Sharif University of Technology

Machine Learning Project Report

Professor Motahari

Mehd Lotfian

99105689

Introduction to Machine Learning Course

Spring 2023

1 Project Explanation

In this project we used a data consisting of people's comments about different movies to conclude their sense about the movie (between 0 being totally negative and 4 being totally positive).

2 Overall Method

At first, we try to tokenize the comments to their words and then convert the words to a matrix X which can be used for training different models. and use the the Sentiments as labels for the classification task at hand.

We consider to approaches for converting word to vectors in order to be able to make the matrix X:

TF-IDF

In this approach, we consider Term Frequency (TF) which means how many times is the word used in the sentence and Inverse Document Frequency (IDF) which means the inverse of how many times the word is used in the whole context (here means all the comments). In this approach each comment, will be remade by an array with the length of all the words in the vocabulary. As each sentence is only consist of a few words, the matrix X would be a very large and sparse matrix which will use up a lot of memory and time to train an evaluate.

Fast text

In this approach, each word is mapped to a vector where the similar words have less distance in vectors measures and reverse. In this approach, we can show each sentence by it's corresponding words' vectors by some functions like calculating means or … This approach will use up less space and needs less time to be trained but it's less accurate in specifying each sentence, in oppose to the previous approach.

3 Used Packages

Sklearn

We used different methods from SKlearn library to build and train different classifiers like logistic regression, decision trees and multinomial naïve bayes. We also used this library to split the data in to train, test and validation (If needed) parts in order to be able to evaluate different models. We also used different metrics from this library like confusion matrix, F1- score, precision and recall to evaluate each model.

Torch

We used this library different methods to build a neural network for training the data in order to reach better performances as opposed to machine learning classical algorithms.

nltk

This package was used to perform preprocessing such as removing stop word, punctuations and normalizing data in order to perform better on the learning and predicting

fasttext

This package was used to perform word to vector as explained in the previous part.

pandas

We used this package to build the data frame from the data and manipulate the data with its function to be able to perform better on the training and predicting.

numpy

We used this package to make X and y for the training and testing parts/

tqdm

We used this package to be able to see the progress on long runs.

other packages like os and string were used briefly for their few usages in saving, loading data and model and manipulating the text.

4 Feature Extraction in Second Approach

```
class FastText:

    def __init__(self, preprocessor=None, method='skipgram'):
        self.method = method
        self.model = None
        self.preprocessor = preprocessor

    def train(self, texts):
        """
        train the fasttext model and save it into self.model
        Parameters
        ----------
        texts: list of list of str
        """
        self.model = fasttext.train_unsupervised('my_text.txt',
model=self.method, dim=100)

    def get_query_embedding(self, query, tf_idf_vectorizer):
        """
        get the embedding of the query. You can use the
tf_idf_vectorizer to get the weights of the words in the query.
preprocess the query using self.preprocessor if it is not None
        Parameters
        ----------
        query: str
        tf_idf_vectorizer: TfidfVectorizer
        Returns embedding of the query
        """
        query_embed = self.model.get_sentence_vector(query)
        return query_embed

    def save_FastText_model(self, path='FastText_model.bin'):
        self.model.save_model(path)

    def load_FastText_model(self, path="FastText_model.bin"):
        self.model = fasttext.load_model(path)

    def prepare(self, dataset, mode, save=False):
        if mode == 'train':
            self.train(dataset)
        if mode == 'load':
            self.load_FastText_model()
        if save:
            self.save_FastText_model()
```

The above class was defined to use fast text as a feature extraction in second approach as explained in the second part. we also used a saver and loader for

the model to avoid spending time training the model every time the colab crashes or refreshes.

```
embeddings = []
for phrase in phrases_processed:
  embeddings.append(FastText_model.model.get_sentence_vector(phrase))
```

In the end we make embeddings array which is consists of each comment array by giving the comments one by one to our built model (using above class) and appending the result to embeddings list which will be converted to X matrix.

5 Used models

Multinomial Naïve Bayes

In this model we consider each feature independent from others and use Bayesian rule to calculate the probability of each data belonging to different classes. The parameters are very smaller oppose to normal bayes.

Logistic Regression

In this model, we use a linear regression to split the space in to various parts and giving each data a label according to its position on the space. It's parameters are like simple linear regression which is about the same size as number of the feature which can reduce the chance of overfitting.

Decision Tree Classifier

In this model we split the data multiple times based on its entropy or gini index in order to make pure spaces where in each space, labels are the same. of course 100 purity will lead to overfitting so we tune the hyperparameter max_length to find the best decision tree model.

Neural Net Classifer

A neural net classifer using torch was built to train on the data and predict the labels which leaded to error and crached google colab.

Other Classical models like SVM (Support Vector Machines), Perceptron Classifier and Random Forest were tried but lead to google colab crash which could not be used to compare.

6 Comparing Models

At last, we succeeded to train and evaluate 6 models (3 models with each approach so we can compare their result)

The logistic regression with both first and second approaches leaded to better result as it was a simple model consisting few free parameters.

Multinomial Naïve Bayes leads to an acceptable result in the first approach (better than other models except logistic regression on first approach) which can be concluded that it's because of its simplicity to other models after logistic regression.

Decision tree classifier results on both approaches and Multinomial Naïve Bayes leaded to unacceptable results with high errors which can be due to their complexity.

Another cause of better results in logistic regression is due to its class imbalance handling which was needed seriously in our data.

7 Comparing Classic and Modern Approaches (1st and 2nd)

As was explained in the previous parts, TF-IDF which is a simpler way of converting words to array, will consume lots of memory and time, and the resulting matrix will be sparse which can be saved in less memory than non-sparse matrices using new saving methods, but still consumes a lot of space and doesn't consider the meaningful relationship between data.

On the other hand, Fasttext uses meaningful relations to build a vector for each words according to its similarities and differences from other words (these measures can change through different methods which leads to different vectors). For this approach to work probably, we need different words and sentences in order to build better relations.

As it can be concluded from the data, the new sentences are not enough and most of the data is just a copy of a part of other sentences which leads to incomplete dictionary and Fast text cannot be trained on the given data properly which leaded to less accurate results.

8 Conclusion

As was discussed in the previous sections, according to the data which is simple and imbalance, consisting of an incomplete dictionary, models with less complexity and less free parameters to be trained are likely to perform better and leads to better accuracy. Logistic Regression which is a linear model consisting of a few free parameters (the size of features) leaded to best accuracy (about 64%) on the data using TF-IDF as a comment to vector convertor which is simpler than Fast Text and needs less new vocabularies to be trained.