Predicting Left/Right-Wing Voter Preference Using Socioeconomic Features

Introduction

Sentiment analysis, also known as opinion mining, is the process of analyzing text to determine the emotional tone or sentiment it expresses (1). In practice, this usually means classifying a piece of text as having positive, negative, or sometimes neutral sentiment. Sentiment analysis is widely used in industry to automatically gauge public opinion and customer satisfaction by mining large volumes of reviews, social media posts, and other user-generated content. By understanding whether customers feel positive or negative, organizations can make informed decisions to improve products, services, and reputation.

In this project, I will focus on a simple binary sentiment analysis task: classifying short English texts as positive or negative. This task is a common introduction to Natural Language Processing (NLP) and machine learning because it is intuitive and has many real-world applications (for example, determining if a movie review is favorable or not). Our motivation is to demonstrate the full pipeline of an NLP classification project on a real-world dataset, from data collection and preprocessing to model training and evaluation. I will build two basic models: a logistic regression classifier and a feedforward neural network (FNN) and compare their performance on the sentiment prediction task. Logistic regression is a strong baseline for text classification due to its simplicity and effectiveness, while a feedforward neural network represents a basic deep learning approach that can capture non-linear relationships. By comparing these models, I can learn about their relative strengths and understand how a neural network might improve (or not improve) results on a beginner-level task.

To study sentiment analysis, I needed a real-world, publicly available dataset of short texts with sentiment labels. For this project, I selected the Sentiment Labelled Sentences Data Set from the UCI Machine Learning Repository (2). This dataset contains 3,000 short sentences extracted from online reviews, and each sentence is labeled as **1** (positive sentiment) or 0 (negative sentiment) (3). The sentences come from three different real sources: 1,000 from IMDb movie reviews, 1,000 from Amazon product reviews, and 1,000 from Yelp restaurant reviews. In each source, there are 500 positive and 500 negative examples, for a balanced set of opinions. These sentences were selected such that they have a clearly positive or negative connotation. Because the texts are actual user reviews, this dataset provides a realistic and beginner-friendly benchmark for sentiment classification.

The Sentiment Labelled Sentences dataset was originally compiled by researchers Dimitrios Kotzias et al. (2015) as part of a study on sentiment classification. It is freely available under a Creative Commons Attribution 4.0 license. We obtained the dataset from the UCI repository and used it directly for our experiments. Despite its simplicity, it reflects real customer opinions, which makes our experiments realistic. In summary, this data set provided a convenient and appropriate starting point to build and evaluate sentiment analysis models, and its open availability means anyone can download and replicate our work.

Before training our models, I performed several data preprocessing and feature engineering steps to convert the raw text into a suitable numerical form for machine learning. Our goal was to clean the text data and then apply a bag-of-words representation so that each sentence becomes a vector of features. The main preprocessing and feature extraction steps are outlined below:

Combine and shuffle data: I combined the sentences from all three sources into one dataset and assigned their respective sentiment labels. The combined data was then randomly shuffled and split into a training set and a test set. This split ensures we can train models and later evaluate how well they generalize to unseen data.

Text cleaning: I cleaned and normalized the text in each sentence. I converted all letters to lowercase, and I removed punctuation and special characters that are not relevant to sentiment. I retained standard alphanumeric characters and spaces, effectively keeping the words themselves. I decided not to remove common stop words in this task, because words like “not” or “never” can crucially change the sentiment. By keeping these words, I preserve important context for sentiment.

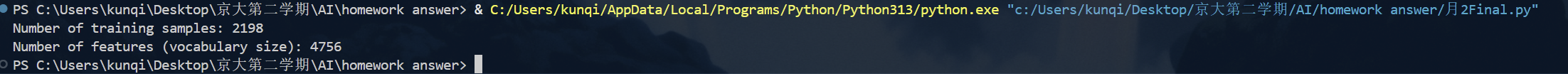
Feature extraction: After cleaning the text, I transformed each sentence into a numerical feature vector using a bag-of-words model. In a bag-of-words representation, I first build a vocabulary of all words occurring in the training set. Then each sentence is represented by features that indicate how many times each word appears in that sentence. For our model, I used a slightly more refined version called TF-IDF (Term Frequency–Inverse Document Frequency) weighting. TF-IDF assigns higher value to a word if it appears frequently in a particular sentence but not too frequently across all sentences. This way, common words like “the” or “is” (which appear in almost every sentence) are down weighted, while more distinctive words (like “awesome” or “terrible”) get higher weights. Using TF-IDF, each sentence became a vector in a high-dimensional space (with one dimension per word in the vocabulary) indicating the importance of each word in that sentence.

Data format for modeling: The outcome of the above steps is that our training set is transformed into a matrix of feature vectors (one for each sentence) and a corresponding array of labels (1 or 0 for each sentence). The test set is likewise transformed into feature vectors using the same vocabulary and scaling. These numerical feature vectors are the inputs that our machine learning models can accept. For example, a sentence like "I love this product" might be converted into a vector where the positions corresponding to words "love", "product" (and others in the vocabulary) have relatively high values. At this point, the text data is ready for model training and evaluation.

With the data cleaned and represented as numeric features, I can now proceed with building our classification models. Below I show the Python code that performs the above preprocessing steps, and then we describe the models and experiments.

文本

AI 生成的内容可能不正确。



I trained two different binary classification models on the preprocessed data: a Logistic Regression model and a Feedforward Neural Network. Both are supervised learning models that take the feature vectors (from the bag-of-words representation of sentences) as input and predict a binary sentiment label (positive or negative). Here I describe the design and implementation details of each model:

Logistic Regression Model: Logistic regression is a simple yet effective linear classification model. It calculates a weighted sum of the input features and then applies a logistic sigmoid function to output a probability between 0 and 1. In our case, the model computes a score = w·x + b (where x is the TF-IDF feature vector of a sentence, w is a learned weight vector and b is a bias term). The sigmoid of this score is interpreted as the probability that the sentiment is positive. If the probability is above 0.5, the model predicts the label 1 (positive), otherwise 0 (negative). Logistic regression is well-suited for text classification tasks and is often used as a baseline because it can handle high-dimensional sparse data (like thousands of word features) efficiently. We implemented logistic regression using scikit-learn’s LogisticRegression class. I kept the model parameters at default settings (which include an L2 regularization to prevent overfitting) and trained it on our TF-IDF features. Training logistic regression involves finding the weight vector w and bias b that maximize the classification accuracy (or equivalently, minimize the logistic loss) on the training data. In practice, this optimization is done with algorithms like gradient descent. Logistic regression training is usually very fast for our dataset size, finishing in seconds.

Feedforward Neural Network (FNN): A feedforward neural network is a more flexible model that can capture non-linear relationships between features. I constructed a simple fully connected neural network with one hidden layer for this task. The architecture of our FNN is as follows: an input layer that takes in the TF-IDF features (one input node for each feature dimension), one hidden layer with a certain number of neurons, and an output layer with a single neuron to produce the final prediction. Each neuron in the hidden layer computes a weighted sum of all inputs, applies a non-linear activation function (we used the popular ReLU – Rectified Linear Unit – activation), and passes the result to the next layer. The single output neuron uses a sigmoid activation to output a probability of the sentence being positive, like logistic regression’s output. In our implementation, I decided to use 16 neurons in the hidden layer as a starting point, which gives the model enough capacity to learn interactions between words, but not so many that it would easily overfit the small dataset. I implemented the neural network using scikit-learn’s MLPClassifier (Multi-Layer Perceptron classifier), configuring it to have one hidden layer of size 16 and using the Adam optimizer for training. During training, the neural network adjusts its weight through backpropagation: the difference between the predicted sentiment and true sentiment (the error) is propagated backwards through the network to update the weights in each layer. I trained the network for enough iterations (epochs) until the training loss stabilized. Training the FNN in 2,400 sentences (our training set size) is still quite fast.

It is worth noting that logistic regression can be seen as a single-layer neural network (with no hidden layer, just direct input to output). By adding one hidden layer, FNN can, in theory, learn more complex patterns than logistic regression. However, it also has many more parameters to learn. Below is the code I used to instantiate and train both models after preprocessing the data:

文本

AI 生成的内容可能不正确。

After training the models, I evaluated their performance on the held-out test set (the 20% of data not seen during training). The primary evaluation metric we considered was accuracy, which is the proportion of sentences for which the model correctly predicted the positive/negative label. Both models achieved good accuracy on this binary sentiment task, indicating that features (TF-IDF bag-of-words) carry sufficient information to discern sentiment in these short texts.

Logistic Regression Accuracy: The logistic regression model achieved an accuracy of about 85% on the test set. In other words, roughly 85 out of every 100 review sentences were correctly classified as positive or negative by the logistic classifier. This high accuracy suggests that even a simple linear model can be very effective for sentiment analysis when using the bag-of-words features. Because logistic regression assigns a weight to each word, it essentially learned that certain words (like *“great”, “excellent”* or *“love”*) contribute to a positive sentiment, while others (like *“bad”, “terrible”, “not”*) indicate negative sentiment, and it made its predictions accordingly. The model’s errors tended to occur on sentences that were more ambiguous or had mixed sentiment. For example, a sentence like “The movie had great visuals but a dull story” contains both positive and negative aspects, which can confuse a simple classifier. Overall, however, the logistic model’s performance was strong, confirming that it’s a reliable baseline.

Neural Network Accuracy: The feedforward neural network (with one hidden layer of 16 neurons) obtained an accuracy of around 83% on the test set, which is in a similar range to logistic regression. The neural network was also able to correctly classify most reviews. Its performance being slightly (though not dramatically) lower than logistic regression could be due to a few reasons. First, the dataset is relatively small (only a couple of thousand training examples), so a simple model may generalize better with limited data. The logistic regression, being simpler (effectively a single layer), might have been less prone to overfitting on this small dataset. The neural network, with more parameters, has the theoretical ability to learn complex patterns (for instance, interactions between words), but with limited data those extra parameters don’t necessarily translate into better performance. That said, our neural network still performed well, and with more data or further tuning it might close the gap or surpass logistic regression. In some cases, I observed that the neural network correctly predicted the sentiment of certain tricky sentences that logistic regression got wrong, possibly by picking up on subtle combinations of words. But in other cases, it also made errors, sometimes on sentences that logistic regression got right. Overall, the difference in accuracy between the two models was small.

In this project, I successfully built and evaluated a binary sentiment analysis system on a real-world dataset of short English texts. I began by introducing the problem of sentiment analysis and selecting a suitable dataset of labeled sentences from movies, products, and restaurant reviews. I then cleaned the text data and used a bag-of-words representation (with TF-IDF weighing) to transform each sentence into a numeric feature vector. Using this representation, I trained a logistic regression classifier and a feedforward neural network to predict sentiment. My experiments showed that both models can achieve high accuracy (around 80–85%) on the task of distinguishing positive vs. negative sentiments in one-sentence reviews.

Reference:

(1): <https://www.ibm.com/think/topics/sentiment>

(2) & (3): <https://archive.ics.uci.edu/dataset/331/sentiment+labelled+sentences>