = batch_seqs.to(device), batch_seq_masks.to(device), batch_seq_segments.to(device), batch_labels.to(device)
        optimizer.zero_grad()
        loss, logits, probabilities = model(seqs, masks, segments, labels)
        loss.backward()
        nn.utils.clip_grad_norm_(model.parameters(), max_gradient_norm)
        optimizer.step()
        batch_time_avg += time.time() - batch_start
        running_loss += loss.item()
        correct_preds += correct_predictions(probabilities, labels)
        description = f'\t\tAverage batch time - {batch_time_avg/(batch_index+1):.4f}s\t\tRunning loss - {running_loss/(batch_index+1):.4f}\t\t'
        tqdm_batch_iterator.set_description(description)
    epoch_time = time.time() - epoch_start
    epoch_loss = running_loss / len(dataloader)
    epoch_accuracy = correct_preds / len(dataloader.dataset)
    return epoch_time, epoch_loss, epoch_accuracy
```

```python
def validate(model, dataloader):
    model.eval()
    device = model.device
    epoch_start = time.time()
    running_loss = 0.0
    running_accuracy = 0.0
    all_prob = []
    all_labels = []
    with torch.no_grad():
        for (batch_seqs, batch_seq_masks, batch_seq_segments, batch_labels) in dataloader:
            seqs = batch_seqs.to(device)
            masks = batch_seq_masks.to(device)
            segments = batch_seq_segments.to(device)
            labels = batch_labels.to(device)
            loss, logits, probabilities = model(seqs, masks, segments, labels)
            running_loss += loss.item()
            running_accuracy += correct_predictions(probabilities, labels)
            all_prob.extend(probabilities[:,1].cpu().numpy())
            all_labels.extend(batch_labels)
    epoch_time = time.time() - epoch_start
    epoch_loss = running_loss / len(dataloader)
    epoch_accuracy = running_accuracy / len(dataloader.dataset)
    return epoch_time, epoch_loss, epoch_accuracy, roc_auc_score(all_labels, all_prob), all_prob
```

```python
def test(model, dataloader):
    model.eval()
    device = model.device
    time_start = time.time()
    batch_time = 0.0
    accuracy = 0.0
    all_probabilities = []
    all_labels = []
    with torch.no_grad():
        for (batch_seqs, batch_seq_masks, batch_seq_segments, batch_labels) in dataloader:
            batch_start = time.time()
            seqs, masks, segments, labels = batch_seqs.to(device), batch_seq_masks.to(device), batch_seq_segments.to(device), batch_labels.to(device)
            _, _, probabilities = model(seqs, masks, segments, labels)
            accuracy += correct_predictions(probabilities, labels)
            batch_time += time.time() - batch_start
            all_probabilities.extend(probabilities[:,1].cpu().numpy())
            all_labels.extend(batch_labels)
    batch_time /= len(dataloader)
    total_time = time.time() - time_start
    accuracy /= len(dataloader.dataset)
    return batch_time, total_time, accuracy, all_probabilities
```

Fine-tuning modela

Glavna funkcija ***finetune_model*** upravlja cijelim procesom *fine-tuning*-a modela, od biranja *pre-trained* arhitekture, pripreme podataka, treniranja, validacije, do testiranja modela te spremanja istog na *Google Drive* disk, zajedno za izračunatim metrikama za klasifikaciju te slikama matrice zabune (engl. *Confusion Matrix*) i *ROC* (engl. *Receiver Operating Characteristic*) krivulje.

```
def finetune_model(
        train_dataframe, validation_dataframe, test_dataframe,
        model_name: str,
        max_sequence_length: int = 50,
        epochs: int = 10,
        batch_size: int = 32,
        lr=2e-05,
        patience: int = 1,
        max_grad_norm: float = 10.0
    ):
    model = TransformerBinarySequenceClassificator(model_name=model_name, requires_grad = True)
    tokenizer = model.tokenizer
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model = model.to(device)
    save_path = os.path.join(MODELS_PATH, model_name)
    if not os.path.exists(save_path):
        os.makedirs(save_path)
    train_data = DatasetProcessor(tokenizer, train_dataframe, max_sequence_length = max_sequence_length)
    train_loader = DataLoader(train_data, shuffle=True, batch_size=batch_size)
    validation_data = DatasetProcessor(tokenizer, validation_dataframe, max_sequence_length = max_sequence_length)
    validation_loader = DataLoader(validation_data, shuffle=True, batch_size=batch_size)
    test_data = DatasetProcessor(tokenizer, test_dataframe, max_sequence_length = max_sequence_length)
    test_loader = DataLoader(test_data, shuffle=False, batch_size=batch_size)
    param_optimizer = list(model.named_parameters())
    no_decay = ['bias', 'LayerNorm.bias', 'LayerNorm.weight']
    optimizer_grouped_parameters = [
        {
            'params':[p for n, p in param_optimizer if not any(nd in n for nd in no_decay)],
            'weight_decay':0.01
        },
        {
            'params':[p for n, p in param_optimizer if any(nd in n for nd in no_decay)],
            'weight_decay':0.0
        }
    ]
    optimizer = AdamW(optimizer_grouped_parameters, lr=lr)
    scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='max', factor=0.85, patience=0)
    best_accuracy = 0.0
    start_epoch = 1
    init_validation_time, init_validation_loss, init_validation_accuracy, init_validation_auc, _, _ = validate(model, validation_loader)
    patience_counter = 0
    for epoch in range(start_epoch, epochs + 1):
        epoch_time, epoch_loss, epoch_accuracy = train(model, train_loader, optimizer, epoch, max_grad_norm)
        epoch_time, epoch_loss, epoch_accuracy , epoch_auc, _, _ = validate(model, validation_loader)
        scheduler.step(epoch_accuracy)
        if epoch_accuracy < best_accuracy:
            patience_counter += 1
        else:
            best_accuracy = epoch_accuracy
            patience_counter = 0
            test_time, test_loss, _, test_auc, all_probabilities = validate(model, test_loader)
            test_predictions = pd.DataFrame({'probability_1':all_probabilities})
            test_predictions['probability_0'] = 1 - test_predictions['probability_1']
            test_predictions['prediction'] = test_predictions.apply(
                lambda x: 0 if (x['probability_0'] > x['probability_1']) else 1,
                axis=1
            )
            test_predictions = test_predictions[['probability_0', 'probability_1', 'prediction']]
            test_accuracy, test_precision, test_recall, test_f1 = classification_metrics(test_dataframe['labels'], test_predictions['prediction'])
            model_file_path = os.path.join(save_path, 'model.pt')
            torch.save(model.state_dict(), model_file_path)
            metrics_file_path = os.path.join(save_path, 'metrics.tsv')
            with open(metrics_file_path, 'w', newline='') as file:
                writer = csv.writer(file, delimiter='\t')
                writer.writerow(['Loss', test_loss])
                writer.writerow(['Accuracy', test_accuracy])
                writer.writerow(['Precision', test_precision])
                writer.writerow(['Recall', test_recall])
                writer.writerow(['F1', test_f1])
                writer.writerow(['ROC-AUC', test_auc])
            roc_curve_path = os.path.join(save_path, 'roc_curve.png')
            fpr, tpr, _ = roc_curve(test_dataframe['labels'], all_probabilities)
            plt.figure(figsize=(5.5, 5.5))
            plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve area = %0.2f' % test_auc)
            plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
            plt.xlim([0.0, 1.0])
            plt.ylim([0.0, 1.0])
            plt.xlabel('False Positive Rate')
            plt.ylabel('True Positive Rate')
            plt.title('Receiver Operating Characteristic Curve')
            plt.legend(loc='lower right')
            plt.savefig(roc_curve_path)
            plt.show()
            plt.close()
            confusion_matrix_path = os.path.join(save_path, 'confusion_matrix.png')
            conf_matrix = confusion_matrix(test_dataframe['labels'], test_predictions['prediction'])
            fig, ax = plt.subplots(figsize=(6, 6))
            disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix)
            disp.plot(ax=ax, cmap=plt.cm.Blues)
            plt.title('Confusion Matrix')
            plt.savefig(confusion_matrix_path)
            plt.show()
            plt.close()
        if patience_counter >= patience:
            break
```

3.1. Modeli strojnog učenja

	Loss	Accuracy	Precision	Recall	F1	ROC-AUC
albert-base-v1	0.3033687885	0.8808347062	0.8949771689	0.8624862486	0.8784313725	0.9456096925
albert-base-v2	0.4285591100	0.8621636463	0.8825581395	0.8349834983	0.8581119276	0.9225073823
albert-xlarge-v1	0.3135402090	0.8753432180	0.8389662028	0.9284928493	0.8814621410	0.9519075811
albert-xxlarge-v1	0.1157202845	0.9588138386	0.9426751592	0.9768976898	0.9594813614	0.9935633914
albert-xxlarge-v2	0.2229177498	0.9236683141	0.9087048832	0.9416941694	0.9249054565	0.9783910409
bert-base-cased	0.3892284415	0.8978583196	0.8777429467	0.9240924092	0.9003215434	0.9658018982
bert-base-uncased	0.2422277712	0.908841296	0.9077936334	0.9097909791	0.9087912088	0.9680051338
bert-large-cased	0.2026091338	0.9220208677	0.9182115594	0.9262926293	0.9222343921	0.9774742825
bert-large-uncased	0.4290135367	0.8451400329	0.8208802456	0.8822882288	0.8504772004	0.9165110590
camembert-base	0.4074957415	0.8286655684	0.7895247333	0.8954895490	0.8391752577	0.9129055449
distilroberta-base	0.2080812189	0.9159802306	0.9324942792	0.8965896590	0.9141895681	0.9747228012
roberta-base	0.1569838062	0.9379461834	0.9335511983	0.9427942794	0.9381499726	0.9855996565
roberta-large	0.3372861563	0.9033498078	0.8751279427	0.9405940594	0.9066808059	0.9482302945
xlm-roberta-base	0.3082347063	0.9104887424	0.8877338877	0.9394939494	0.9128808124	0.9657005723
xlm-roberta-large	0.1854558185	0.9308072488	0.9142857143	0.9504950495	0.9320388350	0.9817299712
xlnet-base-cased	0.2416777974	0.9154310818	0.8771228771	0.9658965897	0.9193717277	0.9828530002
xlnet-large-cased	0.1894222637	0.9483800110	0.9522752497	0.9438943894	0.9480662983	0.9878975836

Najbolji modeli:

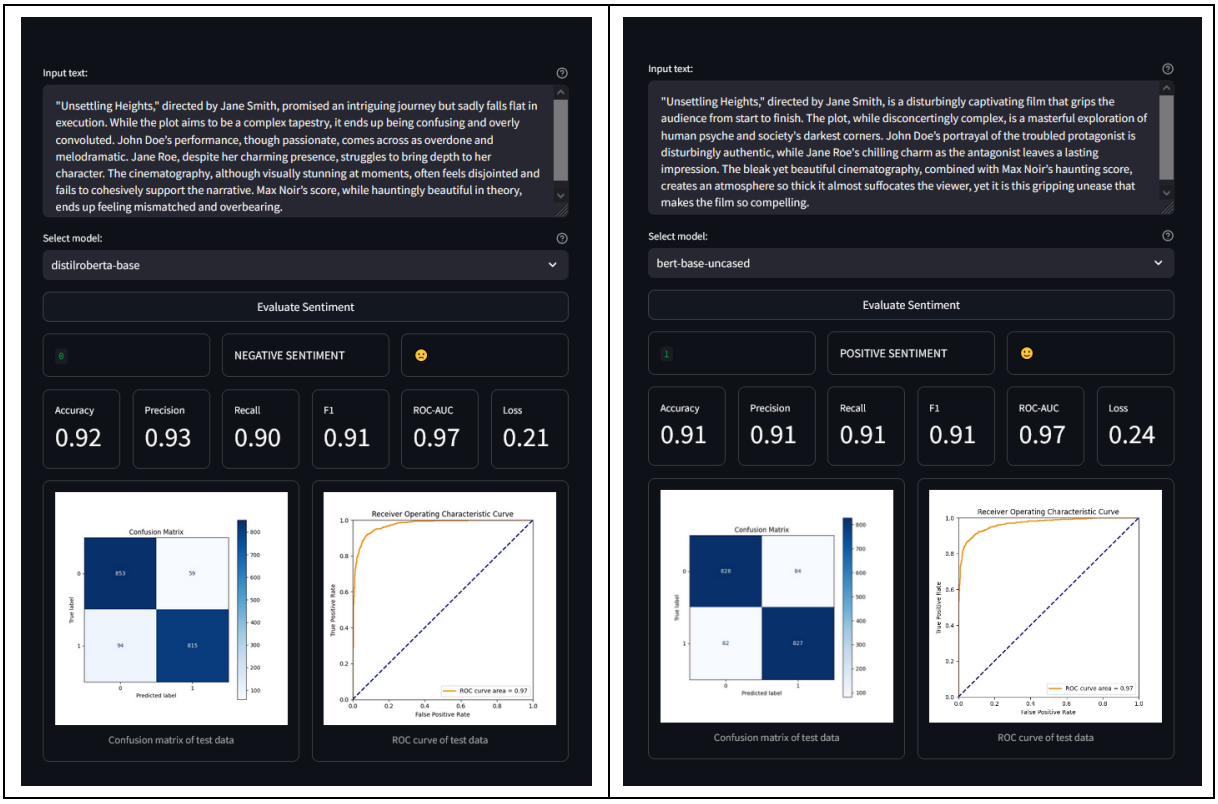
I. *albert-xxlarge-v1*
II. *xlnet-large-cased*
III. *roberta-base*
IV. *xlm-roberta-large*
V. *distilroberta-base, xlnet-base-cased*

Najlošiji modeli:

I. *camembert-base*
II. *bert-large-uncased*
III. *albert-base-v2*
IV. *albert-base-v1*
V. *albert-xlarge-v1, bert-base-cased*

3.2. Klijentska aplikacija

Prikaz aplikacije koja rabi istrenirane modele strojnog učenja, riječ je o web rješenju koje koristi platformu *Streamlit*. Vidimo da za unos teksta zamišljene recenzije filma, modeli pravilno prepoznaju sentiment teksta čak i kada namjerno “ubacimo” negativne riječi u pozitivan tekst i pozitivne riječi u negativan tekst kako bi pokušali zbuniti model da nam da krivo rješenje. Naime ovi modeli “razumiju” jezik, i očito im je da se ove naizgled negativne i pozivne riječi samo tako čine kada iz se gleda izvan konteksta (budući da se radi o prirodnom jeziku, jako je bitno da model zna prepoznati razna značenje istih riječi u drugom kontekstu). Dakle, istrenirani modeli su dovoljno “pametni” da znaju da čak i riječi koje se čine “suprotnog sentimetna” imaju drugačije semantičko značenje ovisno u kontekstu kojem se nalaze, a to je dobar pokazatelj da ovi klasifikatori imaju dobra generalizacijska svojstva.



4. Zaključak

U ovom radu istražili smo klasifikaciju sentimenta teksta koristeći različite modele strojnog učenja, s posebnim naglaskom na *pre-trained BERT* modele i njihovu prilagodbu (*fine-tuning*) za specifične zadatke klasifikacije. Kroz detaljnu analizu i implementaciju, pokazali smo da modeli poput *BERT*-a imaju izvrsnu sposobnost generalizacije i prepoznavanja semantičkog konteksta, što je ključno za točne klasifikacije sentimenta. Ovi modeli ne samo da prepoznaju pozitivne i negativne emocije, već mogu precizno detektirati i suptilne nijanse u tekstu.

Primijenjene metode obrade prirodnog jezika, uključujući predprocesiranje podataka, tokenizaciju i normalizaciju, omogućile su bolje razumijevanje teksta i poboljšale performanse modela. Tokom istraživanja, pokazalo se da je pravilno predprocesiranje podataka ključno za postizanje visokih performansi. Evaluacija na različitim skupovima podataka pokazala je visoku točnost i robusnost modela, čak i u prisutnosti riječi koje na prvi pogled mogu izgledati kontradiktorno sentimentu teksta. Različiti scenariji testiranja potvrdili su da ovi modeli zadržavaju konzistentnu učinkovitost.

Zaključno, ovaj rad potvrđuje učinkovitost naprednih modela strojnog učenja za klasifikaciju sentimenta i naglašava važnost pravilnog predprocesiranja podataka te prilagodbe modela za specifične zadatke. Osim tehničkih aspekata, naglašene su i praktične implikacije ovih metoda. Nadalje, praktična implementacija i razvoj web aplikacije pomoću platforme *Streamlit* demonstriraju primjenjivost ovih modela u stvarnom svijetu, omogućujući korisnicima jednostavnu i intuitivnu analizu sentimenta tekstualnih podataka. Ovaj rad također otvara vrata budućim istraživanjima u području analize sentimenta, pružajući čvrstu osnovu za daljnje unapređenje metoda i alata.

5. Poveznice i literatura

Programsko rješenje na GitHub-u
Web rješenje na Streamlit-u

[1] K.P.Murphy, Machine Learning: A probabilistic perspective, MIT Press, Cambridge, Massachusetts, SAD, 2012

[2] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

[3] Jurafsky, D., & Martin, J. H. (2019). *Speech and Language Processing* (3rd ed.). Pearson.

[4] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

[5] Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

[6] Goldberg, Y. (2017). *Neural Network Methods for Natural Language Processing*. Morgan & Claypool.

[7] Hugging Face Transformers: https://huggingface.co/docs/transformers/en/index

[8] Pytorch Docs: https://pytorch.org/docs/stable/index.html

[9] Streamlit Docs: https://docs.streamlit.io/