CS5014 Practical 2

170011474

April 2021

1 Pre-process

Some features, such as objects' positions, sizes, and ids are not important for this task. They are barely detailed descriptions about objects rather than key factors of colors and textures. Therefore, they are removed from the training samples.

Rows containing missing values are removed to ensure that no `NULL` values will affect the training process. Before splitting the dataset into training set and testing set, **to prevent data dependency existing between rows**, I shuffled the whole dataset.

Since the dataset overall is insufficient, 5 fold cross validation technique is adopted to enhance its utility. Therefore, the whole training set is split into 5 equal folds and each fold will be used as a testing set at each iteration.

To have a quicker converging, a standard scaling is used to normalise numerical values. This step would result in convergable iterations. **The most important reason to do this is that if one of the features has a broad range of values, the distance governs this particular feature**. Besides, when to do gradient descent, there were be very large steps in one dimension but minor steps in other dimensions.

2 Support Vector Machine

Support Vector Machine (SVM) has some outstanding advantages for this task:

Dataset Size SVM is particularly good for small datasets. The reason is that SVM uses kernel tricks which usually maps low-dimension data points to high-dimension points and hence provides a distance representation. During the process, for example, RBF, will compute points' inner dot. If the dataset has n points, then the kernel matrix M will have size $n \times n$. Consequently, it scales quadratically with the number of data points so it will handle large datasets poorly. This dataset is relatively small. Compared to other models, SVM will have better generalisation on it.

High-Dimensional Data SVM scales relatively well to datasets with a large set of features. Since its kernel trick provides a way to represent pairwise distances, it brings convenience when data is high-dimensional. For example, there are 10 points so 45 distances, after mapping them to a high-dimensional space, there are still 45 distances. It saves intermediary computation and results in the faster training process. Our current dataset has more than 400 features, so SVM suits it.

Kernel Trick A perfect kernel function is one of strengths SVM can have. With it implemented, SVM can model any complicated dataset. It is usually a good start point when we are unfamiliar with a new dataset.

3 Hyper-Parameters Setting

C The parameter `C` refers to its regularisation parameter and the penalty is automatically set to 1.0. The penalty is squared L2 penalty. 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. Conversely, smaller C will cause a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. For very tiny values of C, you should get misclassified examples, often even if your training data is linearly separable. `C` is set to be 1.1 here. This results came from a grid-search like procedure.

kernel There are multiple kernel functions to be chosen. Its default value is rbf. Customised kernel functions can be used to replace any of pre-defined functions. There are also other available options, such as linear and poly. The reason why I chose rbf is that compared to others, it brought higher accuracy. The linear kernel only suits the linear separable dataset best. RBF SVM creates non-linear combinations of features to uplift training samples onto a higher-dimensional feature space where we can use a linear decision boundary to separate different classes. Under the current case, the poly kernel reduced the accuracy, so rbf was chosen.

class_weight Since the distribution of different class samples is very different, the algorithms tend to get biased towards the majority classes present and don't perform well on the minority ones. Consequently, class's frequency is highly imbalanced. By introducing `class_weight`, it penalizes the misclassification made by the minority class by setting a higher class weight while reducing weight for the majority class. Large weights result in a large penalty and a large update to the model coefficients.

gamma Intuitively, it defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. Its default value is $1/n$ where n is the number of features. Here, the default value works best from a grid-search algorithm.

4 Regularisation and Optimisation Strategies

L2 is adopted defaultly by SVM. Its regularization term is the sum of square of all feature weights as shown below:

$$L(x, y) = \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^{n} \theta_i^2$$

It performs better when all the input features influence the output and all with weights are of roughly equal size. The reason why SVM with L2 regularisation is more common is that it is differentiable and imposes a bigger (quadratic vs. linear) loss for points which violate the margin [1].

To fine tune its hyper-parameters, grid-search like algorithms were used to find best parameters, mainly focusing on `C` and `gamma`.

5 Evaluation

The evaluation metrics adopted here is balanced accuracy. The balanced accuracy in binary and multiclass classification problems to deal with imbalanced datasets. The reason is similar to the one mentioned in `class_weight`. For example, if one problem is to predict if one person is male or female, and the whole dataset consists of 90 men and 10 women. If a model simply predicts all instances to be men, it still has 90% accuracy. However, this does not reflect the model's genuine performance. In order to solve this problem, the balanced accuracy would take care of this. It is defined as the arithmetic mean of sensitivity (true positive rate) and specificity (true negative rate),

$$balanced\_accuracy(\hat{y}_i, y_i) = \frac{1}{2}(\frac{TN}{TN+FP} + \frac{TP}{TP+FN})$$

It is not a coincidence since it is verified through 5 fold cross validation. At each iteration, its balanced accuracy is roughly the same. Besides, the dataset is shuffled so no dependency inside. Therefore, it is repeatable. Admittedly, there are some easier

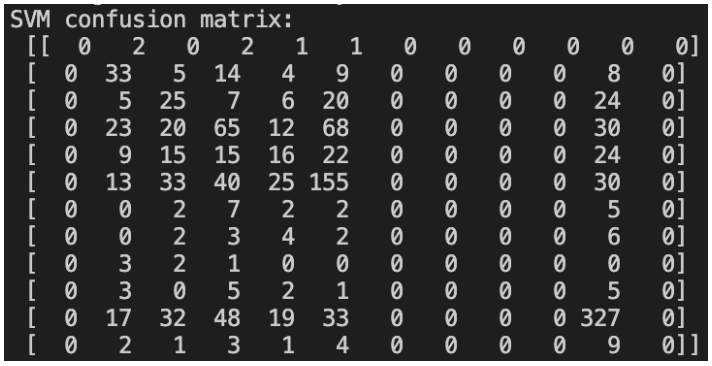

Figure 1: Confusion Matrix

classes and this is because the unbalanced dataset. The discussion is listed in the section 3's `class_weight`. For example, if one class has more samples, then the model can safely predict all testing sample as that class to reach a relatively high accuracy.

The balanced accuracy of my model is 0.223 for the color task and 0.308 for the texture task. The accuracy is 0.472 for the color task and 0.483 for the texture class. Those results can refect my opinion above.

The class labels are: ['beige' 'black' 'blue' 'brown' 'gray' 'green' 'orange' 'pink' 'purple' 'red' 'white' 'yellow'].

The model does not perform equally in all classes. This confusion matrix of SVM states that there are 6 classes do not have any correct predictions: 'beige', 'orange' 'pink' 'purple' 'red' 'yellow'. They have few training samples and hence SVM's soft margin will easily ignore them. Therefore, the SVM classifier suits best for balanced datasets. **This is one of weaknesses of my model.**

6 Comparison Between SVM and Logistic Regression

The conditions: running 100 times with the same training set and testing set, 5 fold cross validation. The final results show that the SVM classifier gave a balanced accuracy about 0.211 while the LR classifier gave 0.151 (color). The SVM gave 0.298 while the LR gave 0.204 in texture prediction. Therefore, those statistic values show that SVM works better than a simple LR.

The most important factor that affects LR's performance is that LR is essentially a linear model, because its outcome depends on the sum of the inputs and its parameters. In other words, the output cannot depend on the product of its parameters. The current task is a multi-class problem and not linear-separable, so SVM could achieve better scores.

7 Comparison Between SVM and GaussianNB

Given the same training set, the final results show that the SVM gave a balanced accuracy about 0.211 while the GaussianNB gave 0.198 after 100 trials in the color task. SVM gave 0.31 and GaussianNB gave 0.282 in the texture task.

The fundamental difference between them is that Gaussian NB assumes independence of each pair of features, while SVM looks at the interactions between them to a certain degree. This difference leads to distinct performance based on the same dataset. The current features are related to color space, histograms of gradients, and so on. Themselves are internally dependent, so the SVM would perform better. Another difference

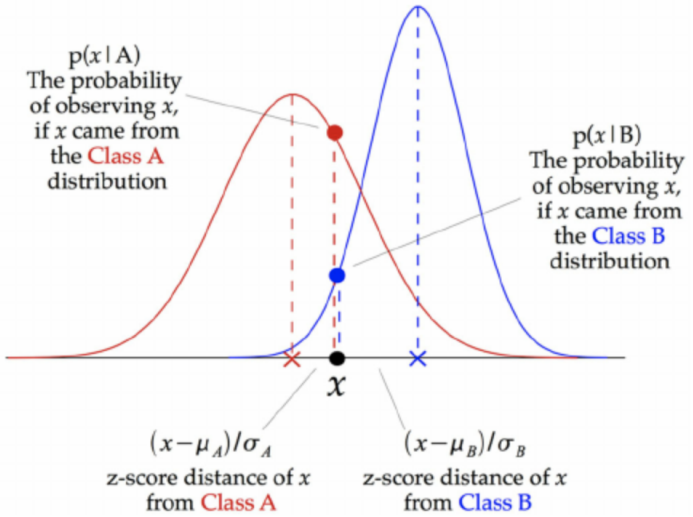

Figure 2: Gaussian NB

is Naive Bayes algorithms assume that each feature is given the same weight. From its first assumption, attributes are irrelevant and and they are assumed to be contributing equally to the outcome. However, the current dataset's features are relevant, color space and HOG will contribute distinctly to different tasks, which will be further explored in the next section. Therefore, Gaussian Naive Bayes will not achieve the same results as SVM does.

The Naive Bayes can be used to solve this problem: given a data point, what is the probability of x belonging to some class c? This is equivalent to a conditional prob-

ability $p(c|x)$. If there are 3 classes c_1, c_2, c_3, then the Naive Bayes classifier will try to find $p(c_i|x)$ directly and choose the one with the highest probability as an answer. That is:

$$Prediction(x) = argmax\ p(c|x)$$

Then use the Bayes's theorem:

$$p(c|x) = \frac{p(x|c)\ p(c)}{p(x)} = \frac{p(x|c)\ p(c)}{\sum_c p(x|c)\ p(c)}$$

Now we want to find $p(c)$ and $p(x|c)$. $p(c)$ can be found from ground-truth. Assumes x has 2 features so $p(x|c) = p(x_1, x_2|c)$. The naive assumption says that features are independent so

$$p(x_1, x_2|c) = p(x_1|c)p(x_2|c)$$

for every class c. This assumption is not hold in general cases. Now we only need to find $p(x_i|c)$.

Naive Bayes can be extended to continuous values, most commonly by assuming a Gaussian distribution, called Gaussian Naive Bayes. Other functions can be used to estimate the distribution of the data, but the Gaussian (or Normal distribution) is the easiest to work with because you only need to estimate the mean and the standard deviation from your training data [2]. The equation is:

$$P(x_i|c_i) = \frac{1}{\sqrt{2\pi\sigma_c^2}}\ exp(-\frac{(x_i - \mu_c)^2}{2\sigma_c^2})$$

where μ is the mean and σ is the variance. Each feature will have its own mean. Therefore, there are 6 means (each data point has 2 features and here are 3 classes).

8 SVM Explanation

Margin is the distance from the hyperplane to the nearest training data point. In Logistic Regression, its margin tend to be relatively small since it focuses on minimizing some loss function rather than maximizing its margin. Each data point contribute equally to the loss function, although far data points will not have a far influence as the same as close-margin data points. To overcome this, SVM is developed.

The ideology behind SVM is to find a hyperplane which best separate training points. Ideally, we are supposed to maximize the margin so it can be better applied to testing set. Only a subset of data points are used. Those nearest data points are seen as support vectors.

The equation of a hyperplane can be:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_m x_m = x_i^T \theta + \theta_0$$

where m means m features. Here we have a hard constraint:

$$y_i(x_i^T \theta + \theta_0) \geq 1$$

to ensure that each data point is classified correctly. However, this only suits for linear-separable dataset. Therefore, we introduce slack variables $\xi = [\xi_1, ..., \xi_n]$ into SVM. This would allow some outliers. This can be seen as error terms as a soft margin:

$$y_i(x_i^T \theta + \theta_0) \geq 1 - \xi_i$$

and

$$\xi_i \geq 0, \sum_{i=1}^{N} \xi_i \leq const$$

So now our loss function becomes:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} (1 - y_i(x_i^T \theta + \theta_0))_+ + \lambda ||\theta||^2$$

This soft-margin SVM uses hinge loss function and a L2 regression. The hinge loss adopts a max function, equivalent to $max(1 - y_i(x_i^T\theta + \theta_0), 0)$. For a random data point i, if $y_i(x_i^T\theta + \theta_0) \leq 1$ then

$$L_i(\theta) = -y_i x_i^T + 2\lambda\theta$$

else the gradient is:

$$L_i(\theta) = 2\lambda\theta$$

Optimisation strategies including tuning hyper-parameters such as `C` and `gamma`. The strategy is to use grid-search like algorithms to find best parameters, which basically brute-forcely iterates over each combination of parameters and sees its accuracy. Another strategy is to optimize its kernel function via trying different kernel functions or customized ones. Different kernels apply to different situations. The difference between RBF and Linear kernel functions are demonstrated above in Section 3.

9 Feature Exploration

Different features often contribute differently to the main task. An rigorous experiment was conducted. In the texture task, there are 3 main subset of features: 27 values for color space, 288 values for HOG, and the rest for cell responses. I conducted trails to test each subset's effect on the final results. Here is the table:

Task	color space	HOG	cell responses	all
color	0.162	0.210	0.133	0.223
texture	0.277	2.86	0.217	0.308

We can see from the table that HOG contributes most to the 2 tasks (2.86 and 0.193), and the cell response features contribute least.

Among all possible combinations of 2 features, the combination of the color space feature and HOG feature achieve the best performance. It would give 0.208 for the color task and 0.306 for the texture task.

The reason why HOG performs so well is that it represents the structural feature of the edge (gradient), so it can describe the local shape information. The quantization of the position and direction space can suppress the influence of translation and rotation to a certain extent. The normalized histogram in the local area can be used to partially offset the impact of light changes. Because of its block and unit processing method, the relationship between the local pixels of the image can be well characterized. Therefore, it would easily describe objects' textures and their colors.

Back to color histograms values. A color histogram is a representation of the distribution of colors in an image. The color space is split into multiple color ranges and in each color range, the number of pixels that have those colors will be represented. Textures will have relatively stable color distribution for each local component, so color histograms can also master their features.

References

[1] Tang, Y., 2013. *Deep Learning using Linear Support Vector Machines.* [online] arXiv.org. Available at: https://arxiv.org/abs/1306.0239v4 [Accessed 13 April 2021].

[2] Brownlee, J., 2020. N*aive Bayes for Machine Learning. [online] Machine Learning Mastery.* Available at: https://machinelearningmastery.com/naive-bayes-for-machine-learning/ [Accessed 15 April 2021].