**Support Vector Machines(SVMs)**

**Algorithm analysis:**

SVMs try to separate the data with a superlevel. What the superlevel they are "looking for" is the one whose distance from the nearest points of the dataset (which are called support vectors and this distance is called margin) is maximum. This separation is easily done when our data is linearly separable. Svm to solve this problem project the data in more dimensions in order to turn the problem into linearly separable, in order to find the optimal superlevel for the classification. This of course is not always easy and there are misclassifications, the percentage of which we can adjust to achieve better generalization to unknown data. To find the hyperplane is cross validation.

**Advantages:**

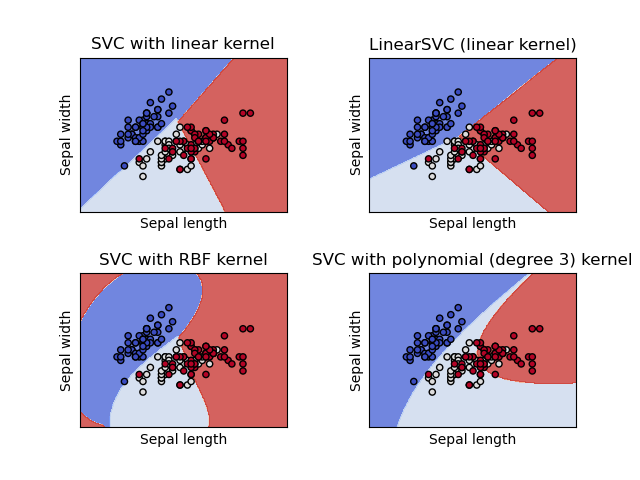
* Effective in high-dimensional spaces and in cases where the number of dimensions is higher than the number of samples.
* They are memory efficient as they use a subset of training points for decision making.
* They are flexible as there are various kernel functions that can be used to modify decision making.

**Disadvantages:**

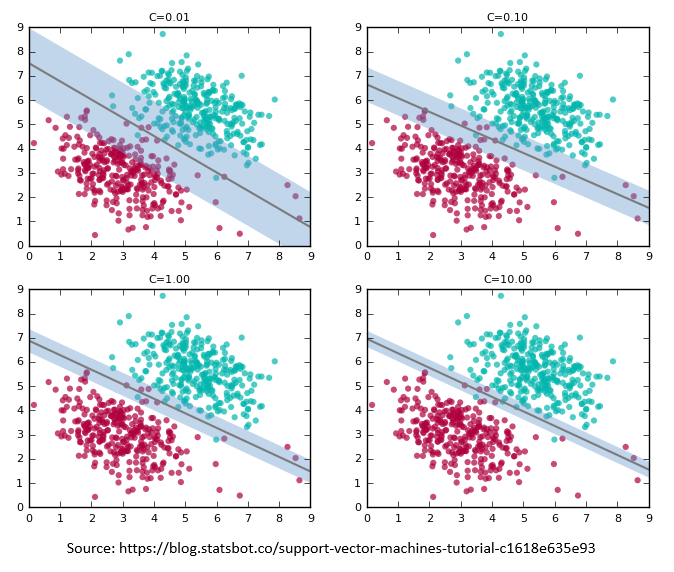
* If the number of features is greater than the number of samples it is very common to overfit
* They do not provide immediate prediction probabilities and complex/time-consuming 5-fold cross-validations are required to calculate them.

**Analysis of SVM parameters:**

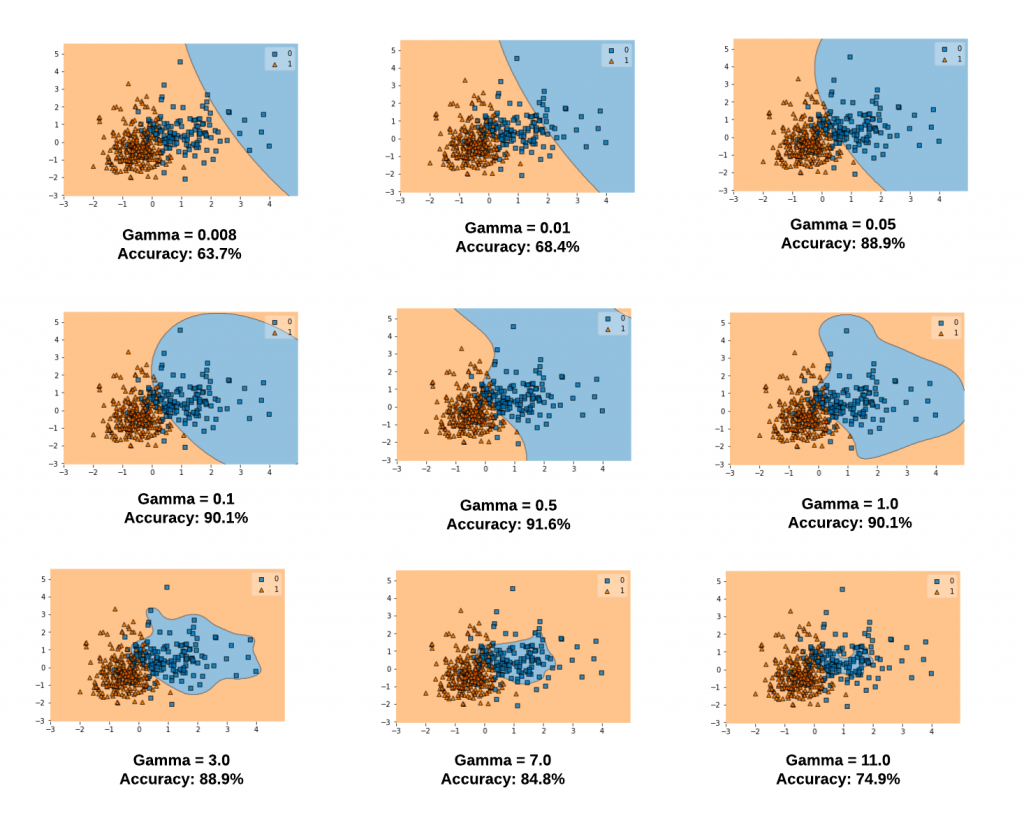
**Kernel:** is the function that we use to project the data in more dimensions and bring our problem to a linearly separable form. Depending on the function there may be some extra parameters for calibration e.g. In the polynomial kernel we can determine its degree, which is the number of additional dimensions to be used to solve.



**C:** applies a kind of regularization to SVM. Indicates how tolerant svm is in misclassification in each training sample. For high values of C the SVM will choose a smaller margin, aiming to have fewer misclassifications but with its cost not generalizing well to unknown data ( overfit). On the contrary for small values of C will choose a larger margin, having more misclassifications but perhaps generalizing better.



**Gamma**: is a parameter used in RBF kernel. It indicates the amount of curvature that we want the decision boundary to have and in fact the "influence interval" that each feature will have. Greater gamma means greater curvature and on the contrary for a smaller one.



(Effect of different values of gamma, we observe greater curvature for larger values)

Data preprocessing is a key part in the creation of a good model as it can help significantly both in the training time and in the effectiveness of the algorithm. Especially in the dataset of cifar 10 which is quite large and the implementation of an SVM in 50000,32,32,32,3 dimensions will be very time consuming, it is necessary to use techniques to reduce its dimensions training set keeping as much "useful" information as possible. The work used several basic methods to achieve this goal which we will comment on below.

**Reduction of samples:** The sample is very large and the training takes too much time so a simple reduction of the training set samples from 50000 to 5000 will significantly reduce this time and will allow us to do more tests and tests at the cost of the lower accuracy of the models we will create

**PCA:** One of the most popular and successful techniques for dimensionality reduction. The Principal Component Analysis succeeds in significantly reducing the dimensions, keeping the largest possible percentage of the "important" information. This makes the dataset more easily manageable, the training time is dramatically reduced and visualization becomes easier of the data. Its function can be described by a few simple steps. First we have to do standardization so that each variable contributes similarly to the analysis and there is no bias. Then we have to find the covariance matrix in order to see the correlations between them and see where there is useless information. Finally we have to calculate the eigenvalues and eigenvectors of the table to get the principal components components which are linear combinations of the original variables.

Chart, diagram

Description automatically generated

(PCA Algorithm)

**Grayscale:** We remove the last 3 dimensions (RGB), assuming that color does not provide meaningful information about our model and significantly reduce the complexity. Exactly the same technique was used in neural networks and in knn and nearest centroid algorithms.

SVMs are not scale invariant, so scaling plays an important role in the performance of the algorithm; the same scaling technique must of course be applied to the test set in order for our observations to be meaningful.

**Normalize:** Since all samples are in the range of values [0.255] as they represent the brightness of the pixels, we can bring all our data to the same scale and close to 0 and 1 simply by dividing each of them by 255, making the range now be [0.1].

**Standard Scaling:** With standard scaling we change the medium of our data and go its center to 0 and its standard deviation to 1. So now our training set follows normal distribution and increases the efficiency of the algorithm.

**Experiment process:**

At the beginning of the experiment, tests were carried out with a very simple SVM model which had all the default settings beyond the decision function = ovo (C=1,kernel=rbf,gamma=scale). The results we got were disappointing as apart from the fact that the model achieved only 14.4% accuracy but it also did 1 hour and 35 minutes train. Note that at this stage there was no data preprocessing and the dataset was complete. Then standardization was tested making the sample follow normal distribution (again in full dataset) but to our surprise the train it took 3+ hours+ to complete something that was completely unexpected, as I would expect improvement in both the accuracy of the model and the time. Some other experiments were done on the whole dataset without major changes in hyperparameters (mainly change of kernels and data preprocessing techniques) without significant improvement in both accuracy and time.

At this point it was decided to use a smaller sample of the dataset (5000 samples) to reduce the training time and to make it possible to search for parameters and better models for the representation of the dataset.

param\_grid = {'C': [0.1, 1, 10, 100],

'gamma': [10,1, 0.1, 0.01, 'scale'],

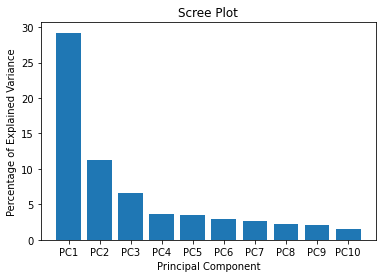
'kernel': ['rbf','sigmoid'],

'cache\_size': [2000]}

In preprocessing we took 5000 samples from the training set, we normalize them (/255), we grayscaled them (removal of RGB dimensions) and then we applied a grid search to find C, Gamma and kernel. In this test we found the best model to have C=100, gamma = 0.01, kerner = rbf and this model achieved accuracy of 44.92% . It seems that bigger C, i.e. smaller margin and misclassifications, is more efficient in dataset our. Also small gamma, therefore relatively small curvature in decision boundaries, is preferable. These above findings lead us to assume that our problem is relatively linear but something that is not true and we can see it from the generally bad accuracy achieved by SVM. Finally, it seems that rbf kernel is by far better than sigmoidoid , which the best percentage that succeeds is 10.4% and in many tests has accuracy less than 10%. As far as time is concerned we see that for the implementation of 90 fits with 5-fold cross-validation we want about 4 hours, while at the beginning of our experiment we needed 1.5 hours for only 1 fit.

With these findings in mind we can explore above the changes in the hyperparameters, looking for larger values of C and smaller of Gamma. It is also worth investigating the differences of preprocessing methods. That's why PCA was then applied where we expect to see much better time and almost the same results in what we have found so far.

**PCA investigation :**



(Variance rate from the first 10 principal components)

The first step of the PCA is to bring our sample either on a scale of the form [-1,1], or to follow a normal distribution or to do it simply normalize (generally we have to bring it to a uniform scale). Making for a start simply normalize in our trainset (5000 samples) we notice that it achieves 95% variance ratio with only 203 principal components and therefore dimensions. In other words, we managed to reduce our dataset from 3072 dimensions to only 203 with a very small loss of information. But before we start the new finding of parameters we can try other scaling tactics to see if we can further reduce the dimensions with the same percentage of variance or not. So by trying to bring our sample to follow a normal distribution we see that it succeeds again 95% variance ratio but with 317 principal components and therefore dimensions. Therefore we see that it is a much better choice to use the first form of PCA (i.e. with normalized training set) at 203 dimensions. Some further preprocessing is not necessary as already the dimensions are significantly lower and we managed to preserve the information.

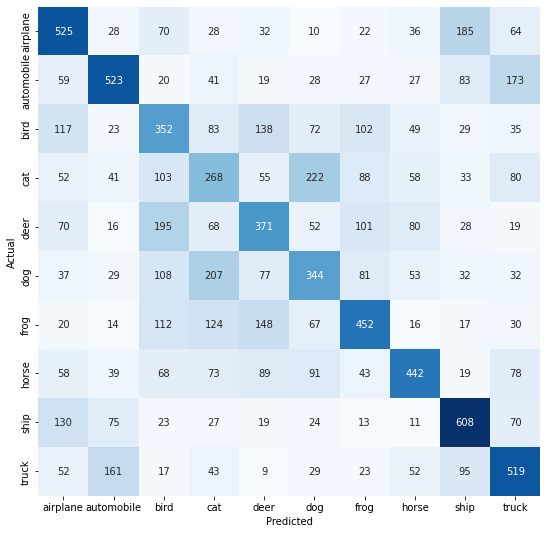
param\_grid = {'C': [100, 1000, 0.01],

'gamma': [0.0001,0. 001,0.01,100,'scale'],

'kernel': ['poly', 'rbf'],

'cache\_size': [2000]}

Regarding the findings of this grid search, we initially see the value of the PCA as the time was dramatically reduced (from 250min to 7 min for 90 fits, cv=5) while the performance of the model almost did not fall at all despite the significant decrease in dimensions (44.6% instead of 44.9%). =1000 has exactly the same accuracy as 100 and all other parameters the same, which indicates that a further increase will probably lead to worse or identical results. In gamma we confirm that bigger gamma brings worse results while for gamma = 0.001 and below, we see that our model begins and deviates from the best value we found so far. The rbf kernel proves once again its value by showing much better results than the kernel poly, which may need further searching in its hyperparameters such as degree and coef for better performance. In particular, the poly kernel occupies the middle positions in the test score ranking with success rates of around 35%. Another thing we notice is that our models make a huge overfit as the train score is around 99% in many of the cases but as we observe they do not generalize to a satisfactory degree! It is noted here that this phenomenon is mainly observed in poly kernel.



(confusion matrix of the best model found by grid search)

**Comparison of algorithms:**

As it is easily understood SVM certainly has more power than all 2 KNN, Nearest Centroid algorithms. Their biggest advantage is that they are universal approximators and theoretically can approach any data as long as they become tuned appropriately. That is, their biggest advantage is that they are flexible . At the same time, this is their biggest disadvantage as it is quite difficult to adapt all the parameters correctly and to find a model that generalizes the data well. As a result, in large datasets and without preprocessing of data, the runtime for training is very large, making it even more difficult to search for hyperparameters. Comparing their performance, of course, we can see that SVMs can create much better models than the other 2 algorithms at a cost of course per year. Nearest Centroid for example, with minimal preprocessing (only grayscaled) and throughout the dataset, achieved accuracy of 30% in 0 second while very few models and after some searching time achieved this in SVMs. Here it is noted that in general the KNN and the Nearest Centroid they achieved 30% accuracy while svm had large variations depending on the parameters but reached percentages of 44%, so surely we can say that in more difficult problems SVMs generalize better by finding the appropriate parameters as long as the resources are in place for this implementation.