Coneixement Raonament i Incertesa

Pràctica 3: Tweet Classifier

ÍNDEX

Introducció

El día 5 de desembre vaig començar la tercera pràctica de CRI en la cual l'objectiu és aplicar Naive Bayes per poder predir si un Tweet és positiu o negatiu

Els objectius de la pràctica són:

- Aplicar l'aprenentatge bayesià per a la classificació de tweets segons el seu sentiment
- Utilitzar tècniques de validació per avaluar la precisió del model
- Demostrar la capacitat de dissenyar i implementar xarxes bayesianes en contextos reals
- Analitzar i justificar l'impacte de diferents mides de dades i diccionaris
- Presentar resultats de manera clara i defensar les decisions metodològiques

Solució proposada

Inicialment partim d'una base de dades que conté aproximadament 1,5 milions de tweets on a cada fila hi ha 4 elements, el primer és l'identificador del tweet, la segona es el text escrit, la tercera és l'hora en què s'ha escrit el missatge i finalment la cuarta i última columna tenim el target en aquest cas tenim un valor int binari que si es 1 significa que el missatge és positiu en el cas contrari un 0 per un missatge negatiu.

16; I fell in love again; 02/12/2015; 1

Així llegeixo la base de dades ignorant l'hora i data i l'identificador perquè considero que no son dades rellevants, ja que lo més important es el text i el target.

```
def open_fitxer():
    df = pd.read_csv("FinalStemmedSentimentAnalysisDataset.csv", delimiter=';')
    df = df.dropna(how='any')  # eliminar files amb valors nuls
    numpy_df = df.to_numpy()
    sentiment_labels = df.iloc[:, 3]

    # Comptem els valors 1 i 0
    num_positius = (sentiment_labels == 1).sum()
    num_negatius = (sentiment_labels == 0).sum()

    # Mostrem el resultat
    print(f"nombre de tweets positius 1: {num_positius}")
    print(f"nombre de tweets negatius0: {num_negatius}")

    X = numpy_df[:, 1]
    Y = numpy_df[:, 3]
    Y = Y.astype('int64')
    return X, Y
```

Com es pot observar retorno X i Y que en la variable X es guarda tota la columna de tweets i en la Y es guarda tota la columna de targets, en aquest cas faig una conversió a int64 per la columna target.

Per poder veure quines mètriques poden ser les més útils per poder avaluar el model d'entrenament si és bo fent prediccions dels tweet he imprimit per pantalla lo que seria la quantitat de tweets positius i negatius per poder veure si la base de dades esta desbalançejada, com que la base de dades esta balancejada com es pot observar en la imatge inferior la métrica més útil en aquest cas sería l'accuracy ja que vull saber que tan bo és el model en general.

```
nombre de tweets positius 1: 781636
nombre de tweets negatius0: 782644
```

Un cop tinc totes les dades necessàries per poder començar l'entrenament lo que faig es declarar una funció per començar a generar diccionaris on agafo cada paraula de la BDD i conto quantes vegades ha sortit la paraula en tweets positius i negatius que això ens servira més endavant per començar a fer les prediccions.

```
def generate_dictionary(X_train, Y_train):
    word_freq = defaultdict(lambda: {'positive': 0, 'negative': 0})
    for text, label in zip(X_train, Y_train):
        for word in text.split():
            if label == 1:
                word_freq[word]['positive'] += 1
            else:
                word_freq[word]['negative'] += 1
    return word_freq
```

Continuadament faig la funció predict classifica un tweet com a positiu 1 o negatiu 0 basant-se en un model de probabilitats bayesianes. Rep com a paràmetres un tweet (cadena de text), un diccionari amb comptatges de paraules per categoria (dictionary), el total de paraules positives (total_pos) i negatives (total_neg), i la mida total del vocabulari (vocab_size). Primer, calcula les probabilitats inicials per a les classes positiva i negativa utilitzant els totals proporcionats. Després, recorre cada paraula del tweet, actualitzant les probabilitats segons els comptatges associats al diccionari, aplicant la funció logarítmica per evitar subdesbordaments numèrics. Les paraules que no són al diccionari es descarten. Finalment, retorna 1 si la probabilitat positiva és superior a la negativa, i 0 en cas contrari.

```python
def predict(tweet, dictionary, total_pos, total_neg, vocab_size):
    pos_prob = np.log(total_pos / (total_pos + total_neg))
    neg_prob = np.log(total_neg / (total_pos + total_neg))

    for word in tweet.split():
        if word in dictionary:
            pos_count = dictionary[word]['positive']
            neg_count = dictionary[word]['negative']

            if pos_count > 0:
                pos_prob += np.log(pos_count / total_pos)
            if neg_count > 0:
                neg_prob += np.log(neg_count / total_neg)
        # Si la palabra no está en el diccionario, se ignora.

    return 1 if pos_prob > neg_prob else 0
```

Per completar la part C, s'utilitza el mètode de validació creuada K-Fold amb 5 particions per avaluar el rendiment del model. A cada iteració, es divideixen les dades en entrenament i prova, es genera un diccionari de paraules a partir del conjunt d'entrenament, i es calcula la frequència de termes positius i negatius. Aquest diccionari, juntament amb estadístiques com la mida del vocabulari, s'utilitza per predir les etiquetes del conjunt de prova. Les prediccions es comparen amb les etiquetes reals mitjançant l'accuracy, que es calcula per cada partició i s'emmagatzema. Al final, es computa l'accuracy mitjana entre les 5 iteracions per obtenir una estimació global del rendiment. Aquest procés assegura una avaluació robusta i equitativa del model en tot el conjunt de dades.

```python
# lectura del dataset
X, Y = open_fitxer()

# configuracio del KFold
k = 5  # Nombre de plegaments
kf = KFold(n_splits=k, shuffle=True, random_state=42)

accuracies = []  # Per emmagatzemar l'accuracy de cada iteraci

for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    Y_train, Y_test = Y[train_index], Y[test_index]

    # generar diccionari
    dictionary = generate_dictionary(X_train, Y_train)

    # Comptar paraules positives i negatives
    total_pos = sum([freq['positive'] for freq in dictionary.values()])
    total_neg = sum([freq['negative'] for freq in dictionary.values()])
    vocab_size = len(dictionary)

    # prediccions
    predictions = [predict(tweet, dictionary, total_pos, total_neg, vocab_size) for tweet in X_test]

    # Avaluaci
    accuracy = accuracy_score(Y_test, predictions)
    accuracies.append(accuracy)
    print(f"Fold Accuracy Part C: {accuracy}")

# accuracy mitjana
mean_accuracy = np.mean(accuracies)
print(f"Mean Accuracy across {k} folds Part C: {mean_accuracy}")
```

Resultats Part C

Resultat mètriques obtingudes:

```
Fold Accuracy Part C: 0.6950482010893191
Fold Accuracy Part C: 0.6958536834837753
Fold Accuracy Part C: 0.6969372490858414
Fold Accuracy Part C: 0.6963970644641624
Fold Accuracy Part C: 0.6955660112000409
Mean Accuracy across 5 folds Part C: 0.6959604418646278
```

Conclusions dels resultats

En cada iteració del procés d'entrenament amb els subconjunts, s'imprimeix l'accuracy corresponent per monitoritzar el rendiment parcial del model. Un cop completades totes les iteracions, es calcula la mitjana de les accuracies obtingudes en cadascun dels subconjunts d'entrenament i validació. Aquesta mitjana proporciona una estimació global i més fiable del rendiment del model en tot el conjunt de dades, evitant esbiaixos derivats d'una única partició.

Preguntes i Respostes:

• Generació de diccionaris (estructura i contingut). Com genereu el/s diccionari/s?
Es genera 1 diccionari que te 2 components una negativa i l'altra positiva, es a dir per a cada paraula es guarda la cuantitat de vegades que ha sortit la paraula en un tweet negatiu o positiu.

• Si hi ha múltiples diccionaris, perquè i com?
En el meu cas només em fa falta generar 1 sol diccionari amb 2 components una negativa i una altra positiva per cada paraula.

• Justificació del mètode de validació (cross-validation, leave-one-out, etc.)
He utilitzat el metode de test split pero em donava accuracies més baixes i he acabat utilitzant KFold.

• Justificació de la mètrica?
La métrica més important per poder avaluar el nostre model es l'accuracy ja que mesura que tant bó es el model en general, a més a més, es molt útil en aquesta BDD ja que les dades estan balancejades

• Resultats i anàlisi
En general el model té un accuracy promig d'aproximadament un 70% de les vegades sense utilitzar Laplace Smoothing i considero que prediu bastant bé.

Part B

En l'apartat B em demana utilitzar varies estrategies per poder veure com funciona el model depenen de limitar el diccionari o limitan el conjunt de train o combinar les 2 coses.

Primera Estrategia

La primera estrategia que utilitzo es la de limitar el conjunt d'entrenament

```
############################  PART B ############################
print("Part B")

# estrategia 1: ampliar conjunto de entrenamiento
def experiment_strategy_1(X, Y):
    train_sizes = [100, 200, 300]  # Reducido para simplificar
    results = []

    for train_size in train_sizes:
        X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=train_size, random_state=42)
        dictionary = generate_dictionary(X_train, Y_train)

        total_pos = sum([freq['positive'] for freq in dictionary.values()])
        total_neg = sum([freq['negative'] for freq in dictionary.values()])
        vocab_size = len(dictionary)

        predictions = [predict(tweet, dictionary, total_pos, total_neg, vocab_size) for tweet in X_test[:200]]
        accuracy = accuracy_score(Y_test[:200], predictions)
        results.append((train_size, accuracy, vocab_size))

    return results
```

Aqui lo que faig es limitar el conjunt d'entrenament a 100, 200 i 300 ho faig bastant petit, perquè vull reduir el temps d'execució i mostri resultats molt més ràpid.

```
estrategia 1: ampliar elconjunt dentrenament [(100, 0.36, 540), (200, 0.43, 999), (400, 0.415, 1694)]
```

(Primer s'imprimeix el train_size, després l'accuracy i finalment el tamany del vocabulari)

Com es pot veure a mesura que augmentem el conjunt d'entrenament al doble en cada iteració més precís és el model per poder predir si el tweet és positiu o negatiu, lo més probable es que si hagués utilitzat més un conjunt d'entrenament més alt es podría dir que també sería més precís.

Segona Estrategia

La Segona Estrategia que he utilitzat és limitar el tamany del diccionari i vaig fent lo mateix en cada iteració pero augmentant el doble el tamany.

```
# çestrategia 2: modificar tamany del diccionari
def experiment_strategy_2(X_train, Y_train, X_test, Y_test):
    dictionary = generate_dictionary(X_train, Y_train)
    sorted_dict = sorted(dictionary.items(), key=lambda item: sum(item[1].values()), reverse=True)
    dictionary_sizes = [50, 100, 200]  # Más pequeño para simplificar
    results = []

    for size in dictionary_sizes:
        reduced_dict = dict(sorted_dict[:size])

        total_pos = sum([freq['positive'] for freq in reduced_dict.values()])
        total_neg = sum([freq['negative'] for freq in reduced_dict.values()])
        vocab_size = len(reduced_dict)

        predictions = [predict(tweet, reduced_dict, total_pos, total_neg, vocab_size) for tweet in X_test[:200]]
        accuracy = accuracy_score(Y_test[:200], predictions)
        results.append((size, accuracy))

    return results
```

```
estrategia 2: modificar mida del diccionari [(50, 0.675), (100, 0.66), (200, 0.545)]
```

Com es pot veure en els prints d'aquesta imatge com més augmenta el tamany del diccionari més baix es l'accuracy del model es degut al ser tant petit el diccionari s'introdueixen noves paraules que tenen tan poca frecuencia que fa que el model no pugui predir bé.

```
estrategia 2: modificar mida del diccionari [(5000, 0.478), (10000, 0.4811), (20000, 0.48115)]
```

Com es pot veure aquí si augmenta per 10 les mides dels diferents diccionaris, com més augmenta la mida augmenta lleugerament l'accuracy.

Tercera Estrategia

Finalment la tercera estrategia es fixar el tamany del diccionari però anar canviant el tamany del conjunt d'entrenament.

```
# estrategia 3: modificar conjunt dentrenamiento amb diccionari fixe
def experiment_strategy_3(X, Y, fixed_dict_size=10000):
    train_sizes = [1000, 2000, 4000]  # Reducido para simplificar
    results = []

    for train_size in train_sizes:
        X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=train_size, random_state=42)
        dictionary = generate_dictionary(X_train, Y_train)
        sorted_dict = sorted(dictionary.items(), key=lambda item: sum(item[1].values()), reverse=True)
        reduced_dict = dict(sorted_dict[:fixed_dict_size])

        total_pos = sum([freq['positive'] for freq in reduced_dict.values()])
        total_neg = sum([freq['negative'] for freq in reduced_dict.values()])
        vocab_size = len(reduced_dict)

        predictions = [predict(tweet, reduced_dict, total_pos, total_neg, vocab_size) for tweet in X_test[:1000]]
        accuracy = accuracy_score(Y_test[:1000], predictions)
        results.append((train_size, accuracy))

    return results
```

```
estrategia 3: modificar conjunt dentrenament amvb diccionari fixe [(1000, 0.473), (2000, 0.507), (4000, 0.546)]
```

Com es pot veure en els resultats com més augmenta la mida del train size mantenint el diccionari fixe més augmenta la precisió de manera molt aguda.

Preguntes i Respostes

- **Què passa en el primer cas, quan es va ampliant el nombre de tweets i el diccionari per fer el train?**
  A mesura que s'amplia el conjunt d'entrenament va augmentant l'accuracy progressivament.
- **Com afecta la mida del diccionari?**
  Com més paraules afegeixo en el diccionari l'accuracy creix molt lleugerament.
- **Com afecta la mida del conjunt de tweets de test?**
  No afecta molt ja que al final lo que importa és com es fan les prediccions ja que si les prediccions es fan bé coincidiràn amb el conjunt de tests.

Part A

1. Aplicació de Laplace Smoothing

Lo que he fet per l'apartat A ha sigut reutilitzar el codi de l'apartat C per aquest cop aplicant Laplace Smoothing a la funció predict en aquest cas li afegeixo un paràmetre alpha a la funció.

```
def predictLaplaceSmoothing(tweet, dictionary, total_pos, total_neg, vocab_size, alpha=1):
    pos_prob = np.log(total_pos / (total_pos + total_neg))
    neg_prob = np.log(total_neg / (total_pos + total_neg))

    for word in tweet.split():
        # usar valores por defecto si la palabra no está en el diccionario
        pos_count = dictionary[word]['positive'] if word in dictionary else 0
        neg_count = dictionary[word]['negative'] if word in dictionary else 0

        pos_prob += np.log((pos_count + alpha) / (total_pos + alpha * vocab_size))
        neg_prob += np.log((neg_count + alpha) / (total_neg + alpha * vocab_size))

    return 1 if pos_prob > neg_prob else 0
```

Si alpha es superior a 1 en cas de que ens trobem amb alguna palabra que tingui frecuencia 0 la funció no ha de passar per ifs previs per evitar desbordaments numerics cuan opero amb logaritmes.

2. Resultats

En aquest cas també per entrenar el model he utilitzat el Kfold dividint la BDD en 5 subconjunts i en aquest cas al aplicar Laplace Smoothing ha augmentat l'accuracy bastant passant de aproximadament 70% a un 77% com es pot observar en les següents imatges.

```
Fold Accuracy Part C: 0.6950482010893191
Fold Accuracy Part C: 0.6958536834837753
Fold Accuracy Part C: 0.6969372490858414
Fold Accuracy Part C: 0.6963970644641624
Fold Accuracy Part C: 0.6955660112000409
Mean Accuracy across 5 folds Part C: 0.6959604418646278
```

(Accuracy sense aplicar Laplace Smoothing)

```
Fold Accuracy Part A: 0.7757274912419772
Fold Accuracy Part A: 0.7766160789628456
Fold Accuracy Part A: 0.7774950776075894
Fold Accuracy Part A: 0.7761717851024114
Fold Accuracy Part A: 0.7762165341243256
Mean Accuracy across 5 folds Part A: 0.7764453934078298
```

(Accuracy aplicant Laplace Smoothing a la funció predict)

3. Preguntes i Respostes

- **Com afecta el 'Laplace smoothing?**
    - Al aplicar Laplace smoothing com s'ha explicat prèviament augmenta la precisió del model i també fa que en cas de que la frecuencia sigui 0 no hi hagi problemes de desbordament al operar amb logaritmes ja que hi ha una component alpha que farà que el numerado mai sigui 0.

Problemes trobats durant el projecte

1. **Desbordaments**
   Al principi tenía bastants problemes a la funció predict sense aplicar Laplace Smoothing ja que si una paraula mai ha estat en el diccionari cuan hem de calcular la freqüència de la paraula em saltaven errors per el logaritme ja que no pot operar amb un 0 en el numerador.

2. **Augmentar mida del diccionari**
   En l'apartat B en la segona estrategia on es tenía que canviar la mida del diccionari a mesura que augmentava la mida més baixava l'accuracy al principi no sabía perque pasava i m'he donat compte que el diccionari era tan petit que encara que incrementes el tamany feia que surtin paraules que havien sortit molt poques vegades o no havien sortit mai i això feia baixar l'accuracy. Però augmentant les mides que utilitzaba per 10 es pot observar que si s'augmenta la mida va creixent progressivament la precisió del model.

Conclusions i treballs futurs

Amb aquest projecte he après a aplicar els meus coneixements en aprenentatge Bayesià per poder fer coses tan útils com poder determinar si un tweet es negatiu o positiu i gràcies a aquesta pràctica els meus coneixements es poden aplicar en camps tan importants com el de Data Science etc.

1. Moderació de Xarxes Socials

- Identificar comentaris o texts que violin les normes en una Xarxa Social
- Preveure tendències de moda i poder mostrar productes.

2. Marketing

- Mostrar anuncis personalitzats als usuaris.
- Ajustar estratègies de publicitat.

3. Finances

- Preveure com varien els mercats.

4. Industria

- Poder saber cuan s'espatllarà una màquina.
- Anàlisis de durabilitat de productes.