ENGR 421

Homework 01: Naïve Bayes' Classifier

Deadline: October 14, 2019, 11:59 PM

Sitare Arslantürk, 57677

In this assignment, Naïve Bayes' Classifier is implemented via multivariate methods.
This means that we will be dealing with class conditional densities described as

$$p(x|y = c) \sim N(x; \mu, \Sigma)$$

where x is the D-dimensional column vector, μ is the mean column vector and Σ is the covariance matrix. Thus, the PDF or probability density function becomes

$$N(x; \mu, \Sigma) = \sqrt{\frac{1}{(2\pi)^D |\Sigma|}} * exp\left[-\frac{1}{2} * (x-\mu)^T * \Sigma^{-1} * (x-\mu)\right]$$

In this multivariate gaussian distribution x is vector input, μ is the vector mean and $|\Sigma|$ is the determinant of a matrix and Σ^{-1} is the inverse of covariance matrix. Scipy's library function can calculate this function for multiple input vectors.
Naïve Bayes assumes that none of the features are correlated, assuming all are independent.This property allows all the covariance terms that are off the diagonals to be zero, since covariance terms show us whether the variables are scattered together or not.

$$Cov(i,j) = E\left[(x_i - \mu_i)(x_j - \mu_j)\right] = 0, \qquad given\ x_i\ is\ independent\ of\ x_j$$
$$Cov(i,i) = \Sigma_{ii} = var(x_i) = \ \sigma_i^2$$

Thus the covariance matrix can be shown as

$$\Sigma = \begin{bmatrix} \hat{\sigma}_1^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \hat{\sigma}_D^2 \end{bmatrix}$$

Instead of a D*D covariance matrix, we only need D different single dimension variances which is a axis aligned elliptical covariance. Instead of inverting the covariance matrix, we only need to invert the diagonal matrix calculated by

$$(\Sigma^{-1})_{ii} = \ 1/\sigma_i^2$$

Scipy's library allows using a D*D covariance matrix or a D sized vector. This is still doing the same thing as

$$p(x|y=c) = N(x_1;\mu_1,\sigma_1^2) * N(x_2;\mu_2,\sigma_2^2) \dots N(x_D;\mu_D,\sigma_D^2)$$

Calculating the exponential function is quite time consuming. Instead of this, we will use logarithm probability which will bring the exponentials down and turn the multiplications into summation. Scipy has a built-in logpdf function to calculate the multivariate gaussian.

$$Prediction: p(y=c|x) = argmax_c\{\log[p(x|y=c)] + \log[p(c)]\}$$

where $p(y=c|x)$ is the posterior, $\log[p(x|y=c)]$ is the likelihood and $\log[p(c)]$ is the class prior.

I read the data using pandas from the two given cvs files and assigned the images to X and labels to Y. I assigned the model as the Naïve Bayes classifier I wrote and fit my training points. I defined two functions as fit and predict, which below shown are the pseudocodes. In the fit function, the mean and covariance for each class is calculated which will represent the gaussian distribution. The priors are also calculated in this function.

In predict function, I looped through every sample I wanted to predict on that loop through for each class in which I get the mean, variance and the prior for that specific class. The mean and variance are used to calculate the logpdf for the current sample point in the current class. Then this is added to the logarithm of the prior to get the posterior. If this new posterior is better than the maximum posterior, it is set as the best current prediction.

The results are shown via a confusion matrix with the help of sklearn.metrics. I calculate two confusion matrices each for the data points in the train and test set. The output matrices matched the ones given in the guidelines, as well as the outputs for the priors, means and variances were matching.

```
def fit(X, Y):
      dict_of_gaussian = {}
      dict_of_priors = {}
      for c in classes:
            x_c = X[curreny Y == c]
            mean, var = mean, diagonal of covariance of x_c
            dict_of_gaussian[c] = {'mean':mean, 'var':var}
            dict_of_priors[c] = x_c.len - x.len
```

```
def predict(X):
      predictions = [], max_post = -inf, best_class = None
      for x in X:
            for c in classes:
                  mean, var = dict_of_gaussian[c]
                  post = logpdf(x, mean, var) + log(dict_of_priors[c])
                  if(post>max_post):
                        max_post = post
                        best_class = c
            predictions.append(best_class)
      return predictions
```