Hot Topics in Machine Learning (HWS17) Assignment 2: Naive Bayes

Steffen Schmitz
University of Mannheim
stefschm@mail.uni-mannheim.de

1. TRAINING

Task. Provide a function that trains a Naive Bayes classifier for categorical data using a symmetric Dirichlet prior and MAP parameter estimates.

We want to use a Naive Bayes classifier to assign probabilities to each possible class-label given a new, previously unseen datapoint. Given Bayes' rule we say that

$$P(Y|X) = \frac{P(Y)P(X|Y)}{P(X)} \propto P(Y)P(X|Y) \qquad (1)$$

where $P(Y|X)$ is the probability or our posterior belief that $Y = y$ for a specific datapoint x.

Following Murphy [1, p.65] we can use the proportionality of Equation 1 to predict the class-label of a feature vector x. Therefor, we need to compute the class-conditional densities $P(X|Y)$ for each class y and the prior class probability $P(Y)$.

To compute the prior we can maximize the Multinomial distribution which would lead to a Maximum Likelihood Estimate (MLE) that is equal to the relative frequencies of each category in the training data. This can result in a zero-count problem, when a category does not appear in the test data. Using a symmetric Dirichlet prior we obtain the Maximum a Posteriori estimation (MAP) that is computed as

$$\hat{\theta}_{MAP} = \frac{n_1 + \alpha - 1 \; n_2 + \alpha - 1 \; ... \; n_K + \alpha - 1}{n + K \cdot \alpha - K} \qquad (2)$$

with $\alpha \in \mathbb{R}^+$ and $\alpha = 1$ being equal to the MLE.

The Python implementation is shown in Figure 1.

```
# Compute the priors P(Y)
num = np.bincount(y) + alpha - 1
denom = y.size + K * alpha - K
priors = np.divide(num, denom)
```

Figure 1: Prior class probability.

For the class-conditional densities we compute the number of occurrences of each feature for every class and apply the same formula as above.

We combine the implementations in Figure 1 and Figure 2 to define the nb_train method that returns both results in logarithmic form.

```
cls += (alpha - 1)
for i in range(C):
    indizes = np.where(y == i)
    for index in indizes[0]:
        for d in range(D):
            cls[i][d][X[index][d]] += 1

    for d in range(D):
        cls[i][d] = np.divide(
            cls[i][d],
            np.sum(cls[i][d])
        )
```

Figure 2: Class-conditional densities.

2. PREDITION

Task. Provide a function that takes your model and a set of examples, and outputs the most likely label for each example as well as the log probability (confidence) of that label.

In this exercise we will implement a function that takes in a model that we trained with the function from Section 1 and use it to compute our belief that a new feature vector x belongs to a specific class.

We know from Equation 1 that we need to compute the product of the prior class probability and the class-conditional densities to obtain the probability that a feature vector x belongs into class y. If we do this computation in logarithm-space we obtain the log-likelihood and can use sums instead of products. This leads to Equation 3 [1, p.83].

$$\ell(\mathbf{X}, \mathbf{y}|\theta) = \sum_{c=1}^{C} n_c \log \pi_c + \sum_{j=1}^{D} \sum_{c=1}^{C} \sum_{i:y_i=c} \log[\theta_{jc}] x_{ij} \qquad (3)$$

We compute the log-likelihood for each possible class for each feature vector x and use the maximum result as our prediction.

3. EXPERIMENTS ON MNIST DIGITS DATA

3.a Accuracy

Task. Train your model with $\alpha = 2$ on the MNIST training dataset, then predict the labels of the MNIST test data using your model. What is accuracy of your model?

If we create a model and make predictions as explained in Section 2 we obtain an accuracy of 83.6% on the MNIST-dataset[1]. This is equal to the accuracy that we get by applying the Multinomial Naive Bayes algorithm from the scikit-learn Python package[2]. The scikit-learn Bernoulli Naive Bayes program performs a little better and has an accuracy of 84.1% on the testset.

Given that the result is the same as the scikit-learn implementation we can conclude that our Naive Bayes implementation is reasonable.

Nevertheless it should be possible to get much better prediction results. The TensorFlow documentation states that a result of 92% accuracy is fairly bad for a simple neural network and that they can get results of around 99.2% with a Multilayer Convolutional Neural Network[3].

3.b Error discussion

Task. Plot some test digits for each predicted class label. Can you spot errors? Then plot some misclassified test digits for each predicted class label. Finally, compute the confusion matrix. Discuss the errors the model makes.

First, we look at the predicted classifications of a sample of images and later analyse the confusion matrix. Figure 3 shows some images from the testset grouped by the predicted class label. This Figure contains images with a wrong label and images with a correct label. Due to the accuracy of 83,6% we would expect 125 of the 150 images to have the correct label (Actual: 24 misclassifications).

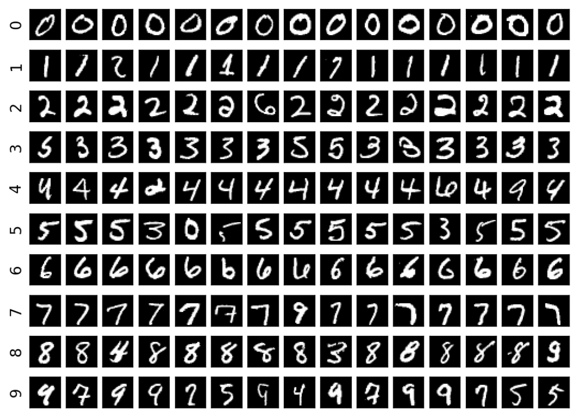

Figure 3: Digits grouped by predicted label

We can see that most of the misclassifications share some characteristics with the surrounding data and for some images it is even for a human hard to classify the number correctly. Compare for example the images in the first column for images labeled as "4" and "9". Although they look very similar they are classified differently and the label is ambiguous.

Now we will look at the classification report (Figure 4) and the confusion matrix (Figure 5) to quantify our notion of the results. We can see in the Classification Report that

```
             precision    recall  f1-score   support

          0       0.91      0.89      0.90       980
          1       0.86      0.97      0.91      1135
          2       0.89      0.79      0.84      1032
          3       0.77      0.83      0.80      1010
          4       0.82      0.82      0.82       982
          5       0.78      0.67      0.72       892
          6       0.88      0.89      0.89       958
          7       0.91      0.84      0.87      1028
          8       0.79      0.78      0.79       974
          9       0.75      0.85      0.80      1009

avg / total       0.84      0.84      0.84     10000
```

Figure 4: Classification Report

the labels 0, 1 and 7 have the highest F1-Score (around 0.9) and are therefor predicted most accurately. This could be explained by the shape of 1 and 7 that is very different from the rather round shapes that all other elements have in common. A possible explanation of the high F1-Score of 0 might be, that there aren't many ways to write a 0, but for example multiple ways to write a 4.

The worst F1-Score has the number 5 with a score of 0.72. This is caused by the very low recall of 0.67. The recall is defined as the ratio between True Positives (TP) and True Positives and False Negatives (FN) [2].

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

This means that it is more unlikely, compared to all other classes, that the classifier will detect the true class label of an input feature vector that should be labeled as a 5.

```
[[ 872    0    3    5    3   63   18    1   14    1]
 [    0 1102    8    3    0    3    4    0   15    0]
 [   15   28  816   37   26    8   31   18   49    4]
 [    4   22   28  835    1   29   10   14   45   22]
 [    2    8    6    1  808    2   15    1   20  119]
 [   22   22    5  128   29  602   20   16   20   28]
 [   15   20   16    1   20   30  852    0    4    0]
 [    1   41   15    3   17    0    2  862   18   69]
 [   15   23   11   68   12   31   10    7  759   38]
 [   12   14    5    9   64    8    1   26   15  855]]
```

Figure 5: Confusion Matrix

We can see from the confusion matrix that there are 63 cases where a 0 is interpreted as a 5 and 128 cases where a 3 is predicted to be a 5. Especially the confusion between a 3 and a 5 may be explained through the various ways to write the 5.

All in all we can conclude that our classifier is easily confused by similarities between numbers that are caused by the specific handwriting style.

4. MODEL SELECTION

Task. Use cross-validation to find a suitable value of the hyperparameter α (of the symmetric Dirichlet prior). Also plot the accuracy (as estimated via cross-validation) as a function of α. Discuss.

For cross-validation we split our labeled training data into K random folds that have a similar size. We use $K - 1$ of them to train our model and the remaining one to calculate

[1]http://yann.lecun.com/exdb/mnist/
[2]http://scikit-learn.og/MultionmialNB.html
[3]https://tensorflow.org/multilayer_cnn

the accuracy of our trained model. This procedure has the advantage that we can tune our hyperparamater without touching our test data. If we use our optimized model on the test data afterwards we obtain an unbiased result of how well our model works on real or previously unseen data.

```python
from sklearn.metrics import accuracy_score

i = 0
alpha = 2
alphas = np.zeros(K)
accurracies = np.zeros(K)

for i_train, i_test in Kf.split(X):
    X_train, X_test = X[i_train], X[i_test]
    y_train, y_test = y[i_train], y[i_test]

    x_model = nb_train(X_train, y_train,
                       alpha=i+alpha)
    x_pred = nb_predict(x_model, X_test)

    accurracies[i] = accuracy_score(
        y_test,
        x_pred['yhat']
    )
    alphas[i] = i + alpha
    i += 1
```

Figure 6: K-Fold Model Selection.

Figure 6 shows how we approach this with $K = 5$. For each fold we receive a labeled training set and a labeled test set that we can use to calculate the accuracy. Then we create a model and run a prediction on the test set and compare it with the true values. In the end we save the accuracy in a list with the corresponding alpha and afterwards we can just look up the maximum accuracy and use the alpha at this index.

In the notebook we try $\alpha \in [2, 6]$ and get 82,8% accuracy as the best result with $\alpha = 2$. The results for all α are shown in Figure 7.

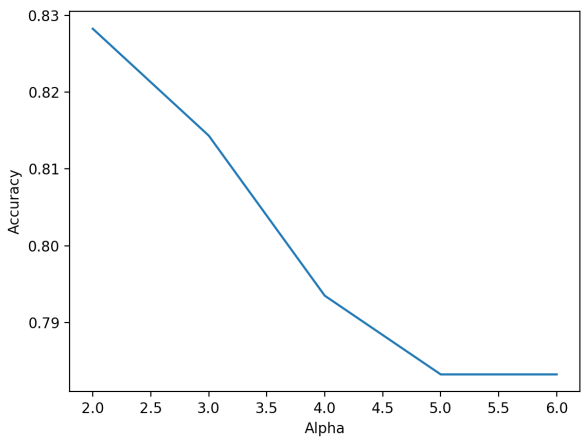

Figure 7: α - Accuracy

5. GENERATING DATA

5.a Generation

Task. Implement a function that generates digits for a set of provided list of class labels.

We use the log-likelihoods that are stored in our model to create vectors, given a class label. Here we can use np.random.choice(range(K), p=prob) to generate a random sample with the probabilities associated with each entry of the first parameter[4]. We do this generation for each element

```python
def nb_generate(model, ygen):
    logcls = model['logcls']
    n = len(ygen)
    C,D,K = logcls.shape
    Xgen = np.zeros((n,D))
    for i in range(n):
        c = ygen[i]
        for d in range(D):
            prob = np.exp(logcls[c][d])
            Xgen[i][d] = np.random.choice(
                range(K),
                p=prob
            )
    return Xgen
```

Figure 8: Generate data.

in the list of class labels and for each feature.

5.b Interpretation

Task. Generate some digits of each class for your trained model and plot. Interpret the result. Repeat data generation for different models by varying the hyperparameter α. How does α influence the results? Discuss.

The result of the digit generation is displayed in Figure 9. We can see that the resulting images look ”cloudy”, but most of them are still identifiable as the class label that they should represent. The ”cloudyness” can be explained

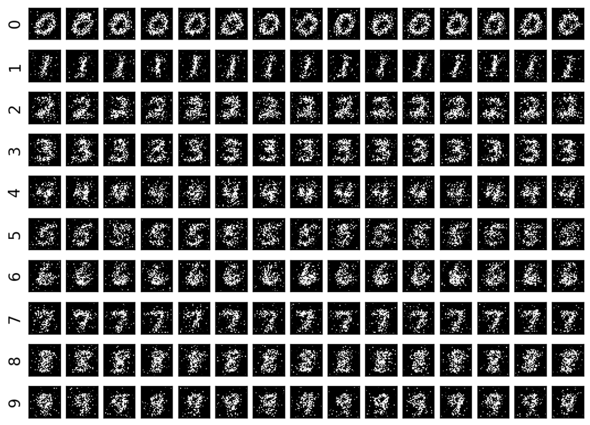

Figure 9: Some generated digits for each class

through the randomization that calculates the color or each pixel individually without looking left or right. This means that creating a line, where neighbouring pixels have a similar color, is not taken into account during the image generation.

[4]https://docs.scipy.org/numpy.random.choice.html

If we repeat this process for increasing values of alpha, the accuracy of our generated models decreases. While the white dots were clustered in the center for $\alpha = 2$ they appear also around the edges of the image for increasing alphas.

Interpreting α as a pseudo count of prior observations we can conclude that a large α leads to a larger prior belief for each possible pixel in our image. This is caused by the usage of the symmetric Dirichlet prior. Given more training data we would expect the number of points that are close to the edges to go down and to have them clustered around the middle and to resemble to class label for closely.

This is also supported by the observation from Section 4 that larger αs reduce the accuracy of our predictions.

6. PREPROCESSING FEATURES AND GAUSSIAN NAIVE BAYES

Task. Try to find a better Naive Bayes classifier. E.g., you may try to preprocess features or to use a continuous Naive Bayes classifier. Can you improve on the Naive Bayes classifiers obtained in the previous tasks? What is the best accuracy you can get?

6.a Preprocessing

One way to preprocess the features is to binarize them. In our input set we deal with features in a range from 0 to 255 representing different shades of grey. If we binarize them we may reduce the complexity and think of each pixel in the input image as being either black or white.

We use the sklearn.preprocessing.Binarizer to make each feature either 0 or 1[5].

This results in an accuracy of 84.2% which is around 0.6% better than the accuracy on unpreprocessed data.

6.b Gaussian Naive Bayes

The Gaussian Naive Bayes implementation in the sklearn.naive_bayes package assumes that all features have a Gaussian distribution and estimates the mean and variance using maximum likelihood[6].

If we use it on the unprocessed input data like in Figure 10 we obtain an accuracy of 55.6% which is worse than our accuracy of 83.6% for the unnormalized categorical model.

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X, y)
yhat = nb.predict(Xtest)

accuracy_score(ytest, yhat)
```

Figure 10: Applying Gaussian Naive Bayes.

The best result we obtained is 84.2% accuracy with a binarized input and a categorical Naive Bayes implementation.

[5]http://scikit-learn.org/Binarizer.html
[6]http://scikit-learn.org/naive_bayes.html

7. REFERENCES

[1] K. P. Murphy. *Machine Learning: A Probabilistic Perspective.* The MIT Press, 2012.

[2] D. M. W. Powers. Evaluation: From precision, recall and f-measure to roc., informedness, markedness & correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.