Machine Learning in Document Classification

STOR 767 Final Project

Cunyuan Huang

PID: 730041361

Supervisor: Prof. Andrew Nobel

November 2016

Contents

1 Abstraction

In this project, we will delve into a subfield of natural language processing (NLP) called sentimental analysis to classify movies by the reviews they received. We will first introduce the dataset, and the language model used for data preprocessing, where we will transform documents into vector representations. Then we will choose 3 machine learning algorithms to train the model, and analyze each of them. In the experiment, the core python code as well as the the main results will be presented. Lastly, some comparisons are made and further works are pointed.

2 Introduction

2.1 About the Dataset

In this project, we will be working with a large dataset of movie reviews from the Internet Movie Database (IMDb). It is collected by Maas et al[1]. The data is divided into a training set and a test set both containing 25,000 reviews, positive and negative reviews are labeled 1 and 0 respectively. Since some movies receive substantially more reviews than others, it is restricted that at most 30 reviews are included in any movie in the collection.

2.2 Word to Vector

In our predictive modeling, the response variable (positive 1 / negative 0) is straightforward, but the explanatory variable is reviews consist of words, we have to convert them into a numerical form before we can pass it on to a machine learning algorithm. Here we introduce the bag-of-words model that allow us to represent text as numeric feature vectors based on the word counts.

The basic idea behind the bag-of-words model is simple:

1. Create a vocabulary of unique tokens-for instance, words-from the entire collection of documents.

2. Construct a feature vector from each document that contains the counts of how often each word occurs in the particular document.

In addition, a technique called term frequency-inverse document frequency (*tf-idf* [2]) is used to downweight the frequently occurring words in the feature vectors. It is represented as follows:

$$\text{tf-idf}=tf(t,d)\times idf(t,d),$$
$$\text{Where: idf(t,d)}=\log\frac{n_d}{1+df(d,t)}$$

Here the tf(t,d) is the term frequency which counts the number of occurrence for a word in a certain document(i.e. a certain review), n_d is the total number of documents, and df(d,t) is the number of documents d that contain the term t. The log is used to ensure that low document frequencies are not given too much weight.

2.3 Data Cleaning

First step of data cleaning is to get rid of all unwanted characters like punctuations. Then we will use a technique called word stemming which transforms a word into its root form, it will allow us to map related words to the same stem. The stemming algorithm used in this project is WordNet, which can be accessed in the NLTK [3] module of python. In the last step, the common 'stopwords' in English are removed.

3 Our Training Model

Before delving into details of the training process, let us take a look back at the input and output of our model:

Input

- a document d. In the introduction part, we have transform it into a feature vector $\vec{v}$.
- a fixed set of classes $C = \{C_1, C_2, ..., C_k\}$. In our project $C = \{0, 1\}$.

Output

- a predicted class c$\in C = \{0, 1\}$.

3.1 Naive Bayes

As its name implies, naive Bayes is heavily based on Bayes' Theorem, it is a standard approach to use probability-based algorithm to machine learning. For a document d and a class c, $P(C|d) = \frac{P(d|C)\cdot P(C)}{P(d)}$, to have the model return the target level that has the highest posterior probability given the state of the descriptive features in the query, we use the **maximum a posterior(MAP)** formula:

$$C_{MAP} = \arg\max_{c\in C} \frac{P(d|c)\cdot P(c)}{P(d)} = \arg\max_{c\in C} P(d|c)\cdot P(c)$$

As we represent the document d as feature vector $\vec{w} = \{w_1, w_2, ..., w_n\}$, then C_{MAP} can be written as:

$$C_{MAP} = \arg\max_{c\in C} P(w_1, w_2, ..., w_n|c)\cdot P(c)$$

For simplicity we are assuming a **unigram language model**[4], thus it can be further decomposed as:

$$C_{MAP} = \arg\max_{c\in C} \prod_{w_i\in\vec{w}} P(w_i|c)\cdot P(c)$$

But how do we know the value of $P(w_i|c)$? Naturally, we think of estimating it using maximum likelihood

$$\hat{P}(w_i|c) = \frac{count(w_i,c)}{\sum_{w\in V} count(w,c)}$$

$\hat{P}(w_i|c)$ is the fraction of times word w_i appear among all words in documents of class c(i.e. the vocabulary V of class c). There is a problem with simple maximum likelihood estimation, because 0 probability can be returned if a term does not appear in a certain collection, but our model cannot deal with this condition, thus we need to do smoothing [5] to our original MLE formula:

$$\hat{P}(w_i|c) = \frac{count(w_i,c)+\alpha}{\sum_{w\in V} count(w,c)+\alpha|V|}$$

When $\alpha = 1$, it is called **Laplace smoothing**, which will be used in our project.

Now we have build up a multinomial naive Bayes model, and remember our feature vector is based on the bag-of-words model and calculated through tf-idf transformation, which is known to work well in practice. By using the python **scikit-learn**[6] module, we can adjust the smoothing priors α when making the maximum likelihood estimation.

3.2 Logistic Regression

We now turn to the second algorithm for classification, logistic regression. Logistic regression is a generalized linear model, which means it combines our feature vectors linearly and then applies a function to this combination. It is also a discriminative model that computes $P(C|d)$ directly:

$$P(C|d) = \frac{\exp(\sum \beta_i w_i)}{1+\exp(\sum \beta_i w_i)}$$

Here w_i is our feature vector and β_i is the corresponding weight.

Next we explain how the parameters of the model, the weights β are learned. The logistic regression is trained with **conditional maximum likelihood estimation**[7], which means we choose the parameters β that maximize the log probability of class labels in the training data given a document d.

$$\hat{\beta} = argmax_{\beta} \log P(c^{(j)}|d^{(j)})$$

For the entire dataset, the objective function L that we are maximizing is thus

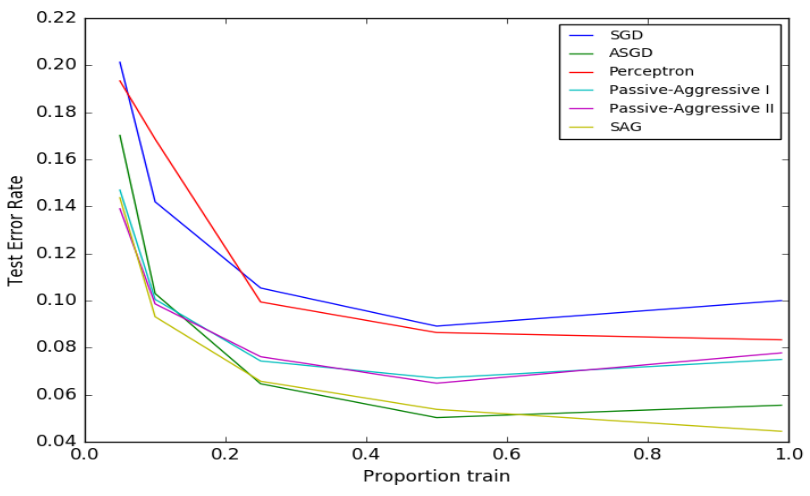
$$L(\beta) = \sum_j logP(c^{(j)}|d^{(j)})$$

To avoid overfitting, a regularization term is added to the objective function as follows:

$$L(\beta) = \sum_j logP(c^{(j)}|d^{(j)}) - \alpha R(\beta)$$

There are two common regularization terms called L2 regularization $R(\beta) = ||\beta||_2^2$, and L1 regularization $R(\beta) = ||\beta||_1$. In our experiment, we are using the L2 regularization, it is also known as **ridge logistic regression**[8].

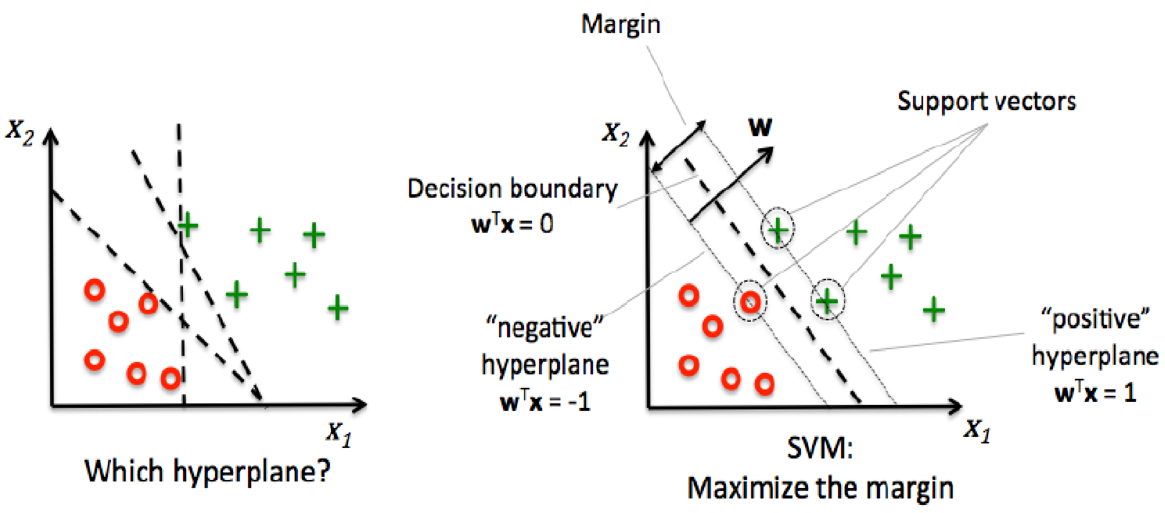
Finding the weights that maximize the objective turns out to be a convex optimization problem, so we use hill-climbing methods like stochastic gradient ascent, L-BFGS, or conjugate gradient[9]. Performance of some methods in python is compared bellow.



In our project we choose the **stochastic average gradient (SAG)**[10] method.

3.3 SVM

A support vector machine is a vector-space-based machine learning method. As a kind of large-margin classifier, its optimization objective is to maximize the margin, which is defined as the distance between the separating hyperplane (decision boundary). Support vectors refer to those training samples that are closest to this hyperplane.



This can be generalized as a quadratic optimizing problem–Find $\vec{\beta}$ and b such that:

- $\frac{1}{2}\vec{\beta}^T\vec{\beta}$ is minimized.
- for all $\{(\vec{w}_i, c_i)\}$, $c_i(\vec{\beta}^T\vec{w}_i + b) \geq 1$

To solve a nonlinear problem using an SVM, we transform the training data onto a higher dimensional feature space via a mapping function $\phi(\cdot)$ and train a linear SVM model to classify the data in this new feature space. In order to save the expensive step of calculating this dot product between two points explicitly, we define a so-called kernel function: $k(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^T\phi(x^{(j)})$

We will use the **Radial Basis Function** kernel (RBF kernel) in our project, it is also one of the most widely used kernels.

$$k(x^{(i)}, x^{(j)}) = \exp(-\frac{||x^{(i)} - x^{(j)}||^2}{2\sigma^2})$$

The term kernel can be interpreted as a similarity measurement between two samples. The minus sign inverts the distance measure into a similarity score and, due to the exponential term, the resulting similarity score will fall into a range between 1 (for exactly similar samples) and 0 (for very dissimilar samples).

In our experiment, except for the choice of kernel function, there are another two important parameters we have to adjust:

- gamma: Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.
- C : Penalty parameter C of the error term. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly.

We will see their usage later.

4 Experiment

4.1 Data Preprocessing

As stated in the experiment part, we use the NLTK module of python to get English stopwords and WordNet lemmatizer to clean the data in the first step.

```
In [ ]: from nltk.corpus import stopwords
        from nltk.stem.wordnet import WordNetLemmatizer

        def tokenizer_stem_wordnet(text):
            lmtzr=WordNetLemmatizer()
            stop=stopwords.words('english')
            textlst=[lmtzr.lemmatize(word) for word in text.split()
                     if word not in stop]
            return ' '.join(textlst)
```

Our input data here will be a csv file containing two columns– 'Review' which is the raw text data, and 'Sentiment' which indicates the sentiment value , respectively. To transform words into feature vectors(using the bag-of-words model and after the tf-idf transformation), we can apply the TfidfVectorizer in the sklearn module directly.

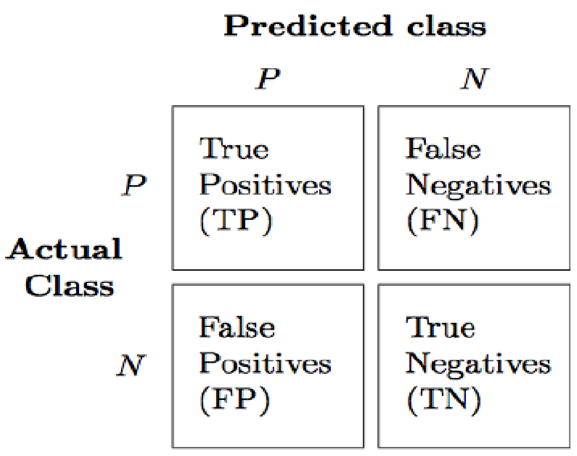
```
In [ ]: from sklearn.feature_extraction.text import TfidfVectorizer

        def wordtovec(Input_data):
            doc_lst=Input_data.Review.apply(tokenizer_stem_wordnet)
            X_data=pd.DataFrame(columns=['Review'])
            X_data['Review']=doc_lst
            Y_data=Input_data.Sentiment
            tfidf = TfidfVectorizer(min_df=1)
            X_data=tfidf.fit_transform(X_data.Review.tolist())
            return X_data,Y_data
```

The returned value will be two dataframes, one is the vectorized text data being the explanatory variables, and the other is the class label being the response variable. Now we have completed the data cleaning and transforming process, let us move up to the training and testing stage.

4.2 Model Training & Test Results

As we will see later in the results, a **confusion matrix** [11] for binary classification will be read in this way:



4.2.1 Naive Bayes

The multinomial naive Bayes model is relatively easy, we choose the Laplace smoothing ($\alpha = 1$) here.

```
In [ ]: from sklearn.naive_bayes import MultinomialNB

        def train_mnb(X,y):
            mnb = MultinomialNB(alpha=1.0, fit_prior=True)
            mnb.fit(X,y)
            return mnb
```

```
In [ ]: def main_test():
            X_train, Y_train,X_test, Y_test=read_data()
            model=train_mbn(X_train, Y_train)
            pred = model.predict(X_test)
            # Output the hit-rate and the confusion matrix for each model
            print(model.score(X_test, Y_test))
            print(confusion_matrix(pred, Y_test))
```

As we fit the training model and predict for the test set, we get the following cofusion matrix as well as the accuracy:

```
In []: #multi NB
        main_test()
```

```
0.831
[[10988  2713]
 [ 1512  9787]]
```

I.e. number of true negative(TN) is 10988, true positive(TP) is 9787, false negative(FN) is 1512, false positive(FP) is 2713, overall $accuracy = \frac{TN+TP}{TN+TP+FN+FP} = 0.831$.

4.2.2 Logistic Regression

```
In [ ]: from sklearn.linear_model import LogisticRegression
```

```
def train_logistic(X, y):
    """
    Create and train the Logistic regression model.
    """
    logis = LogisticRegression(C=10.0, verbose=10,
                                  n_jobs=-1, solver='sag')
    logis.fit(X, y)
    return logis
```

Here C is the inverse of regularization strength. Use the similar test function, we get the following results:

```
In [18]: #logistic
         main_test()
```

```
convergence after 33 epochs took 1 seconds
0.87248
[[11065  1753]
 [ 1435 10747]]
```

Here TN=11065, TP=10747, FP=1753, FN=1435, overall $accuracy = 0.87248$

4.2.3 SVM

The hyperparameters of SVM are a bit more complex, thus we first use a **GridSearchCV**[12] technique to find the optimal set of parameters for our model using 5-fold stratified cross-validation.

```
def grid_svm(X, y):
    clf = svm.SVC()
    tuned_parameters = [{'kernel': ['rbf','poly'], 'gamma': [0.0001],
                        'C': [1000,2000,500]}]
    search = GridSearchCV(clf, tuned_parameters, cv=5,scoring='accuracy',
                          verbose=10,n_jobs=-1)
    search.fit(X, y)
    print("best param:",search.best_params_)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed: 10.7min
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed: 18.8min
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed: 32.2min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 55.0min finished
best param: {'C': 1000, 'kernel': 'rbf', 'gamma': 0.0001}
```

After getting the optimized parameters, we can start training the model.

```
In [10]: def train_svm(X, y):
             """
             Create and train the Support Vector Machine.
```

```
          """
          svm = SVC(C=1000.0, gamma=0.0001, kernel='rbf')
          svm.fit(X, y)
          return svm
```

```
In [24]: #svm
         main_test()
```

```
0.87904
[[10895  1419]
 [ 1605 11081]]
```

Here TN=10895, TP=11081, FP=1419, FN=1605, overall $accuracy = 0.87904$

5 Discussion

5.1 Comparison

For the Naive Bayes model, it is very fast to train and has low storage requirements, although it is also robust for irrelevant features, the independence of feature vectors may not hold. In addition, this model performs well in domains with many equally important features. Thus it is an economical model but not giving good enough results.

The overly strong conditional independence assumptions of Naive Bayes sometimes overestimate the evidence, logistic regression is much more robust to correlated features, if there are many correlated features(which is common in practice), logistic regression will assign a more accurate probability than naive Bayes. In addition, algorithms like logistic regression and SVMs generally work better on large datasets. As we can see, logistic regression have a significantly better accuracy than Naive Bayes.

Support vector machines model suits the best for document classification task in our project, there is theoretical evidence behind this fact: Firstly, SVMs can handle high dimensional and sparse input data well, since SVMs use overfitting protection, which does not necessarily depend on the number of features. Secondly, Most text categorization problems are linearly separable[13]. The shortage of SVMs is that it is relatively time consuming.

In summary, the three machine learning models in this project have their pros and cons respectively. Due to the high dimensionality of our feature vector space, more complex models like SVMs tend to give better results, logistic regression is a good enough alternative considering the computational complexity.

5.2 Future Work

There's two main aspects that we can improve the prediction accuracy in the future. On one hand, in the data cleaning process, some emoticon characters such as ':)' may be useful for sentimental analysis, similarly, certain stopwords may also suggest sentimental information, we can discover those and keep them. On the other hand, unigram language model assumes independence between features, it is convenient to apply in practice, but this assumption may not hold, thus a multigram language model may perform better.

In addition, a more modern alternative to the bag-of-words model is **word2vec**[14], an algorithm that Google released in 2013, it is based on neural networks that attempts to automatically learn the relationship between words.

Working with text data can be computationally quite expensive, in many real-world applications it is common to work with datasets even larger than our computer's memory, thus some out-of-core [15] or incremental leaning is also a good technique for researchers.

References

[1] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts., *Learning Word Vectors for Sentiment Analysis*, Stanford University, CA, 2011

[2] Christopher D.Manning, Prabhakar Raghavan, Hinrich Schutze, *Introduction to Information Retrieval*, Cambridge University Press, 2008, pages 107-110.

[3] Retrieved from *http://www.nltk.org/*.

[4] Christopher D.Manning, Prabhakar Raghavan, Hinrich Schutze, *Introduction to Information Retrieval*, Cambridge University Press, 2008, pages 218-223.

[5] John D. Kelleher, Brian Mac Namee, Aoife D'Arcy, *Fundamentals of Machine Learning for Predictive Data Analytics*, The MIT Press, 2015, pages 272-276.

[6] Retrieved from *http://scikit-learn.org/stable/modules/naive_bayes.html*.

[7] Daniel Jurafsky and James H. Martin, *Speech and Language Processing: An introduction to natural language processing, computational linguistics, and speech recognition.*, draft of November, 2016, chapter 7.

[8] Scott A. Czepiel, *Maximum Likelihood Estimation of Logistic Regression Models: Theory and Implementation*, unpublished.

[9] Daniel Jurafsky and James H. Martin, *Speech and Language Processing: An introduction to natural language processing, computational linguistics, and speech recognition.*, draft of November, 2016, chapter 7.

[10] M. Schmidt, N. Le Roux, F. Bach. arXiv, *Minimizing Finite Sums with the Stochastic Average Gradient*, MAPR 2016.

[11] Sebastian Raschka, *Python Machine Learning*, Packt Publishing, 2015, page 190.

[12] Sebastian Raschka, *Python Machine Learning*, Packt Publishing, 2015, pages 244-245.

[13] Thorsten Joachims, *Text Categorization with Support Vector Machines: Learning with Many Relevant Features*, University of Dortmund.

[14] T. Mikolov, K.Chen, G.Corrado, and J. Dean, *Efficient Estimation of Word Representations in Vector Space*, arXiv preprint arXiv:1301.3781, 2013

[15] Sebastian Raschka, *Python Machine Learning*, Packt Publishing, 2015, pages 246-249.