PRML Assignment 3: Spam or Ham

Rudra Panch
ME21B165

May 4, 2024

1 Binary Classification

Binary classification is the process of using an algorithm to split a dataset which is labelled to be part of 1 out of 2 classes, in such a way that we ge tall data points of similar class onto one side and the other class on another side. There are many ways to do this. Some popular examples include K-Nearest Neighbours, Decision Trees, Support Vector Machines, Naive-Bayes Algorithm, Naive-Bayes-Gaussian Algorithm.
Some well-known examples of binary classification examples are the cat-dog categorization, credit fraud classification, spam classifiers and many more. In this report, we shall be looking at building spam classifiers from scratch using multiple algorithms available.

2 Dataset

We are working with training a model to classify emails into spam and ham based on the content/words used in the email. This can be done by looking at a sample dataset of emails of which we have knowledge to be either spam or ham and make the model learn the features of a spam email and predict it on a new data point. Here, I have used the dataset provided by the SpamAssassin website which has thousands of emails already analyzed and marked as spam or ham available to be used for training.

2.1 Features

To classify an email as spam, we need to know the salient features that a spam email usually contains. Here, we analyse this using the Bag-Of-Words technique. We look at all words in the dictionary of these emails and check the frequency of appearance of these words in a spam and a ham email. This gives us a fair idea about the dataset and a starting point to classify.

```
from collections import Counter

def build_dictionary(emails):
    word_counter = Counter()
    for email in emails:
        email = remove_html_tags(email)
        words = email.split()
        word_counter.update(words)
    return word_counter

def sort_dictionary(emails):
    word_counter = build_dictionary(emails)
    sorted_dict = dict(sorted(word_counter.items()))
    sorted_dict = dict(list(sorted_dict.items())[0:N])
    return sorted_dict
```

With build_dictionary(emails), we put together all the words used in the emails into a dict structure along with their frequencies. The frequency is important to understand which words are the

important features. Also, we do not need some unnecessary features like html tags and the remove_html_tags(email) takes care of this. The function itself is taken from the 're' library(Stackoverflow answer). When we print the words and their frequencies, we can see that there are a lot of rarely used words which show up only once or twice, the probability of these appearing in the test emails are also very low and can be ignored. We take the top 10000 words for easier calculations while not compromising on the efficiency too much

2.2 Data Points

The 10000 words that we have serve as the features now. Hence, we need to represent all of our emails in terms of these 10000 words. We make a 10000 sized vector for each email. The ith value in this array is set to 1 if the ith word appears in this email and set to 0 else. In this way, we can make an array of binary values for each email. These binary arrays will now be our data points

```
def make_data(email, sorted_dict):
    words = email.split()
    X = np.zeros(10000)
    i = 0
    for key in sorted_dict:
        if key in words:
            X[i] = 1
        i = i+1
    return X
```

3 Hard Margin Support Vector Machine

Support Vector Machines are a popular method of modelling binary classifiers. The biggest advantage is the compression of data achieved using this. This algorithm decreases the computational cost significantly from other algorithms. In the end, we shall looking at the loss function, which gives us more insight into why the SVM is one the best methods for binary classification.

3.1 The Problem

We have a dataset of points in the real space, all labelled as either -1 or +1(sometimes 0 and +1). We need to find an optimal vector which categorizes all the points into 2 different classes correctly. As explained above, the data points are all arrays of binary valued features which determine the existence of a word in that particular email. We also need to validate the performance of our model and as we do not have access to the test data, we split the dataset into a training and a validation set. We train the model on only 80% of the data and then find the error on the validation set to understand the accuracy of the model.

3.2 Assumptions and Objective

We assume that the data is linearly separable with a γ-margin. This means that there exists an optimal vector $\vec{w}$ which separates the data perfectly and there is a margin of width 2γ(s.t $\gamma > 0$ between 2 lines parallel to $\vec{w}$ within which no data point can be found. This assumption is important to solving the NP-Hard binary loss minimization problem. These are formally stated as

$$w^T x_i y_i \geq \gamma \qquad s.t. \gamma > 0$$

$$w = \arg\min_w \sum_{i=1}^{n} 1(sign(w^T x_i) \neq y_i)$$

3.3 Primal and Dual Problem

In the SVM, we assume the γ-margin to be 1 and minimize the length of the w vector while having the linear separability assumption in place. We state the objective function formally as

$$\min_w \frac{1}{2}\|w\|^2 \qquad s.t. \quad w^T x_i y_i \geq 1 \quad \forall i$$

We can here, invoke the concept of Lagrangian and include the condition function also into the minimization equation. We take $f(w) = \frac{1}{2}\|w\|^2$ and $g(w) = 1 - w^T x_i y_i$. With this, the dual problem can be stated as

$$\min_w \frac{1}{2}\|w\|^2 = \min_w \left[\max_{\alpha \geq 0} \left[\frac{1}{2}\|w\|^2 + \sum_{i=1}^{n} \alpha_i (1 - w^T x_i y_i)\right]\right]$$

The min and max can be swapped as well, as f(w) and g(w) both are convex functions. After swapping , we find the optimal w for the inside function by differentiating and equating to zero. This lead us to

$$w = \sum_i^n \alpha_i x_i y_i \quad \text{as a summation and}$$

$$w = XY\alpha \quad \text{in matrix notation}$$

When we substitute this back into the dual equation, we get the loss function in terms of α and we find the α values by applying gradient descent on the objective

$$\max_{\alpha \geq 0} \left[\alpha^T 1 - \frac{1}{2}(XY\alpha)^T (XY\alpha)\right]$$

3.4 Complementary Slackness

From the Lagrangian, we have a very important result in the complementary slackness which states $\alpha^* g(w^*) = 0$ which implies that at least 1 out of the 2 terms in the equation have to zero for all data points

$$\alpha_i (1 - w^T x_i y_i) = 0$$

This leads us to 2 cases namely $\alpha = 0$ and $\alpha > 0$. The first case leads to $1 - w^T x_i y_i) \geq 0$ and second case leads to $1 - w^T x_i y_i) = 0$. Therefore, if $\alpha > 0$, these points lie on the hyperplane which indicates the end of the margin being considered. In the calculation of w, α can be assumed to be the weights given to each point and a zero weight implies that the data point has no contribution to the value of w. Formally, α is known as the importance of the point. This leads to the fact that only the points on the hyperplane contribute to the value of w in any way.

This is one of foremost advantages of SVM as the optimal solution can be expressed in terms of merely a few points as compared to the size of the whole dataset. The biggest drawback, on the other is the linear separability assumption which is very rarely ever followed in real life problems

4 Soft-Margin Support Vector Machine

The algorithm discussed uptil now is known as the Hard-Margin SVM as it has a strong condition on the γ-margin. The algorithm going to be discussed now, Soft-Margin SVM, does not require the linear separability assumption and is therefore, not imposing a strong margin condition.

4.1 Bribes

To deal with the problem of outliers, we introduce a concept called bribes. We can fix any w as the solution in this algorithm, there is no necessity for an optimal w to exist. This w, to classify the misclassified points, pays a bribe to push it past the margin. Of course, we do not want solutions to be taking too many bribes and we need to impose a minimization condition on the bribes as well. So, we re-define the objective function

$$\min_{w,\epsilon} \frac{1}{2}\|w\|^2 + C\sum_{i}^{n} \epsilon_i \qquad s.t. \quad w^T x_i y_i \geq 1 \quad \text{and} \quad \epsilon_i > 0 \quad \forall i$$

C here is a hyper parameter that is too be found using cross-validation. Again, we use the concept of Lagrangian and state the dual problem as

$$\min_{w} \left[\max_{\alpha \geq 0} \left[\frac{1}{2}\|w\|^2 + \sum_{i=1}^{n} \alpha_i (1 - w^T x_i y_i - \epsilon_i) + \sum_{i=1}^{n} \beta_i (-\epsilon_i) \right] \right]$$

solving this by differentiating it w.r.t. w and ϵ, we obtain the same solution for $w = XY\alpha$ and we also get an important condition that states $C = \alpha_i + \beta_i$.

4.2 Complementary Slackness

This time, we have 2 equations for complementary slackness, one in terms of α and the other in β, These can be stated as

$$\alpha_i (1 - w^T x_i y_i - \epsilon_i) = 0$$
$$\beta_i \epsilon_i = 0$$

This leads us to 3 different cases. This is due to the constraint of $C = \alpha_i + \beta_i$. We take 3 cases, $\alpha = 0$, $0 < \alpha < C$ and $\alpha = C$.

- Case 1: $\alpha_i = 0$
  This leads to $\beta_i = C$ and $\epsilon_i = 0$ and $1 - w^T x_i y_i \geq 0$ which implies these points have already been correctly classified and do not require a bribe and they also do not contribute to calculation of w.

- Case 2: $0 < \alpha_i < C$
  β_i also lies in 0,C now and is non-zero. Hence, $\epsilon_i = 0$ and $1 - w^T x_i y_i = 0$. These points lie on the hyperplane, are classified correctly, do not require any bribes and contribute to the calculation of w.

- Case 3: $\alpha_i = C$
  $\beta_i = 0$ which implies $\epsilon_i \geq 0$. $1 - w^T x_i y_i - \epsilon_i = 0$. These points are misclassified and require bribes to be brought past the margin. These points also contribute to the calculation of w. This case is the special case which discriminates the Soft-Margin from the Hard-Margin

5 Implementation of SVM

For implementing the SVM model, I have used the SVC function which is part of the sklearn.svm library class and fit an SVM model on the training data. The error was validated on the validation set by comparing the model's predictions in these points with the actual labels.

```
from sklearn.svm import SVC

def svm(X_train, Y_train, X_test, Y_test, c):
    svm_classifier = SVC(kernel='linear', C=c)
    svm_classifier.fit(X_train, Y_train)
    Y_pred = svm_classifier.predict(X_test)
    error = 0
```

```
for i in range(Y_pred.shape[0]):
    if Y_test[i] != Y_pred[i] :
        error += 1
accuracy = 1 - error/Y_test.shape[0]
return Y_pred, accuracy
```

The model was trained and the prediction values turned to have an accuracy close to 99% which turns out to be a very efficient solution. We can give a value for c which can be cross-validated. If we need to run Hard-Margin SVM, we can set c to a very high number, like 10^9 which will make sure the bribes are close to zero and we are in fact, just running Hard-Margin. The loss function of the SVM is a special loss function called the hinge loss which mimics the 0-1 NP-Hard loss the best among all classifiers. Therefore, the SVM is usually the best performing of all classifiers on any dataset.

6 Naive Bayes Algorithm

There are 2 types of modelling the distribution of a binary classification problem, the generative model and the discriminate model. The discriminate model does not focus on how the data was generated and instead focuses on the probability of the label occurring given the dataset. The generative model, on the other hand, considers both and even models how the data could be generated.

6.1 Generative Story

Let us consider the problem at hand itself, the spam classifier and come up with a generative story for it. Let us have a probability p which denotes the chance of an email being spam. Now we generate an email based on this probability and label it spam/ham. As discussed above, the features of the spam classifier are the words in the email. Each of these words, has a probability of appearing in a spam mail and a different probability of appearing in a ham mail. For each word, we do a coin toss with a coin having the bias equal to this probability and include the word in the mail depending on the result. Like this, we toss a coin for all words in the dictionary and create an email.

6.2 Maximum Likelihood Estimator

Using MLE, we can find the values of these probabilities. p is the probability of the email being spam, this can be easily calculated by finding the number of spam mails in the whole dataset. The probability of occurrence of a particular word in a spam mail is the number of spam mails in which the word occurs to the total number of spam mails and the corresponding values for ham mails as well. Let y be the variable which describes if the mail is spam or ham i.e., y=1 denotes spam and y=0 ham. f_j^i is the jth feature of the ith data point. p_j^y refers to the probability of the jth feature being 1 in a data point labelled y.

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

$$p_j^y = \frac{\sum_{i=1}^{n} 1(f_j^i = 1)1(y_i = y)}{\sum_{i=1}^{n} 1(y_i = y)}$$

6.3 Probability and Prediction

The probability of an email being generated given the training dataset is given by the Likelihood function which takes into account the occurrence of each word given that now we know the probability of the word appearing

$$L(x|y) = \prod_{i=1}^{n} (p_j^y)^{f_i^j} (1 - p_j^y)^{1-f_i^j}$$

We know that the Bayes' rule is a powerful formula which relates the prior to the posterior with help of the Likelihood the data that we have. In this case, we have the value of p(x—y) but we need the value of p(y—x). Since, we have no method of directly calculating it, we make use of Bayes' rule

$$P(Y|X) = \frac{P(X|Y)}{P(X)} P(Y)$$

$$P(Y|X) = \frac{1}{P(X)} \left[\prod_{i=1}^{n} (p_j^y)^{f_i^j} (1 - p_j^y)^{1-f_i^j} \right] P(Y)$$

P(X) is just a normalizing factor which depends on the data and can be neglected for further calculations. P(Y) is p when y=1 and (1-p) if y=0. To make a prediction on a test data, we find the probability of that test data being a spam mail and of that being a ham mail. Whichever turns out to be higher, we predict the same. Essentially, a mail is predicted spam when

$$\left[\prod_{i=1}^{n} (p_j^1)^{f_i^j} (1 - p_j^1)^{1-f_i^j} \right] p > \left[\prod_{i=1}^{n} (p_j^0)^{f_i^j} (1 - p_j^0)^{1-f_i^j} \right] (1 - p)$$

6.4 Code

We first split our dataset into a training and test set in 80:20 split. We pass these to the function, which first calculates the probability p of an email being spam. Then we find the p_j^y values by traversing through the data for all features

```
def naive_bayes(X_train, Y_train, X_test, Y_test):
    n_spam = np.sum(Y_train)
    n_ham = Y_train.shape[0]-n_spam
    p = np.zeros((2,max_words))
    for j in range(max_words):
        for i in range(X_train.shape[0]):
            p[int(Y_train[i])][j] += X_train[i][j]
    p[0] = p[0]/n_ham
    p[1] = p[1]/n_spam
    for i in range(X_test.shape[0]):
        p0=n_ham/(n_ham+n_spam)
        p1=n_spam/(n_ham+n_spam)
        for j in range(max_words):
            p0 = p0*pow(p[0][j],X_test[i][j])*pow(1-p[0][j],1-X_test[i][j])
            p1 = p1*pow(p[1][j],X_test[i][j])*pow(1-p[1][j],1-X_test[i][j])
        if p1 > p0:
            Y_pred[i] = 1

def run_naive_bayes(X,labels):
    ind = np.random.permutation(X.shape[0])
    X_new = X[ind]
    Y_new = [labels[i] for i in ind]
    index = int (0.8*X.shape[0])
    X_train = X_new[:index, :]
    X_test = X_new[index:, :]
    Y_train = np.array(Y_new[:index])
    Y_test = np.array(Y_new[index:])
    Y_pred,accuracy=naive_bayes(X_train, Y_train, X_test, Y_test)
```

Now that we have all the parameters ready, we can predict on the test data by using the Bayes' rule. Prediction of the label is done based on whether the probability of it being spam is higher or ham. In the end, we calculate the error of prediction on the test data by comparing with labels available and find the accuracy of our model. Our code achieved an average accuracy of 90% on random train-test splits.

7 Logistic Regression

one more popular method of solving binary classification problems is using the Logistic Regressor. in this, we use probabilistic model where we want to assign probabilities dependent on the distance of the point from the optimal w separating plane. Points farther away from the plane should have a higher probability of being classified correctly whereas the ones near the line should have a half and half chance of being classified.

7.1 Sigmoid Function

To achieve this mapping of the whole dataset to a [0,1] range with points near the line having values closer to 0.5 and the one farther having values closer to either 1 or 0 depending on the side of the plane, we use a function called the sigmoid function which is given by

$$g(z) = \frac{1}{1+e^{-z}}$$

Here, z is the classifier being used. If linear, $z = w^T x$ and similar formulas for quadratic and so on. This function value is the probability assigned to a given point x.

7.2 Log Likelihood and Gradient Descent

We can calculate the Likelihood function once we know the individual probabilities and optimize it to get the optimal w value

$$\begin{aligned}
P(y=1|x) = g(z) &= \frac{1}{1+e^{-z}} \\
L(w|x) &= \prod_{i=1}^{n} g(w^T x_i)^{y_i}(1-g(w^T x_i))^(1-y_i) \\
logL(w|x) &= \sum_{i=1}^{n} \left[y_i \log(g(w^T x_i)) + (1-y_i)\log(1-g(w^T x_i))\right] \\
logL(w|x) &= \sum_{i=1}^{n} \left[(1-y_i)(-w^T x) - \log(1+e^{-w^T x})\right]
\end{aligned}$$

Since a closed form solution of this does not exist, we make use of gradient descent to solve this. The formula for the gradient and the update rule are

$$\nabla logL(w|x) = \sum_{i=1}^{n} \left[x_i \left(y_i - \frac{1}{1+e^{-w^T x}}\right)\right]$$

$$w_{t+1} = w_t + \eta_t \nabla logL(w|x)$$

7.3 Implementation

We set a threshold of convergence and run the gradient descent until the threshold is reached. The gradient is computed. The step size is taken to be a constant 10^{-6}. Once the gradient descent is complete, we predict the values on the test data and compare them with the given labels.The prediction is done by calculating the value of the sigmoid function and assigning it to spam if the value is greater than 0.5 and to ham if less than

```
while error>threshold:
    grad = 0
    for i in range(X_train.shape[0]):
        grad = grad + X_train[i]*(Y_train[i]-(1/(1+np.exp(-(w.T @ X_train[i])))))
    grad = grad*eta
```

```
        error = np.linalg.norm(grad)
        w = w + grad
    Y_pred = np.zeros(Y_test.shape[0])
    for i in range(X_test.shape[0]):
        temp = 1/(1+np.exp(-(w.T @ X_test[i])))
        if temp > 0.5:
            Y_pred[i] = 1
```

The model was trained and the accuracy was checked. The model had an average accuracy of 98% on various random train-test splits. This validates our logistic regression model.

8 Conclusions

- In this report, we have implemented 3 different types of binary classifiers, Support Vector Machine, Naive-Bayes Algorithm and the Logistic Regression. All the three models were trained and all of them show above 90% rate of accuracy.
- The SVM had the highest accuracy with an average of almost 99% and had a very fast time of execution. Of course, this depends on the library being used and it's degree of fine tuning. However, the SVM model which was expected to perform the best, did.
- The Naive-Bayes classifier did very well by performing over 90% at all times. The low value could be due to picking of only the top words and not using all of them. Even then, it's time of execution was pretty fast and it showed a good amount of accuracy.
- The logistic regression was on par with the SVM showing 98% accuracy on an average. This can be awarded to the probabilistic approach to the model which trains it better than the deterministic models.
- Overall we have built 3 spam classifiers and all of them are easily on par with the industry standards and can be tested on unkown data to produce good results.