

# SVM Kernel Comparison — Tutorial and Experiments

---

Understanding Linear, RBF, and Polynomial kernels with synthetic datasets

## 1. Introduction and Aim of the Experiments

Support Vector Machines (SVMs) are supervised learning models that construct a separating hyperplane between classes. In their simplest form they learn a linear decision boundary, but by using kernel functions SVMs can represent highly non-linear decision surfaces as well. The notebook associated with this report explores three common kernels — Linear, Radial Basis Function (RBF), and Polynomial — and compares how they behave on several synthetic datasets.

The central goal of the notebook is twofold. First, it provides an end-to-end, fully reproducible pipeline for training and tuning SVM classifiers using scikit-learn. Second, it creates visualisations that make abstract concepts such as margins, support vectors, and kernel-induced non-linearity more intuitive. The report you are reading is designed to accompany those plots: it explains what each experimental block is doing and suggests where each figure should be placed inside the document.

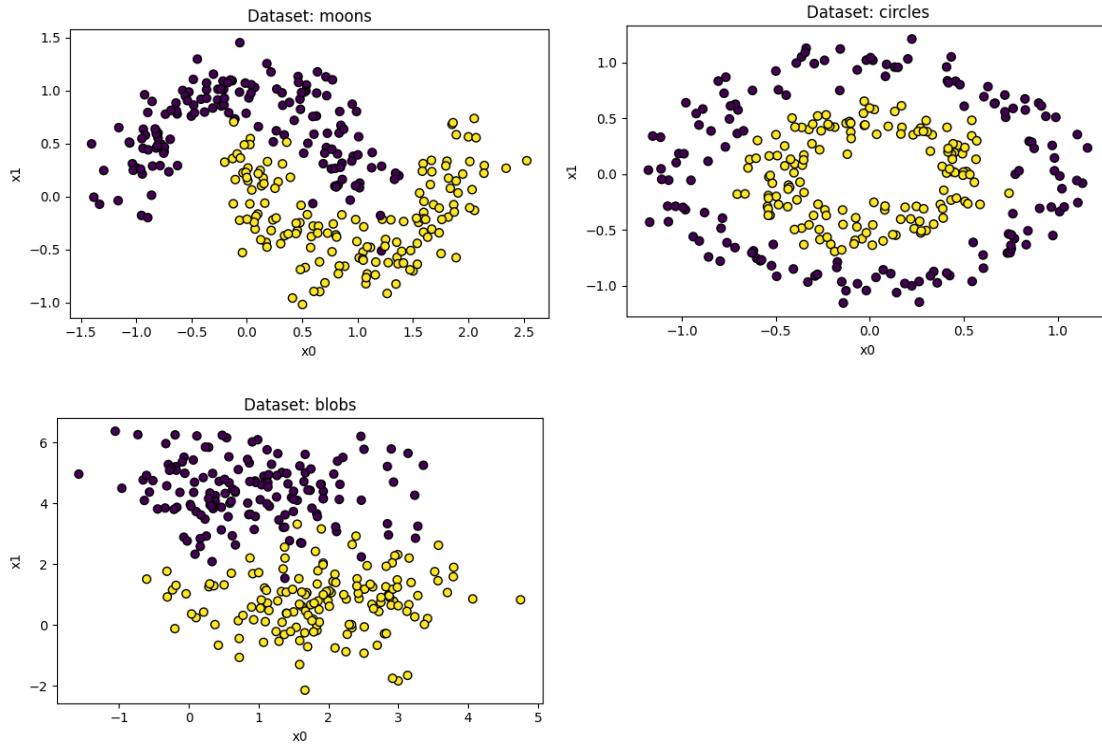
In all experiments the workflow is consistent. Features are standardised, SVM models are wrapped in a scikit-learn Pipeline, and hyperparameters are tuned with GridSearchCV and stratified cross-validation. The resulting best models are evaluated on held-out test data, and their decision boundaries and confusion matrices are exported as image files.

## 2. Synthetic Datasets Used

The notebook focuses on three small two-dimensional synthetic datasets. These are chosen because they are easy to visualise and because each one highlights a different aspect of kernel behaviour:

- Moons: two interleaving half-circles, generated with `make_moons`. This dataset is strongly non-linear and cannot be separated by a straight line in the original feature space.
- Circles: two concentric circles, generated with `make_circles`. This dataset is radially symmetric and demands a curved, ring-shaped decision boundary for perfect separation.
- Blobs: two Gaussian clusters, generated with `make_blobs`. This dataset is close to linearly separable and is therefore a natural candidate for a linear kernel.

Each dataset contains the same number of samples in each class. This makes accuracy, confusion matrices, and other performance measures easier to interpret, because there is no severe class imbalance.



### 3. Implementation and Experimental Pipeline

All experiments are implemented in Python using NumPy, pandas, Matplotlib, and scikit-learn. A random seed is fixed at the start of the notebook so that the generated datasets and cross-validation splits are reproducible. An artifacts directory is created on disk, and all important outputs — plots, trained models, and the summary CSV file — are written into this directory for later reuse.

The classification model itself is provided by `sklearn.svm.SVC`. To avoid data leakage and to ensure a clean training pipeline, each SVC instance is wrapped inside a scikit-learn Pipeline together with `StandardScaler`. This means that feature scaling is learned purely from the training folds and applied both to validation data inside cross-validation and to the held-out test set. Hyperparameters are exposed through the pipeline using names such as `svc_C` or `svc_gamma` so that `GridSearchCV` can tune them systematically.

`StratifiedKFold` is used to construct cross-validation splits that preserve the proportion of classes in each fold. This is important even for balanced datasets, because it reduces variance between folds and avoids degenerate splits where one class might be under-represented. For each dataset and each kernel family, `GridSearchCV` runs over a small but representative grid of hyperparameters and records both cross-validated performance and the hyperparameters associated with the best score.

## 4. Kernel Choices and Hyperparameter Grids

The notebook compares three kernels: linear, RBF, and polynomial. In scikit-learn, the Linear kernel is obtained by setting `kernel='linear'` in SVC. This kernel learns a flat hyperplane in the original feature space. The primary hyperparameter is `C`, the regularisation strength. Small values of `C` produce wider margins and more tolerance for misclassifications, while large values of `C` prioritise fitting the training data and may lead to narrower margins.

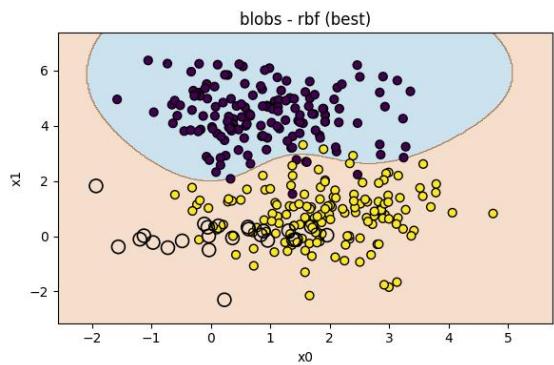
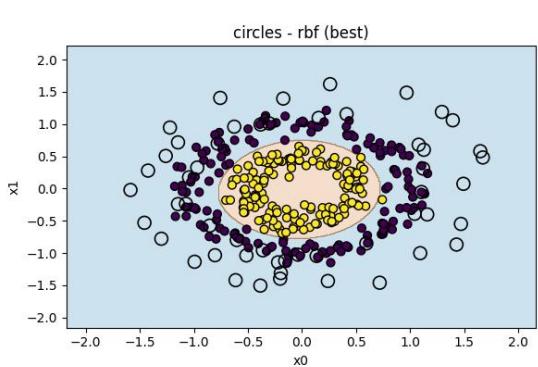
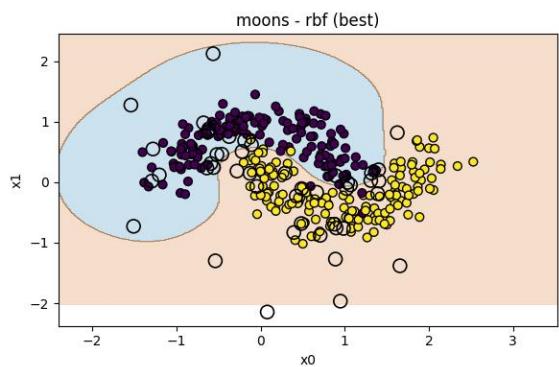
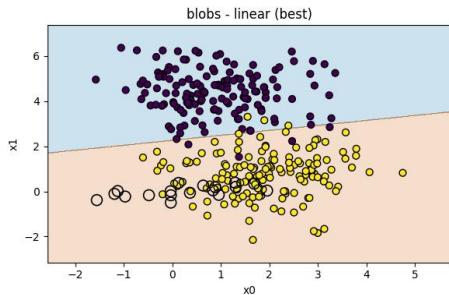
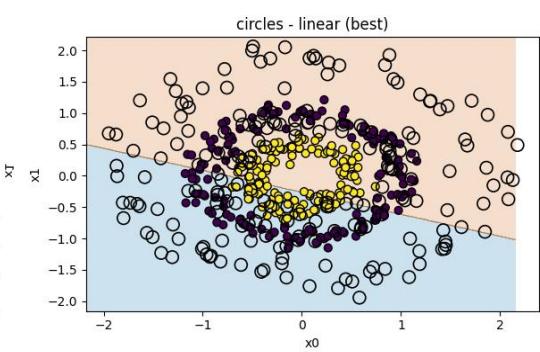
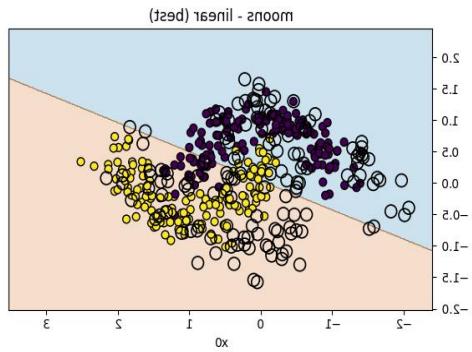
The RBF kernel is specified by `kernel='rbf'` and adds a new hyperparameter `gamma` in addition to `C`. `Gamma` controls how far the influence of a single training point extends. When `gamma` is small, each point influences a broad region and the resulting decision boundary is smooth and slowly varying. When `gamma` is large, points influence only a tiny neighbourhood, allowing the boundary to bend sharply around the data but at the cost of a higher risk of overfitting.

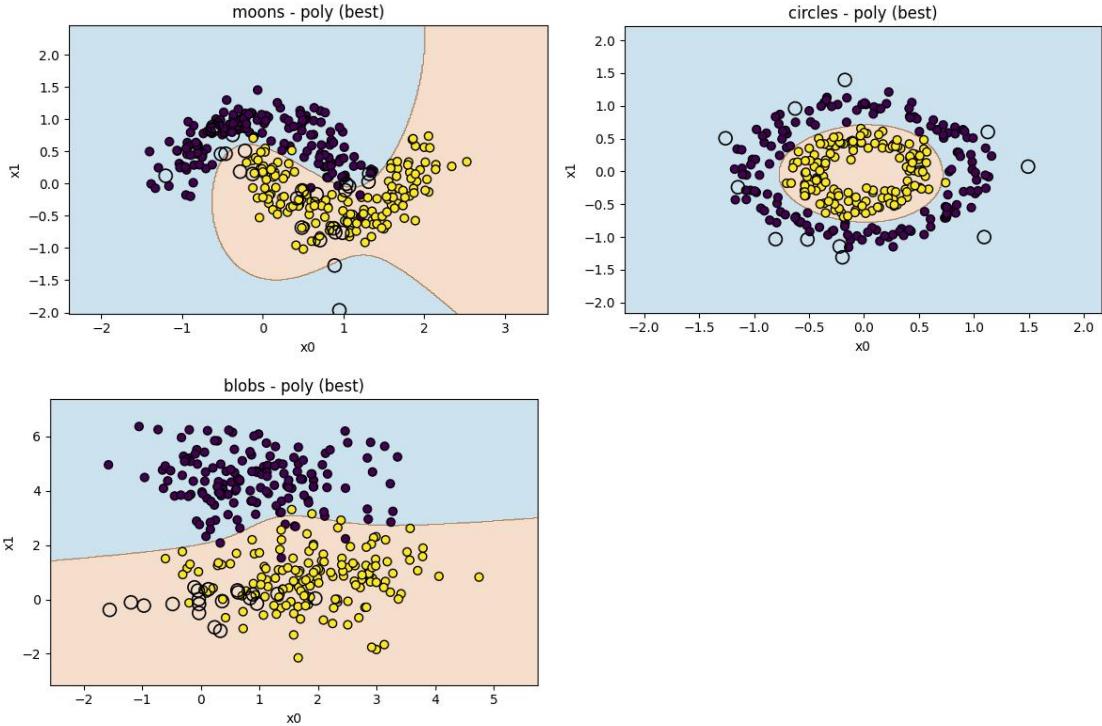
The polynomial kernel is specified with `kernel='poly'` and introduces `degree`, `gamma`, and `coef0`. The `degree` parameter controls the order of the polynomial. Degree two or three is often sufficient for many problems; higher degrees produce richer boundaries but are more prone to overfitting and numerical instability. The `coef0` parameter shifts the polynomial and can change the shape of the decision surface, especially for low degrees. In the notebook, the parameter grid includes a small set of candidate values for each of these hyperparameters so that `GridSearchCV` can search over them.

## 5. Visualising Decision Boundaries and Support Vectors

A key part of the notebook is a helper function that visualises decision boundaries and support vectors. It constructs a fine grid over the range of the two input features, asks the trained classifier to predict the class for each grid point, and then uses a contour plot to colour the regions by predicted label. The original training points are plotted on top of these regions, and the support vectors identified by the SVM are highlighted with special markers.

Because the datasets are two-dimensional, these plots give an immediate, intuitive picture of what each kernel is doing. On the blobs dataset, a linear kernel typically draws a roughly straight line that separates the two clusters. On the moons dataset, the RBF kernel bends around the curved shapes and follows their geometry closely, while a linear kernel cuts through both moons and misclassifies many points. On the circles dataset, a successful model must form a ring-shaped boundary: again, the RBF kernel is usually far more suitable than a purely linear model.



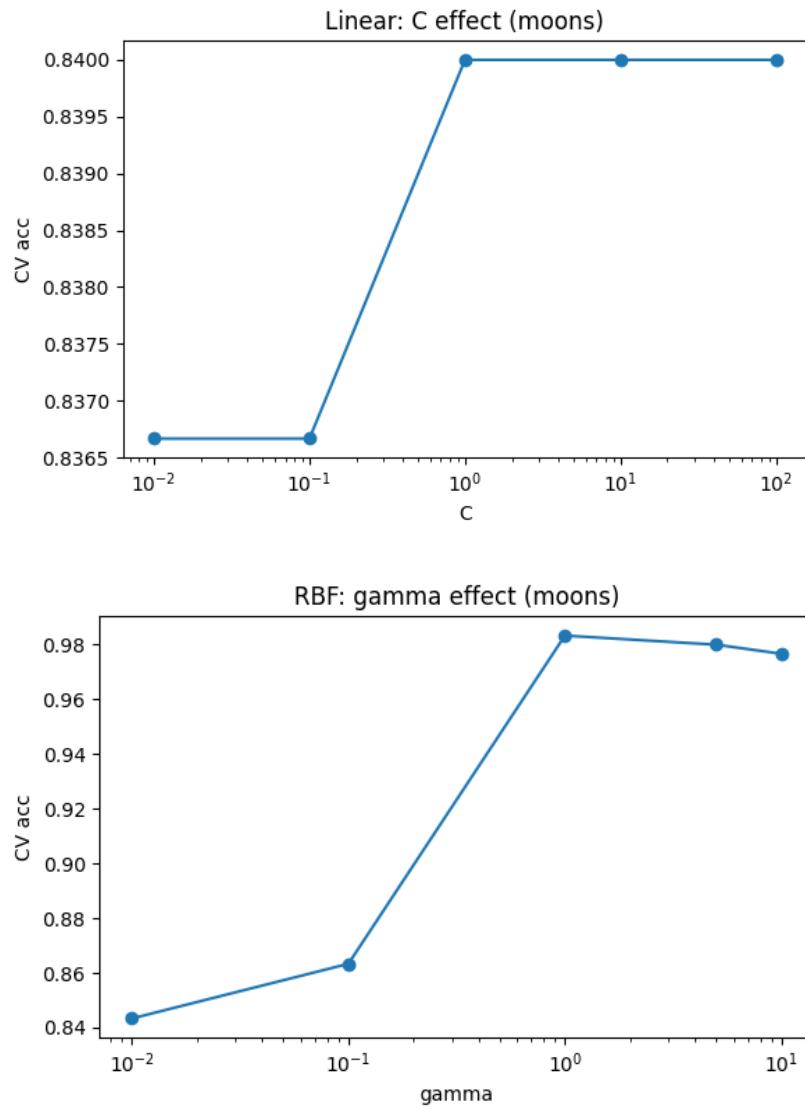


## 6. Effect of Regularisation (C) and Gamma

The notebook contains focused experiments on how key hyperparameters affect performance. On the moons dataset, the code loops over a range of  $C$  values for a linear SVM and uses cross-validation to estimate accuracy for each value. The results are stored in a list and plotted as a simple curve of accuracy versus  $C$ . For very small  $C$ , the model enforces a wide margin and tolerates more misclassifications, which often shows up as underfitting and reduced accuracy. As  $C$  grows, accuracy improves until it reaches a plateau. If  $C$  becomes extremely large, the model may start to overfit, though this effect is often more pronounced on noisier data.

A similar analysis is carried out for the RBF kernel, this time varying gamma while keeping  $C$  fixed. When gamma is too small, the decision surface is overly smooth and the model fails to capture the curved shape of the moons or circles, leading to underfitting. As gamma increases, the boundary becomes more flexible and accuracy improves. Beyond a certain point, however, the boundary becomes excessively wiggly and starts to track individual training points, which typically reduces generalisation performance.

Together, these plots emphasise that  $C$  and gamma act as complementary complexity controls.  $C$  penalises misclassification versus margin width, while gamma controls the locality of the RBF kernel. Effective model selection requires balancing both parameters using cross-validation rather than relying on arbitrary default values.



## 7. Grid Search Results and Confusion Matrices

For each dataset and kernel family, GridSearchCV is run over the defined hyperparameter grid. The best estimator found by cross-validation is then evaluated on a held-out test set, and the resulting accuracy, confusion matrix, and full classification report are printed. These results are also collected into a pandas DataFrame, sorted by dataset and test accuracy, and written out as a CSV file so that they can be inspected programmatically.

On the blobs dataset, the best-performing models are typically simple: a linear kernel with a moderately chosen C usually achieves very high accuracy because the underlying structure is almost linearly separable. On the moons and circles datasets, the RBF and polynomial kernels tend to dominate, because they can express the curved and ring-shaped boundaries required for good performance. Exact numbers depend on the random seed and noise level, but the overall pattern is stable.

Confusion matrices offer an additional view of performance. They show how many examples of each class are correctly classified and where the model makes mistakes. For the poorly matched combination of dataset and kernel — for example, a linear kernel on the circles dataset — the confusion matrix reveals many misclassified points. For a well-matched combination such as RBF on circles, the matrix is almost perfectly diagonal.

## 8. Practical Guidelines and Overall Conclusions

The experiments support several practical guidelines for working with SVMs in real applications. First, feature standardisation is essential. Without scaling, SVMs — especially those with RBF or polynomial kernels — can behave erratically because the kernel computations depend on absolute distances between points. The notebook demonstrates how to integrate scaling into a Pipeline so that it becomes a standard part of the modelling workflow.

Second, kernel choice should be informed by both domain knowledge and exploratory analysis. For data that are roughly linearly separable or live in a very high-dimensional sparse space, a linear kernel is usually the best starting point. For low-dimensional data with obvious non-linear structure, an RBF kernel is a strong default, provided that gamma and C are tuned with cross-validation. Polynomial kernels are most useful when there is reason to believe in polynomial interactions or when they empirically outperform RBF on validation data; in practice, low degrees often work best.

Third, the notebook highlights the value of visualisation. Decision-boundary plots and support-vector markers turn abstract optimisation into something concrete and inspectable. When combined with numerical metrics such as accuracy and confusion matrices, these visuals make it easier to diagnose underfitting, overfitting, and mismatches between kernel choice and data geometry.

**BELOW IS THE REPO LINK :**

<https://github.com/srihari4420/svm-kernel-comparison-tutorial>