PRML

Assignement - 3 Report

by

P Yashwanth Sai(cs22b002)

Dr Arun Rajkumar

Computer Science and Engineering

Indian Institute of Technology Madras

Jan - May 2024

Contents

1 Datasets

1.1 emails.csv

The following dataset is used from kaggle {emails.csv}. The emails were pre-processed by removing the numbers and punctuation marks. The final text in the email contains only words/alphabets in lowercase and whitespaces.

1.2 Preprocessing

The resulting data-frame was pre-processed using *sklearn.feature_extraction.text*

1.2.1 CountVectorizer(binary = True, stop_words = 'english')

- The dataset has been converted into a DataFrame where each column header represents a unique word/feature present in the dataset.
- Each row represents a data point or an email. The values are binary, either 0 or 1, indicating whether the corresponding word is present in the email (1) or not (0).
- stop_words are the common words in english that have no impact on our training methods an hence are removed to decrease the number of features/dimensions.
- This DataFrame will be represented as binary_data_array or something similar further in the report.

1.2.2 CountVectorizer(binary = False, stop_words = 'english')

- The dataset has been converted into a DataFrame where each column header represents a unique word/feature present in the dataset
- Each row represents a data point/email the values are the frequencies of how many times that word has occurred in the mail.
- This DataFrame will be represented as frequency_data_array or something similar further in the report.

1.2.3 TfidfVectorizer(stop_words = 'english')

- The dataset has been converted into a DataFrame where each column header represents a unique word/feature present in the dataset
- Each value in the row represents the value as calculated by the tfidf transformer over the dataset.
- This DataFrame will be represented as tfidf_data_array or something similar further in the report.

1.3 Notations

- X is the data matrix of dimensions $n \times d$. It can obtained from rawdata by using any of the pre-processing methods 1.2.1 , 1.2.2 , 1.2.3 whichever is applicable in the usage context.
- Y is the labels matrix of the data. It is of dimensions $n \times 1$

2 Naive Bayes Classifier

2.1 Training

- The maximum likelihood estimators for the parameters of naive bayes algo yields the following equations
- **binary_data_array** is used for Naive Bayes, Bayesian distribution is used.

$$\hat{p} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

$$\hat{p}_j^y = \frac{\sum_{i=1}^{n} \mathbb{1}(f_j^i = 1, y_i = y)}{\sum_{i=1}^{n}(y_i = y)}$$

The first equation is the fraction of spam emails in the data.
The second equation is the fraction of y-labelled emails that contain the j^{th} word. This is a $2D$ matrix of dimensions $2 \times d$.

2.2 Prediction

The label of a test data point is obtained from the application of Baye's rule

$$P(y^{test} = 1|x^{test}) \propto \left(\prod_{k=1}^{d} (\hat{p}_k^1)^{f_k}(1-\hat{p}_k^1)^{1-f_k}\right) \cdot \hat{p}$$

$$P(y^{test} = 0|x^{test}) \propto \left(\prod_{k=1}^{d} (\hat{p}_k^0)^{f_k}(1-\hat{p}_k^0)^{1-f_k}\right) \cdot (1-\hat{p})$$

if $P(y^{test} = 1|x^{test}) > P(y^{test} = 0|x^{test})$ predict $y^{test} = 1$ else $y_{test} = 0$

As the values of probabilities are very low and the number of features are $\approx$ 30000 the product will be too small and might get ignored even in *float 64* data types. Hence logarithm is applied to the above inequality and on simplification the following equation is obtained.

$$\sum_{i=1}^{d} f_i \log\left(\frac{\hat{p}_i^1(1-\hat{p}_i^0)}{\hat{p}_i^0(1-\hat{p}_i^1)}\right) + \left(\sum_{i=1}^{d} \log\left(\frac{1-\hat{p}_i^1}{\hat{p}_i^0}\right)\right) + \log\left(\frac{\hat{p}}{1-\hat{p}}\right) \geq 0$$

- The 2^{nd} and 3^{rd} terms are constant for all points so let that be denoted by b.
- In the first term, put the log terms in the summation into a $1 \times d$ matrix. Let that matrix be W.
- Also $X_{test} = [f_1, f_2 \dots f_d]$. Hence we can write $y_{pred} = sign(X_{test}W^T + b)$
- If X_{test} has n data points even then the same method is applied and we'll get an $n \times 1$ prediction matrix.

2.3 Evaluation

- Rawdata is processed and split into train_data and test_data with a ratio of 4 : 1.
- The model is trained on the train split and W, b are calculated.
- On the train split, 98.6% and on the test split 96.6% accuracy was obtained

3 Logistic Regression Classifier

3.1 Training

- **tfidf_data_array** is used for this classifier. The reasons will be discussed after tuning the hyper-parameters and cross validation which will be discussed in 3.3
- The likelihood function is given by the following equation. Here g is the sigmoid function.

$$\mathcal{L}(w; Data) = \prod_{i=1}^{n} (g(w^T x_i))^{y_i} (1 - g(w^T x_i))^{1-y_i}$$

- Apply log on both sides and further simplification, we obtain the log-likelihood function as follows

$$\log \mathcal{L}(w; Data) = \sum_{i=1}^{n} \big((1 - y_i)(-w^T x_i) - \log\big(1 + \exp\big(-w^T x_i\big)\big)\big)$$

- We cant obtain a closed form solution for w from the above equation hence we need to apply gradient ascent to maximize the log-likelihood function.
- On calculation, we get the graident as

$$\nabla \log \mathcal{L}(w) = \sum_{i=1}^{n} x_i \left(y_i - \frac{1}{1 + \exp(-w^T x_i)} \right)$$

- Gradient Ascent step:

$$w_{t+1} = w_t + \eta \cdot \nabla \log \mathcal{L}(w_t)$$

$$w_{t+1} = w_t + \eta \cdot \sum_{i=1}^{n} x_i \left(y_i - \frac{1}{1 + \exp(-w^T x_i)} \right)$$

- The above equation can be computationally expensive due to the calculation of gradient at each step which is loop of length n
- We can use the some properties of the np arrays in python and do the following. Let σ denote the sigmoid function.
- $\sigma(w^T \cdot X)$ yields an matrix of size $n \times 1$, where each element represents the corresponding sigmoid function value.
- The Gradient Ascent step can be modified into:

$$w_{t+1} = w_t + \eta(X^T \cdot (Y - \sigma(w^T \cdot X)))$$

3.2 Prediction

- The predication for a test data point is given by the sign of $w^T \cdot x_i$

$$y_{pred} = \begin{cases} 1 & \text{if } w^T \cdot x_{test} \geq 0 \\ 0 & \text{if } w^T \cdot x_{test} < 0 \end{cases}$$

3.3 Evaluation and Inferences

- The following data was obtained for 100 iterations of gradient ascent.
- '−' Indicates that an overflow has occured.

Data	η (step size)	Time	Accuracy	
			Train split	Test split
tfidf_data	1	10s	100%	98.8%
	10^{-1}	10s	100%	99.4%
	10^{-2}	10s	99.8%	99.1%
	10^{-3}	10s	94.9%	91.2%
frequency_data	10^{-3}	–	–	–
	10^{-4}	90s	99.54%	99.56%
	10^{-5}	90s	98.4%	97.6%
	10^{-6}	90s	89.6%	86.1%

- From the above table, we can see that Logistic regression runs **faster** with tfidf_data than frequency_data. It is also more accurate.
- Hence the features are extracted from the text using *tfidf_vectorizer* (1.2.3).
- $\boldsymbol{\eta = 10^{-1}}$ produces a better result over both test and train splits.
- Train accuracy decreases on decreasing η and test accuracy increases and then decreases, the maximum occurs at $\eta = 10^{-1}$

4 SVM Classifier

4.1 Training

- SVM is implemented using the sklearn module in python.

```
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
svm = LinearSVC(dual=True, max_iter=5000, C = 1)
svm.fit(tfidf_data_array, data['label'])
```

- The above code initializes an SVM object with a linear kernel and trains the model on the input feature matrix and corresponding labels.
- tfidf_data(1.2.3) and $C = 1$ is chosen which is explained in 4.2

4.2 Evaluation and Inferences

- Labels are predicted from the test data and the accuracy is calculated as follows.

```
train_predictions_svm = svm.predict(tfidf_train_array)
train_accuracy_svm = accuracy_score(train_data['label'], train_predictions_svm)

test_predictions_svm = svm.predict(tfidf_test_array)
test_accuracy_svm = accuracy_score(test_data['label'], test_predictions_svm)
```

Data	C	Time	Accuracy	
			Train split	Test split
tfidf_data	0.1	1.1s	99.7%	98%
	1	1.1s	100%	99.1%
	5	1.1s	100%	99.1%
	10	1.1s	100%	99.1%
	100	1.1s	100%	99.2%
frequency_data	0.1	2s	100%	98.7%
	1	2s	100%	98.4%
	5	2s	100%	98.3%
	10	2s	100%	98.3%
	100	2s	100%	98.3%

- The input features are processed using tfidf_data (1.2.3) rather than frequency_data (1.2.2) because the former runs relatively faster than latter as evident from the above table. Also test accuracies are better with tfidf_data.
- Hence **tfidf_data** (1.2.3) is used.
- We can see that there isn't much difference with the change of the hyper parameter C hence the default value of $\boldsymbol{C = 1.0}$ is chosen. A high value gave a slight increase in test accuracy but that may lead to hard margin classifier. Hence they arent chosen.

5 Final Predictor

- Entire data, 1.1 is used for training all the models i.e, Naive Bayes, Logistic Regression and SVM. No train and test splits are created.
- The models are initialized with the hyper-parameters and data sets as discussed in the previous sections.
- **For an email{n}.txt which is present in the test folder under the same directory, the output is predicted by the mode of predictions of Naive Bayes, Logistic Regression and SVM.**

- That is we give the most occurring prediction in Naive Bayes, Logistic Regression and SVM as the final output.
- **The outputs are written out through the print command into the standard output stream with each prediction (1 or 0) in a seperate line. Also the individual predictions of each algorithm is written into a *'predictions.txt'* file**

```
for i in range(len(final_predictions)):
    print(final_predictions[i])

with open('predictions.txt', 'w') as output_file:
    for i, prediction in enumerate(final_predictions):
        output_file.write(f'Email {i + 1} : NB: {bnb_predictions[i]},
        SVM : {svm_predictions[i]}, LR: {lr_predictions[i]},
        Final prediction: {prediction}\n')

```