Toggle code

Week 8 - Support Vector Machines

Dr. David Elliott

Common Notation

- $\mathbf{X}$ is a matrix containing all the feature values of all the observations
- n is the number of observations in the dataset
- $\mathbf{x}_i$ is a vector of all the feature values (except the label) of the ith instance in the dataset.
- y_i is the label (desired model output) of the ith instance in the dataset.
- p is the number of features in the dataset
- $\mathbf{x}_j$ is a vector of all the observations values of the jth feature in the dataset.

Notes

We can't always perfectly separate the data with a $p-1$ dimensional hyperplane. To overcome this problem we could either:

- Tweak the constraints on the hyperplane to allow some points to be misclassified (*soft margin*),
- Transform the data to be separable by a hyperplane in another space (*kernel method*).

1.4. Support Vector Classifier (SVC)

SVC's are a generalisation and extension of the maximal margin classifier so it can be applied to a broader range of cases[1].

In practice they are more robust to individual observations and better classify most training observations than the Maximal Margin Classifier. This is because they take the approach it is better to missclassify some training examples in order to do a better job classifying the rest.

This is called a *soft margin* as it allows some violations by the training data by a small subset of training observation, not only on the wrong side of the margin, but wrong side of the hyperplane.

Notes

- *"Developed in the computer science community in the 1990s"*[2]

Figure 16: No Exact Linear Separating Hyperplane

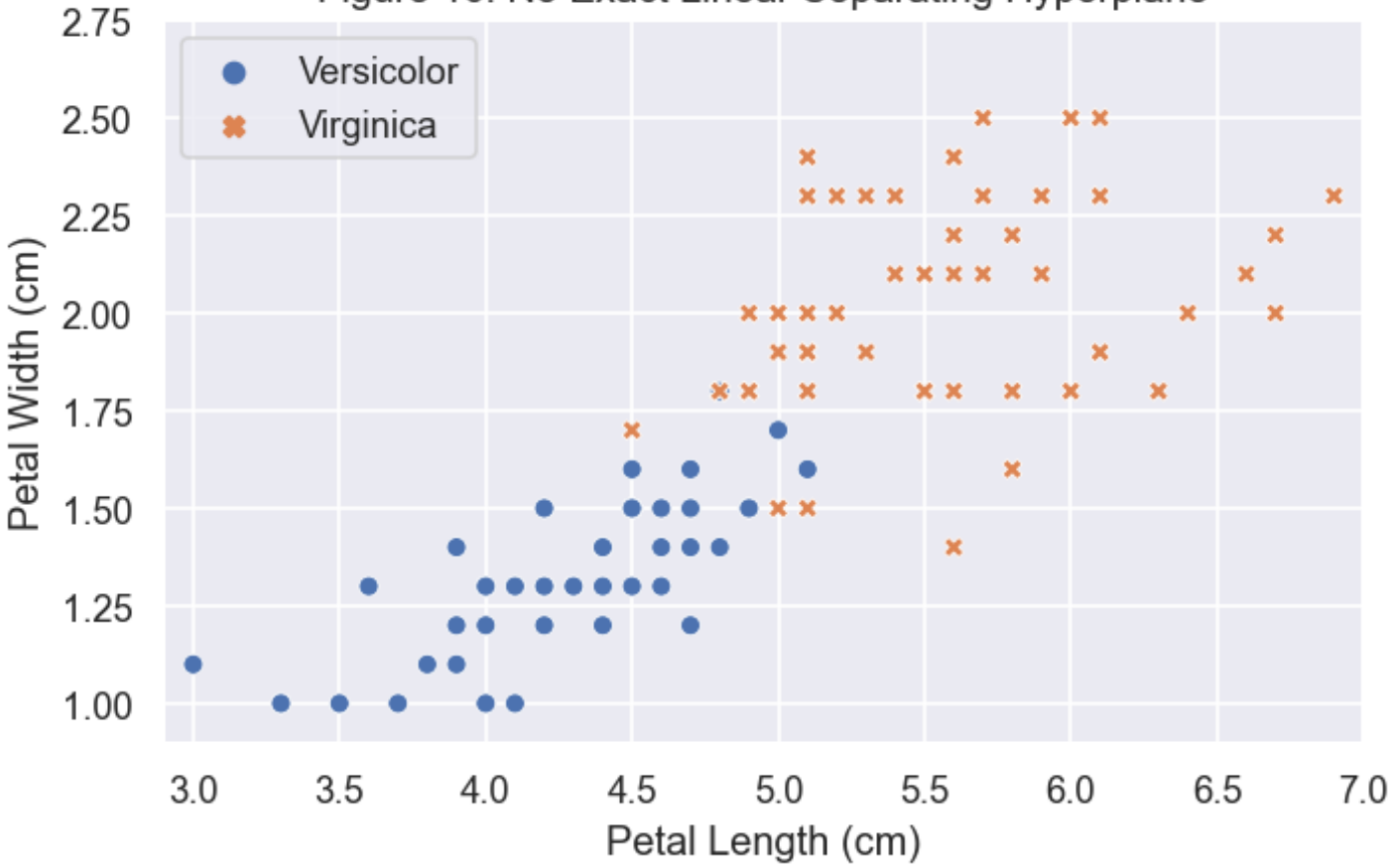
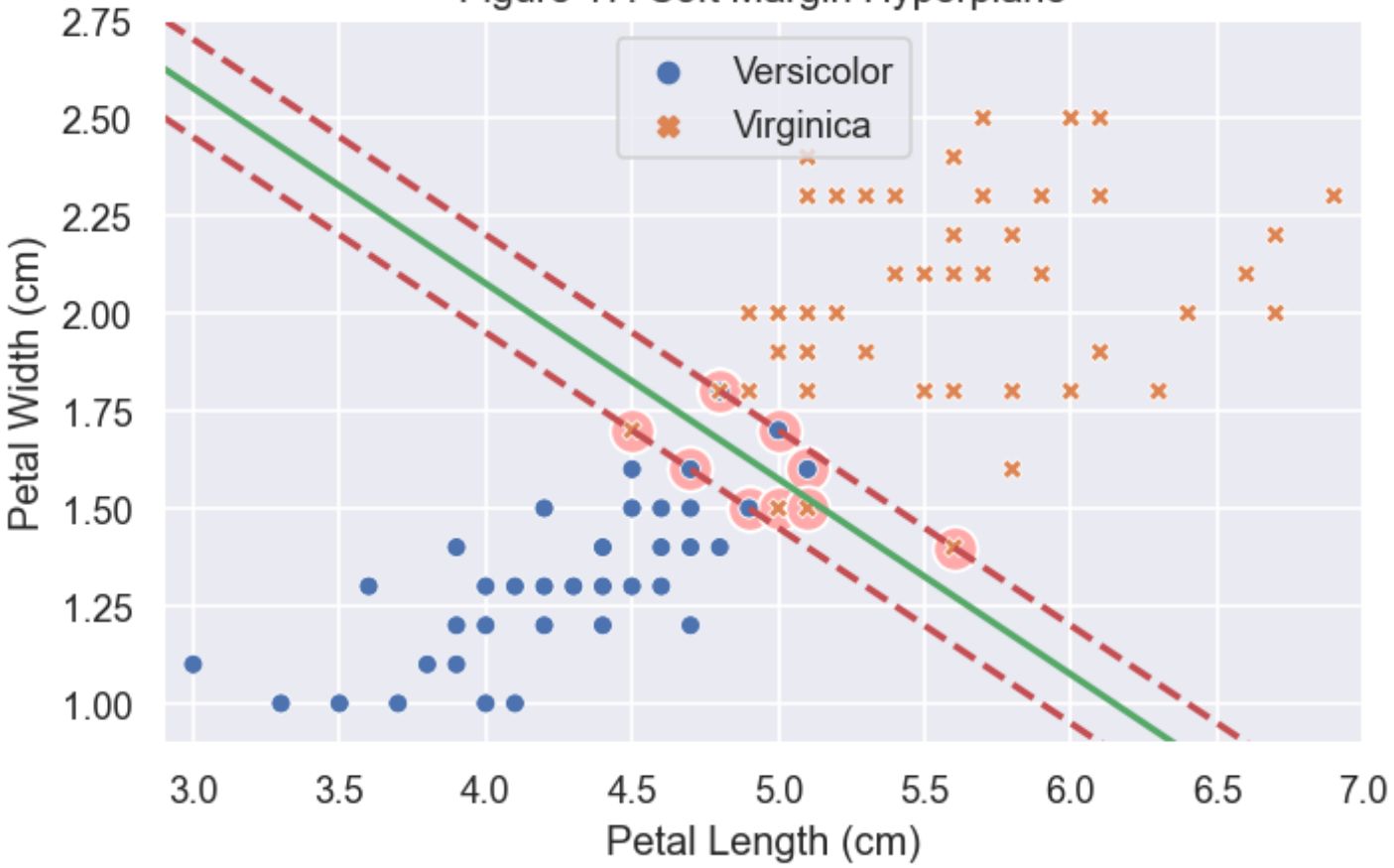


Figure 17: Soft Margin Hyperplane



We want to relax the following constraints when necessary:

$$\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b \geq 1 \text{ for } y_i = 1, \mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b \leq -1 \text{ for } y_i = -1$$

This can be done by introducing positive slack variables $\xi_i, i = 1, \ldots, n$ in the constraints[5,6,10]:

$$\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b \geq 1 - \xi_i \quad \text{if} \quad y_i = 1, \mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b \leq -1 + \xi_i \quad \text{if} \quad y_i = -1, \xi_i \geq 0 \quad \forall_i.$$

Notes

- Slack variable $\xi_1, \ldots, \xi_n$ allow individual observations to be on the wrong side of the margin or hyperplane.
- $\sum_i \xi_i$ is an upper bound on the number of training errors.
- ξ_i tells us where the ith observation is located relative to the hyperplane; $\xi_i = 0$ being on the correct side of the margin, $\xi_i > 0$ being on the wrong side of the margin, and $\xi_i > 1$ on the wrong side of the hyperplane.
- $\xi_1 = \ldots = \xi_n = 0$ is the maximal margin hyperplane optimisation.
- Test observations are classified as before, $f(x^*) = \beta_0 + \beta_1 x_1^* + \ldots + \beta_p x_p^*$.

Tuning Parameter (C)

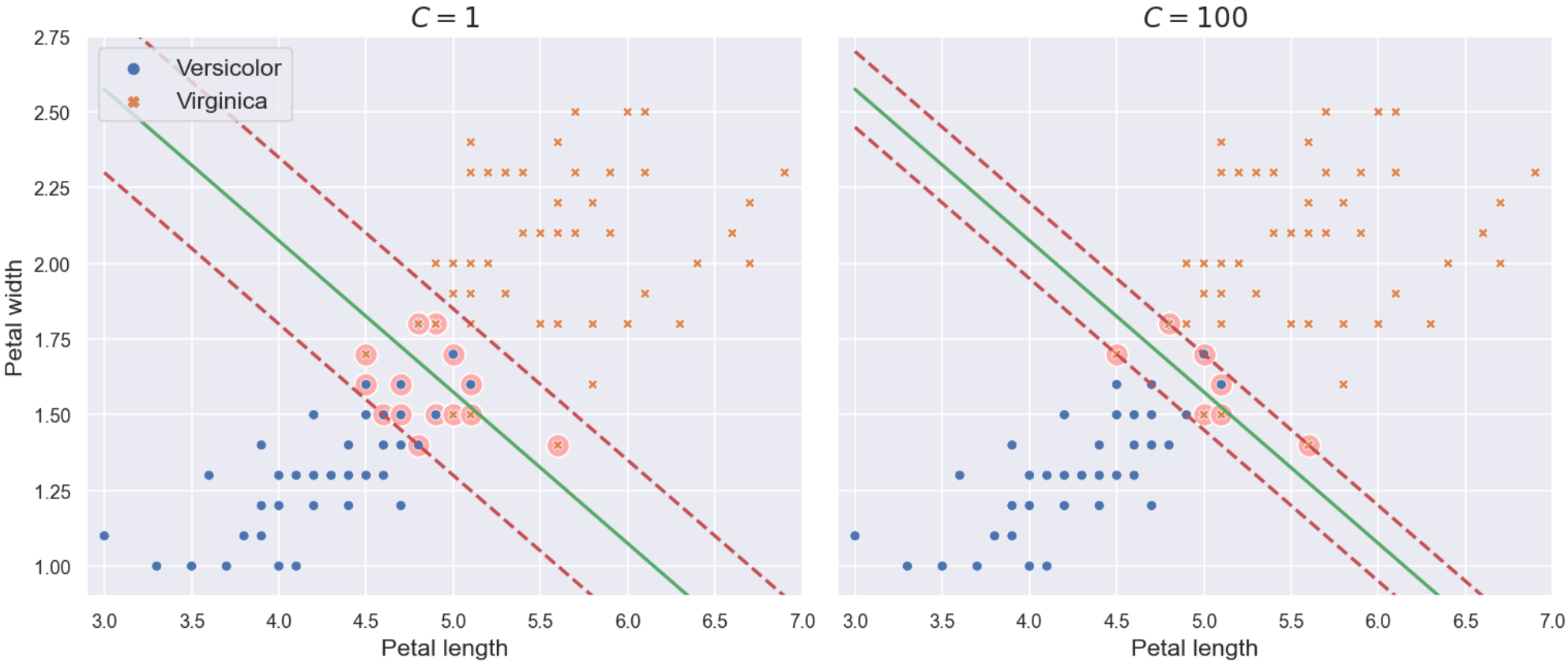
To ensure there is a penelty, C, for relaxing the constraint, we can change our objective function to be minimised from $\frac{1}{2}||\mathbf{w}||^2$ to,

$$\begin{aligned} &\underset{\mathbf{w}, b, \xi}{\text{minimise}} && \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n} \xi_i, \\ &\text{subject to} && y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall_i. \end{aligned}$$

C is a tuning parameter that controls the bias-variance trade-off[1].

The strength of the regularization is inversely proportional to C, meaning a large C has a larger error penalty.

Figure 18: Tuning Parameter `C`



Extra

Neither the ξ_i or their Lagrange multipliers appear in the Wolfe dual problem. This means we now have[6]:

$$\max L_D \equiv \sum_i^n \alpha_i - \frac{1}{2}\sum_{i,k}^n \alpha_i \alpha_k y_i y_k \mathbf{x}_i \cdot \mathbf{x}_k \quad \text{s.t.} \quad 0 \leq \alpha_i \leq C, \quad \sum_i \alpha_i y_i = 0.$$

This also has the same solution as before:

$$\hat{\mathbf{w}} = \sum_{i=1}^{N_S} \alpha_i y_i \mathbf{x}_i.$$

Also, sometimes C is defined as $C = \frac{1}{vN}$, where $0 < v \leq 1$ controls the fraction of misclasified points during the training phase[7].

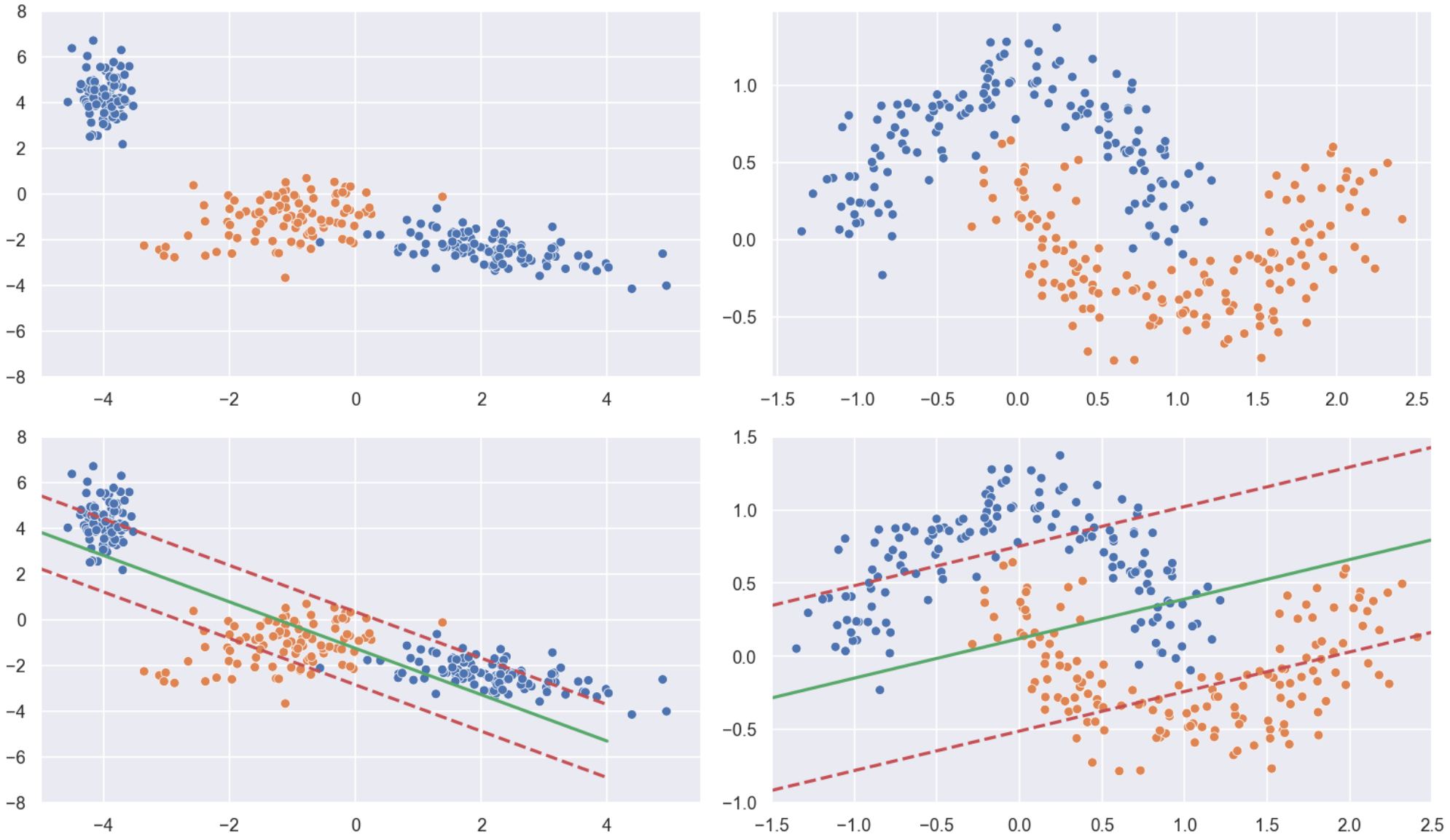
Notes

- Alike to maximal margin classifiers, SVC's only rely on a few observations, those on the margin or those that violate the margin (*Support Vectors*).
  - If they are on the correct side of the margin they dont change the classifier.
  - This does mean that they are robust to observations far away from the hyperplane.
- When C is large we have narrow margins rarely violated, but highly fit to the training data (low bias-high variance).
- Coversely, when smaller it is less strict about missclassification errors, meaning the margin is wider (high bias-low variance).
- Like most hyper-parameters, it is often chosen using cross-validation.

1.5. Support Vector Machine (SVM)

Aims to address the situation where the boundary between two classes is not linear.

Figure 19: Linear SVM on Non-Linear Data

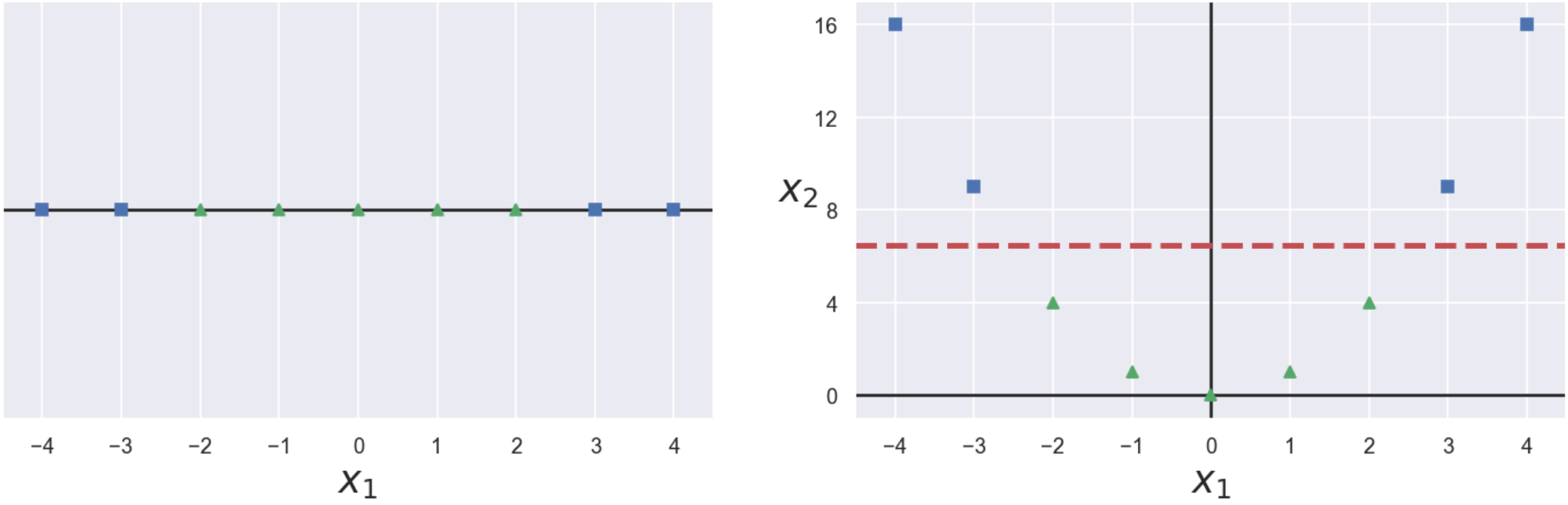


Feature Engineering

We could consider enlarging the feature space to make the dataset linearly separable.

Example: We can see below that our $\mathbf{x}_1$ is not linearly separable but it is when we add in our second feature $\mathbf{x}_2 = (\mathbf{x}_1)^2$

Figure 20: Adding Features to Linearly Separate Data

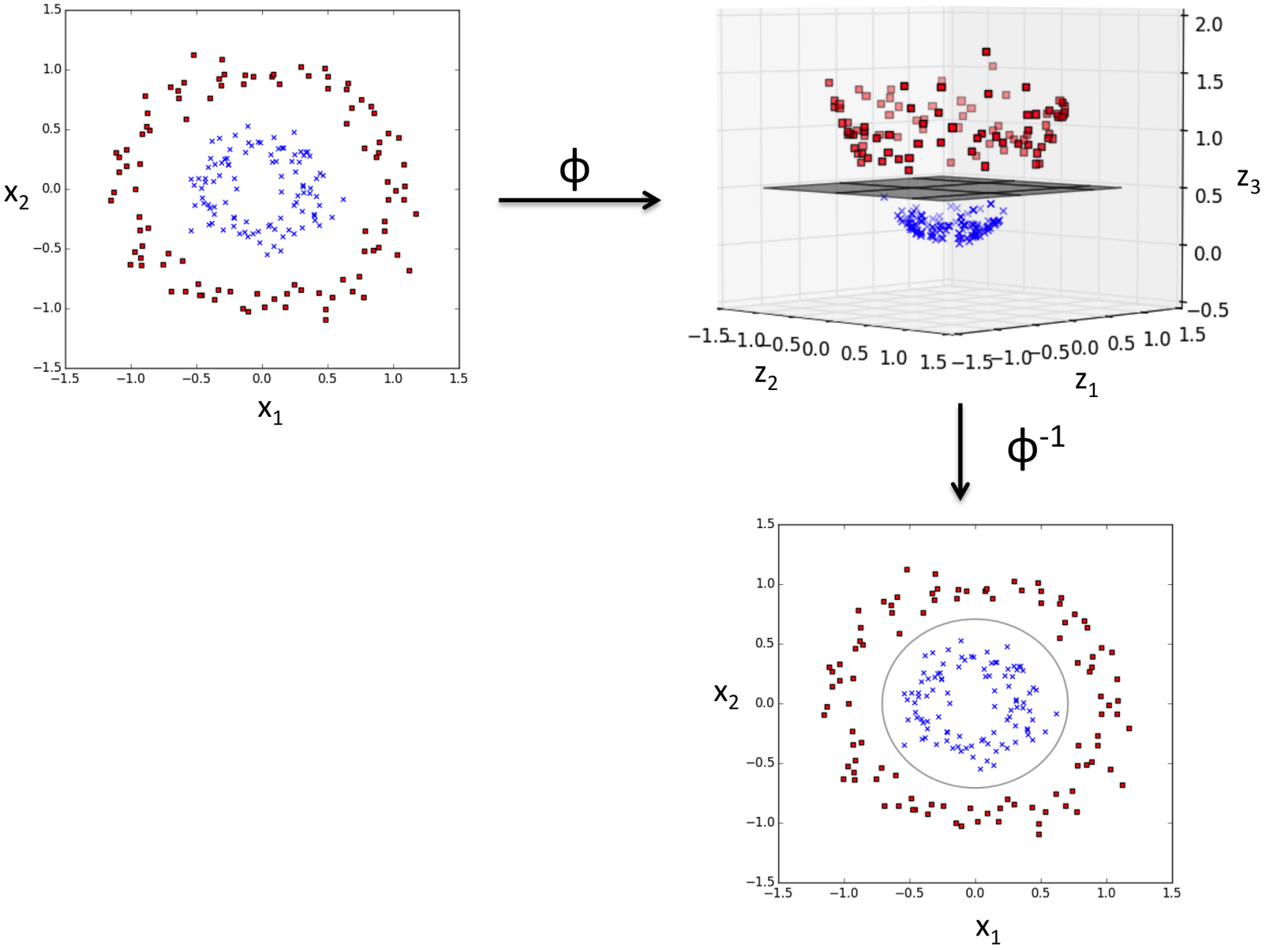


Using quadratic, cubic or higher-order polynomial functions we can project our data onto a higher-dimensional space via a mapping function ϕ where they are linearly separable (using a linear SVM model in this new feature space).

Example

$$\phi(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3) = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_1^2 + \mathbf{x}_2^2)$$

Figure 21: 2D Dataset into Separable 3D Feature Space



Gaussian Radial Basis Function[2]

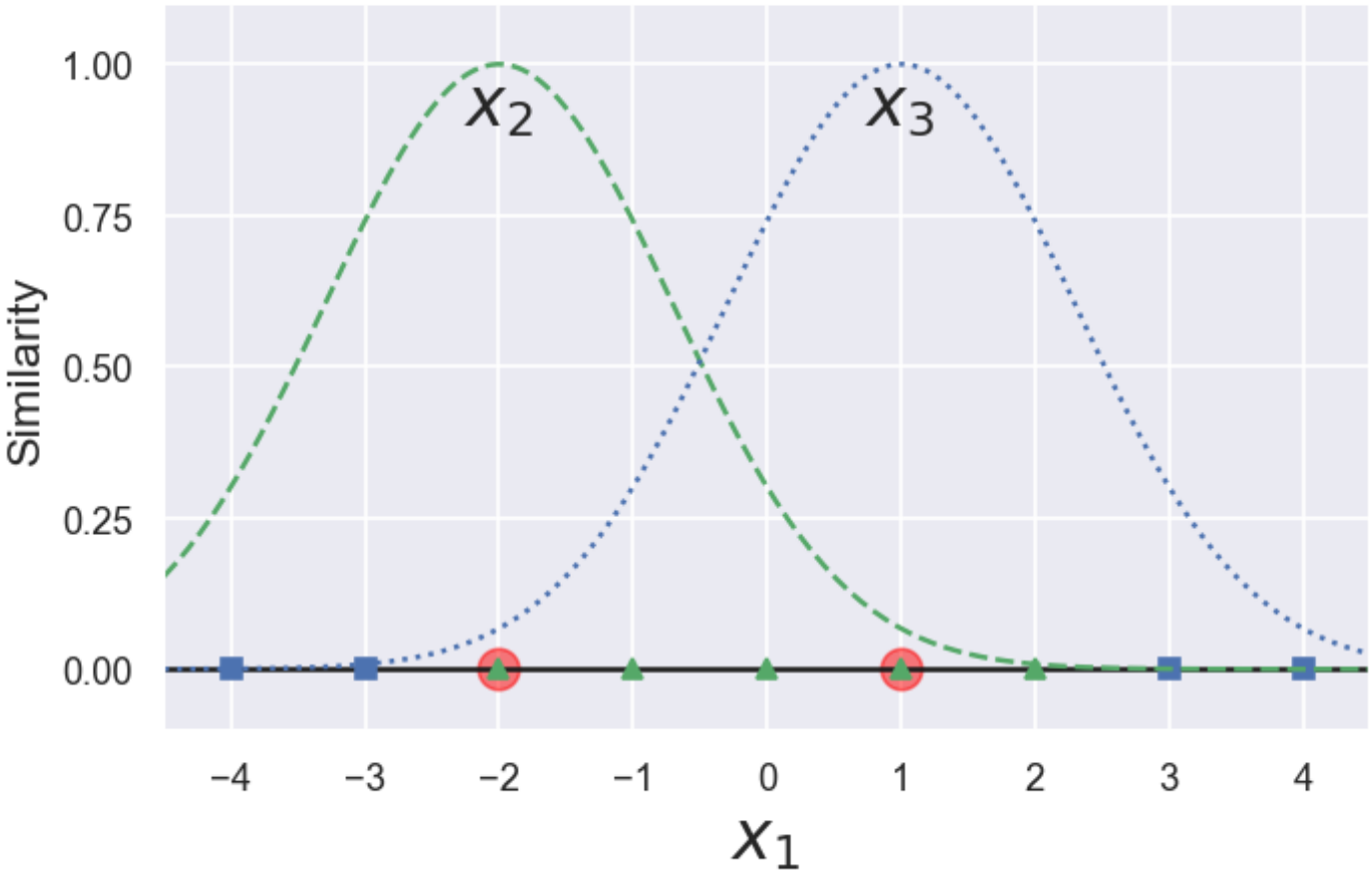
We could instead use a *"similarity function"*, such as a Gaussian Radial Basis Function (RBF),

$$\phi_\gamma(\mathbf{x}, \ell) = \exp(-\gamma||\mathbf{x} - \ell||^2).$$

This is a bell-shaped function which measures how much an instance resembles a *landmark*, with the function varying from 0 (far away) to 1 (at the landmark).

Example: Below we set our landmarks to $x_1 = -2$ and $x_1 = 1$.

Figure 22: RBF Landmarks



Using the example of $x_1 = -1$ we can see it is a distance of 1 from the first landmark and 2 from the second.

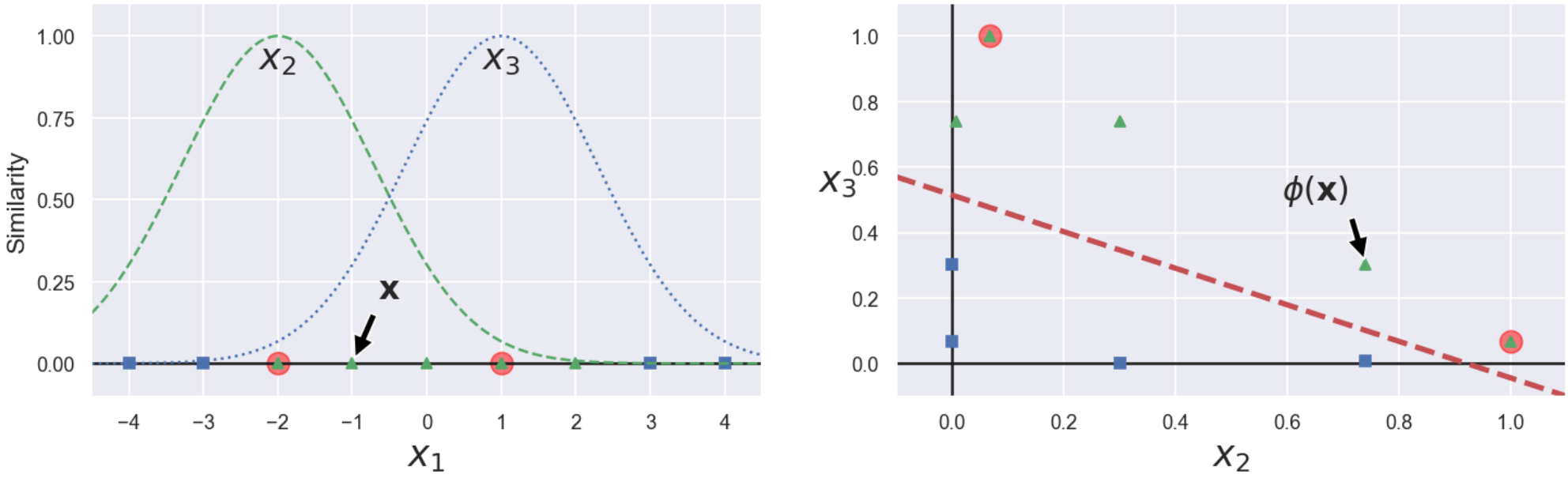
If we set $\gamma = 0.3$ then our new features are:

$x_2 = \exp(-0.3 \times 1^2) \approx 0.74$

$x_3 = \exp(-0.3 \times 2^2) \approx 0.30$

In order to find the landmarks the simplist approach is just to create a landmark at each instance in the dataset.

Figure 23: RBF New Feature Space



Kernels

However, by using feature engineering to enlarge our feature space, the larger the number of features, the higher computational burden.

Instead it is common to enlarge the feature space using an extension of a SVC termed a Support Vector Machine, which uses *kernels*.

The Kernel trick can relies on the fact we can define our SVM in the form of inner products.

$$L_D(\alpha_i) = \sum_i^n \alpha_i - \frac{1}{2}\sum_{i,k}^n \alpha_i \alpha_k y_i y_k \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_k \qquad \text{s.t.} \quad \forall_i \alpha_i \geq 0, \ \sum_i^n \alpha_i y_i = 0.$$

Imagine we had a mapping ϕ which maps the data to some high dimensional Euclidean space

$$\phi : \mathrm{R}^d \mapsto H,$$

then, we could do dot products between vectors after the mapping in H:

$$L_D(\alpha_i) = \sum_i^n \alpha_i - \frac{1}{2}\sum_{i,k}^n \alpha_i \alpha_k y_i y_k \phi(\mathbf{x}_i^{\mathrm{T}})\phi(\mathbf{x}_k)$$

Instead we could use a kernel function,

$$K(\mathbf{x}_i, \mathbf{x}_k) = \phi(\mathbf{x}_i^{\mathrm{T}})\Phi(\mathbf{x}_k).$$

Notes

- Roughly speaking, a kernel can be interpreted as a similarity function betwen pairs of samples[4].
- $\mathbf{a} \in \mathrm{R}^k$ just means it is a vector of length k.
- $\mathbf{A} \in \mathrm{R}^{r \times s}$ just means it is an $r \times s$ matrix.
- $\mathbf{w}$ lives in H ("high dimensional")
- H is often refered to as a Hilbert space.
- *"it is clear that the above implicit mapping trick will work for any algorithm in which the data only appear as dot products (for example, the nearest neighbor algorithm). This fact has been used to derive a nonlinear version of principal component analysis by (Scholkopf, Smola and Muller, 1998b); it seems likely that this trick will continue to find uses elsewhere."*[6]

Second-Degree Polynomial Example[2]

Suppose we wanted to apply a second-degree polynomial transformation to two 2D vectors, $\mathbf{a}$ and $\mathbf{b}$, and then compute the dot product. We could do this by:

$$\phi(\mathbf{a})^{\mathrm{T}}\phi(\mathbf{b}) = \begin{pmatrix} {a_1}^2 \\ \sqrt{2a_1a_2} \\ {a_2}^2 \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} {b_1}^2 \\ \sqrt{2b_1b_2} \\ {b_2}^2 \end{pmatrix} = {a_1}^2{b_1}^2 + 2a_1b_1a_2b_2 + {a_2}^2{b_2}^2.$$

Instead we could use the kernel approach:

$$K(\mathbf{a}, \mathbf{b}) = (\mathbf{a}^{\mathrm{T}}\mathbf{b})^2 = \left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \right)^2 = (a_1 b_1 + a_2 b_2)^2 = {a_1}^2 {b_1}^2 + 2 a_1 b_1 a_2 b_2 + {a_2}^2 {b_2}^2.$$

This is useful as it means we didnt have to map our data using ϕ first. This saves us time!

Notes

- A second-degree polynomial for a training set with two features, $\mathbf{x} \in \mathrm{R}^2$, would be:

$$\Phi(\mathbf{x}) = \Phi((\mathbf{x}_1, \mathbf{x}_2)) = ({\mathbf{x}_1}^2, \sqrt{2\mathbf{x}_1\mathbf{x}_2}, {\mathbf{x}_2}^2)$$

Polynomial Feature Engineering (degree=2)

```
2.11 ms ± 42.3 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

Polynomial Kernel (degree=2)

```
3.04 ms ± 57.4 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

We still have all the same considerations, but replacing $\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_k$ with $K(\mathbf{x}_i, \mathbf{x}_k)$ allows us to produce a SVM in infinite dimensional space.

$$\sum_i^n \alpha_i - \frac{1}{2}\sum_i^n \sum_k^n \alpha_i \alpha_k y_i y_k K(\mathbf{x}_i, \mathbf{x}_k) \qquad \text{s.t.} \quad \forall_i \alpha_i \geq 0, \ \sum_i^n \alpha_i y_i = 0.$$

In the test phase, we can use the support vectors[6]:

$$\begin{aligned} f(\mathbf{x}^*) &= \sum_{i \in s} \hat{\alpha}_i y_i \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}^*) + \hat{b} \\ &= \sum_{i \in s} \hat{\alpha}_i y_i K(\mathbf{x}_i, \mathbf{x}^*) + \hat{b}, \end{aligned}$$

avoiding computing $\Phi(\mathbf{x}^*)$.

Note

- Put another way, in the test phase[6], we can use the support vectors, $\mathbf{s}_i$: $$ \begin{align} f(\mathbf{x}^*) &= \sum_{i = 1}^{n_s}\hat\alpha_i y_i\Phi(\mathbf{s}_i)\cdot \Phi(\mathbf{x}^*) + \hat b \\ &= \sum_{i = 1}^{n_s}\hat\alpha_i y_iK(\mathbf{s}_i,\mathbf{x}^*) +\hat b.

\end{align} $$

Polynomial kernels

The polynomial kernel of degree d (a positive integer) can be defined as:

$$K(\mathbf{x}_i, \mathbf{x}_k) = \left(\gamma \left\langle \mathbf{x}_i, \mathbf{x}_k \right\rangle + r\right)^d$$

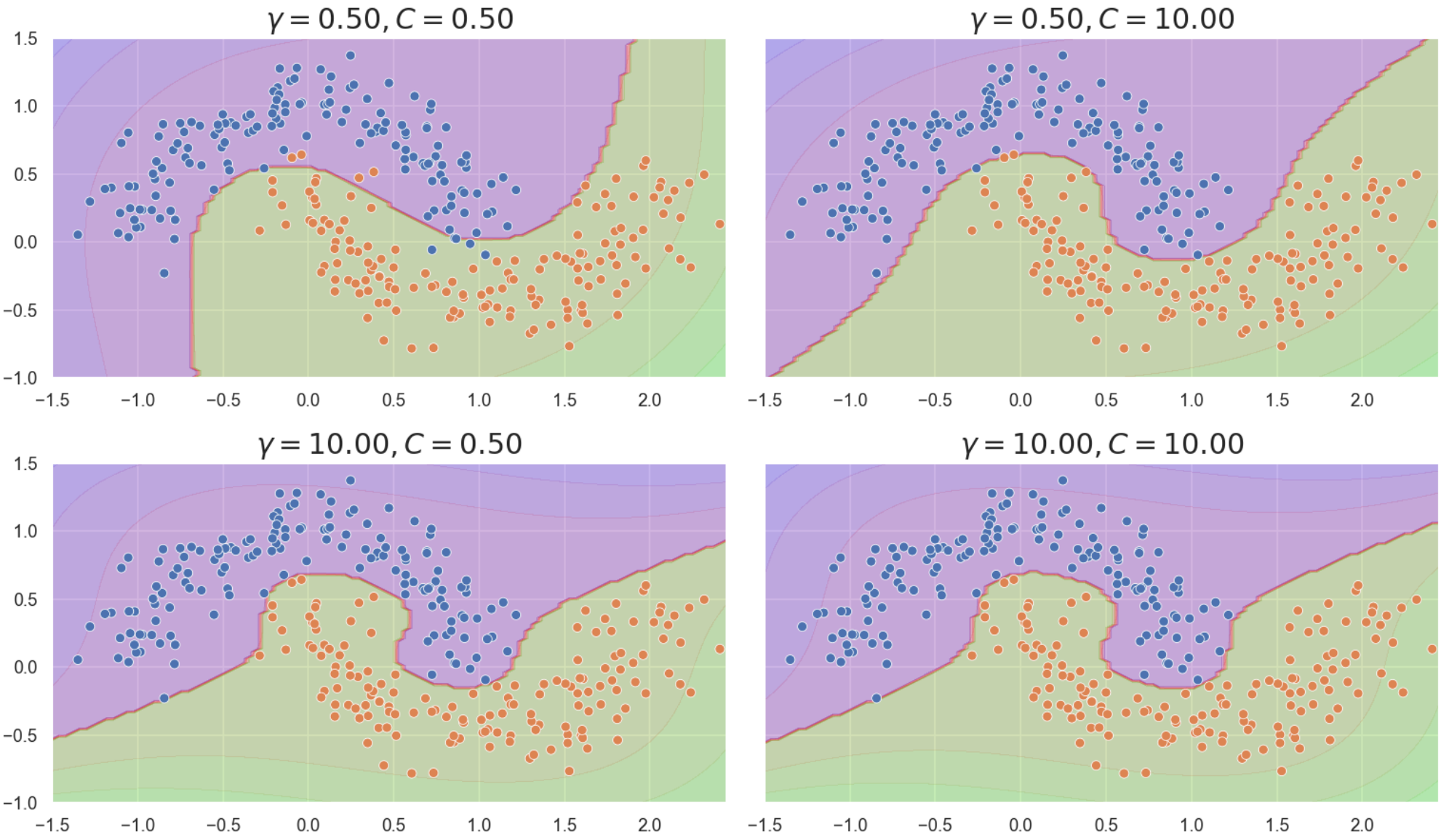
Hyperparameters

- γ (`gamma`) broadly defines the width or slope of kernel function[13].
- r (`coef0`) roughly controls how much the model is influenced by high-degree polynomials[2].
- d (`degree`) is the degree of the polynomial.

Notes

- You will sometimes see a kernel defined in a general sense using $K(\mathbf{x}, \mathbf{x}')$, where $\mathbf{x}'$ means another point in the dataset.
- $\left\langle x_i, x_j \right\rangle = x_i \cdot x_j = \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_k$.
- γ is often 1.
- *"The 'curve' of the decision boundary becomes very low when gamma value is low making the decision region very broad. The 'curve' of the decision boundary becomes high when gamma is high, which creates islands of decision-boundaries around data points."*[13]
- d is a positive integer.
- If you start to overfit you should reduce the polynomial degree and underfitting try increasing it.

Figure 24: Polynomial Kernel Decision Boundaries (degree=3, coef0=1)



You shouldnt stick to the default settings, instead you want to search for optimal hyperparameters for the data. Good suggestions for search spaces are:

- *"The authors of libsvm (Hsu et al. 2009) recommend using cv over a 2d grid with values $C \in \{2^{-5}, 2^{-3} ..., 2^{15}\}$ and $\gamma \in \{2^{-15}, 2^{-13}, ..., 2^{3}\}$"*[4].

	param_svm_clf__C	param_svm_clf__gamma	param_svm_clf__degree	param_svm_clf__coef0	mean_test_score	std_test_score
2829	32768	8	3	2	0.996667	0.006667
957	2	0.5	4	8	0.996667	0.006667
1796	128	0.125	4	8	0.996667	0.006667
1149	8	8	3	2	0.996667	0.006667
757	0.5	0.5	4	32	0.996667	0.006667

Radial Basis Function kernel

The most widely used kernel is the RBF kernel (also known as a Gaussian kernel)[4,8]:

$$K(\mathbf{x}_i, \mathbf{x}_k) = \exp\Big(- \gamma | \, | \mathbf{x}_i - \mathbf{x}_k | \, |^2 \Big),$$

where γ is a free parameter to be optimised and $| \, | \mathbf{x}_i - \mathbf{x}_k | \, |^2$ is the squared *Euclidean distance*.

γ is often either a positive constant or $\frac{1}{2\sigma^2}$.

When classifying a test observation $\mathbf{x}^* = (x_1^* \ldots x_p^*)$, only training observations close to $\mathbf{x}^*$ (in terms of Euclidean distance) will play a role in its class label. This is because $(x_j^* - x_{ij})^2$ will be large, so $\exp(- \gamma \sum_{j=1}^{P} (x_j^* - x_{ij})^2)$ will be small[1].

Notes

Euclidean distance[8]

$$d(\mathbf{x}_i, \mathbf{x}_k) \overset{\text{def}}{=} \sqrt{\big(x_{i1} - x_{k1}\big)^2 + \big(x_{i2} - x_{k2}\big)^2 + \ldots + \big(x_{iN} - x_{kN}\big)^2} = \sqrt{\sum_{j=1}^{D} \big(x_{ij} - x_{kj}\big)^2}$$

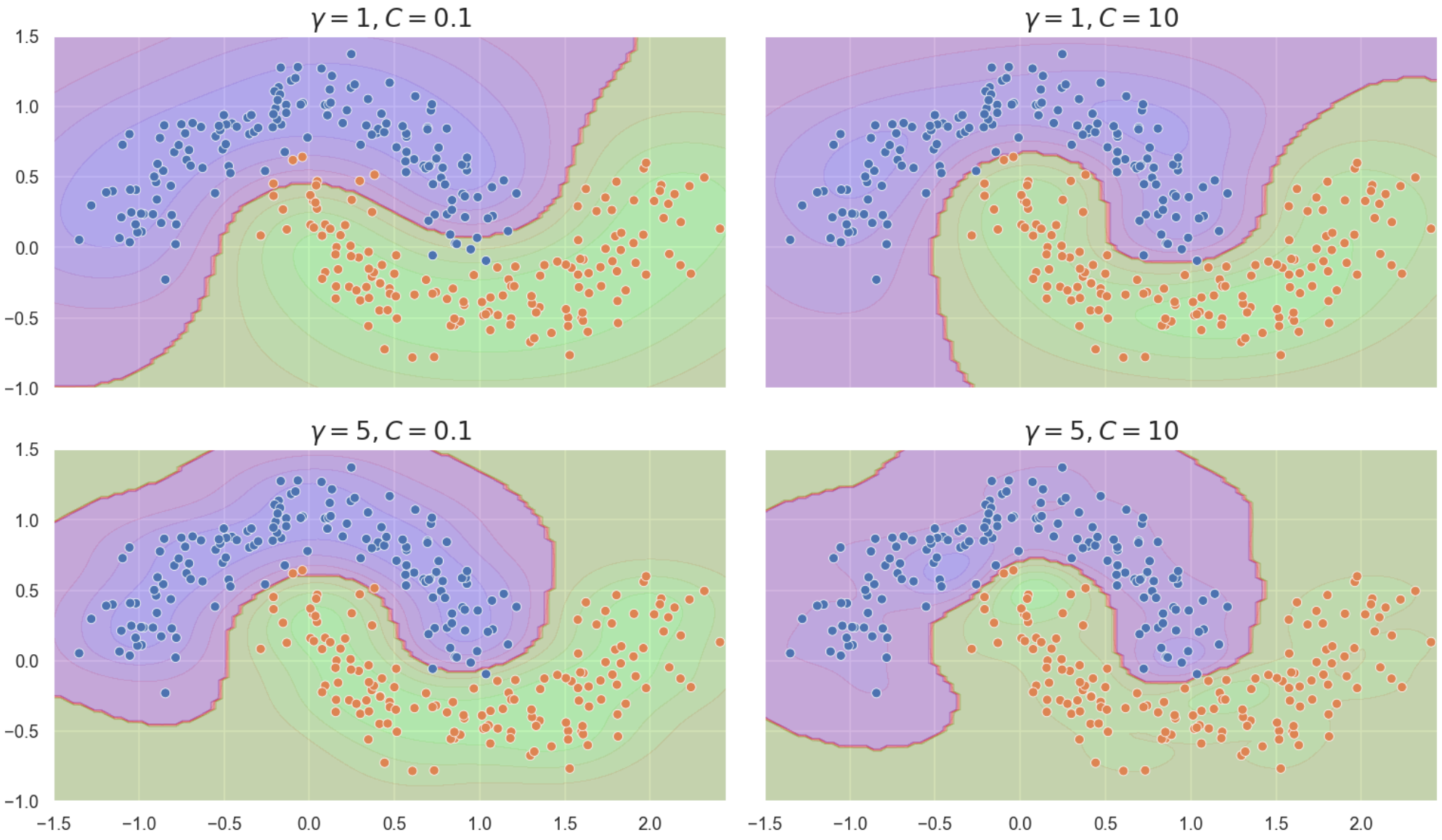
- _"if `gamma='scale'` (default) is passed then it uses 1 / (n*features * X.var()) as value of gamma"*[12]
- the minus sign inverts the the distance measure to a similarity score[10]

Hyperparameters[2]

γ is effectively acting like a regularization hyperparameter, so like C if your model is overfitting reduce it and underfitting then increase it.

- Increasing γ (`gamma`) makes the bell-shaped curve narrower.
  - Each instances range of influence is smaller.
  - The decision boundary becomes more irregular.
- Decreasing γ makes the bell-shaped curve wider.
  - Instances have a larger range of influence.
  - The decision boundary becomes smoother decision.

Figure 25: Gamma Kernel Decision Boundaries



param_svm_clf__C	param_svm_clf__gamma	mean_test_score	std_test_score

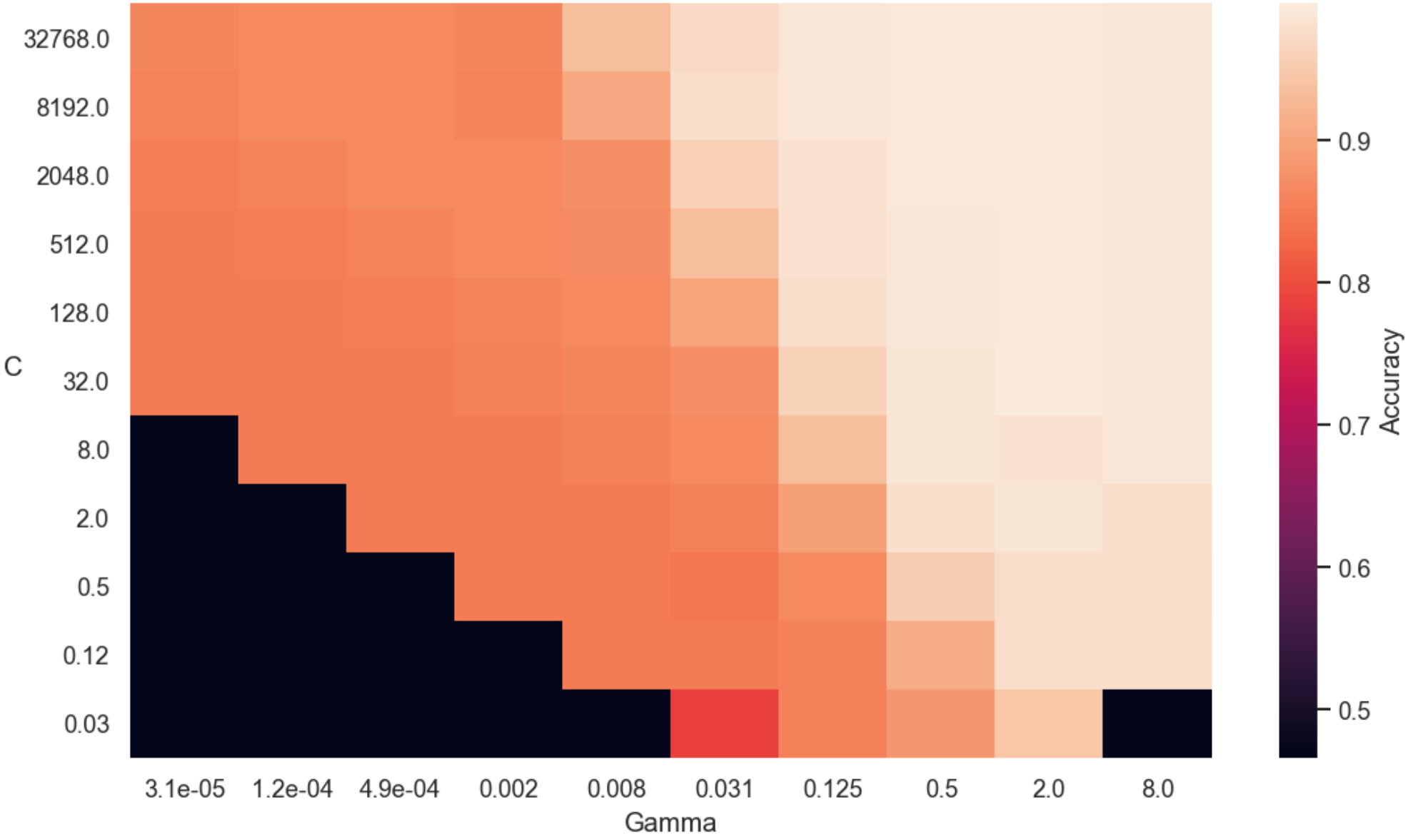
	param_svm_clf__C	param_svm_clf__gamma	mean_test_score	std_test_score
58	32	2	0.996667	0.006667
78	512	2	0.993333	0.008165
107	32768	0.5	0.993333	0.008165
87	2048	0.5	0.993333	0.008165
88	2048	2	0.993333	0.008165

C and γ are tightly coupled.

Generally if you have a larger/narrow `gamma` (e.g. $\gamma = 5$) you'll need more regularisation, so a larger C.

If you have a smaller/wider `gamma` (e.g. $\gamma = 1$), a smaller value of C should be used[7].

Figure 26: GridSearch Results for Gamma and C on the Moons Data



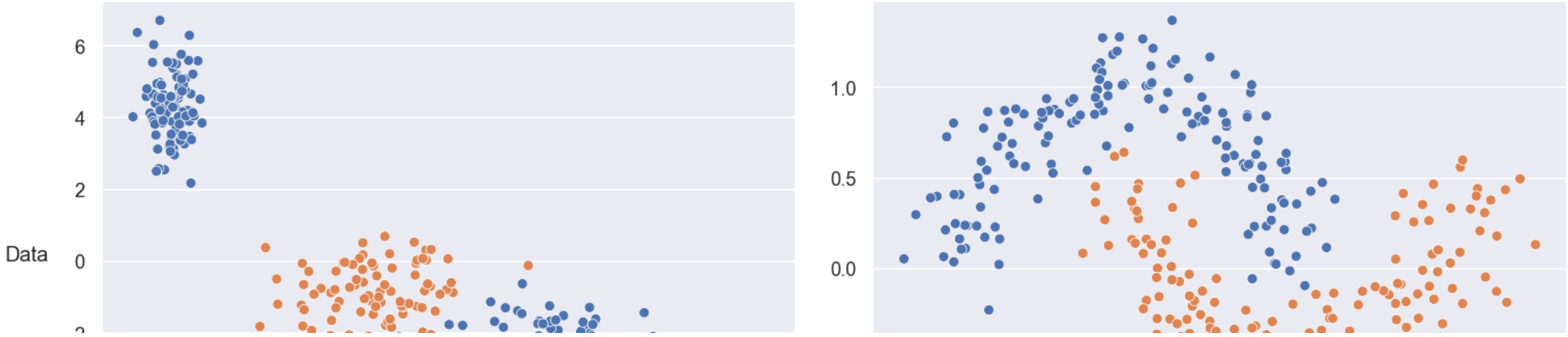
Extra Example

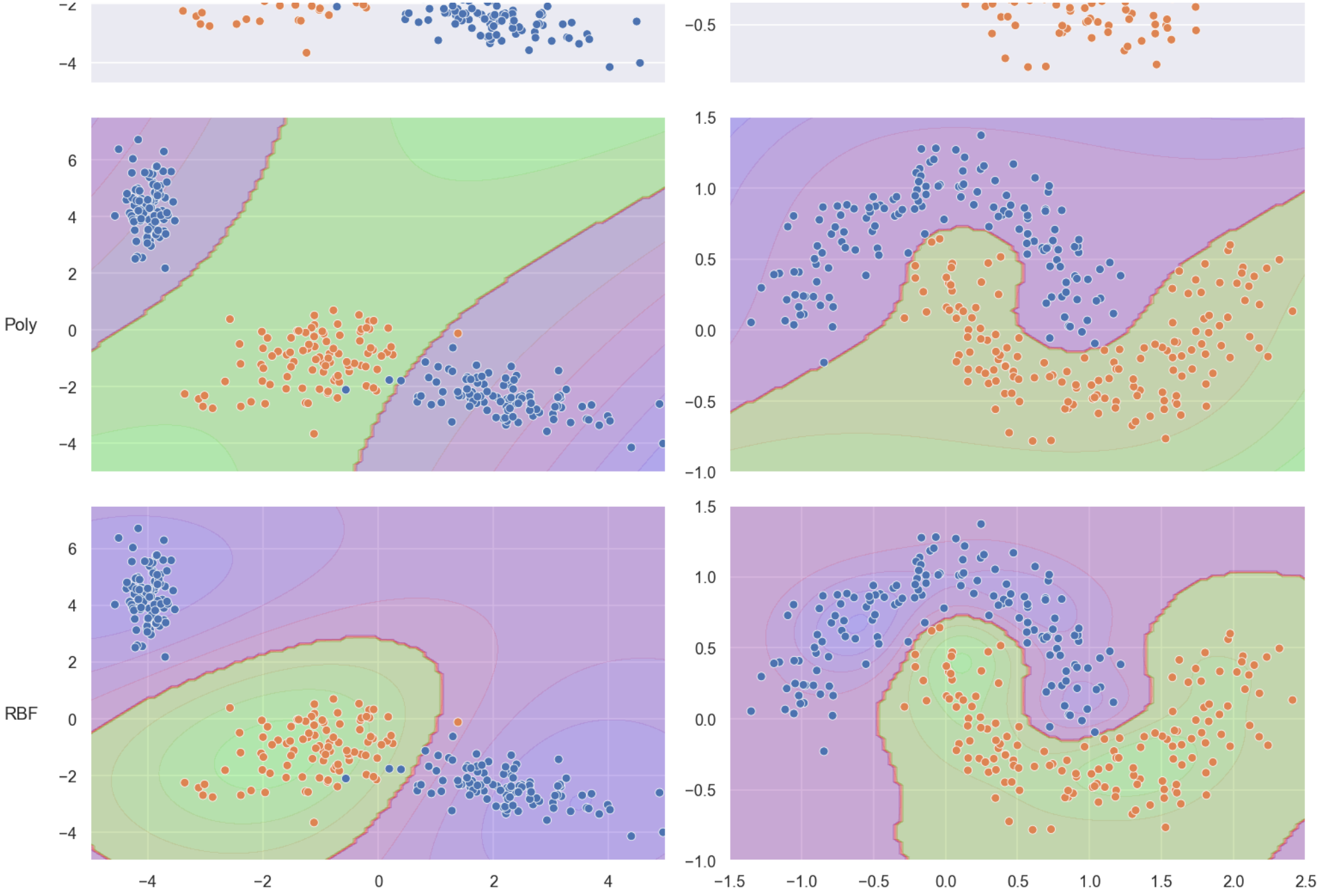
Lets have a look at the decision boundaries for our "optimal hyperparameters" as a comparison to those by the linear models we saw earlier.

	param_svm_clf__C	param_svm_clf__gamma	param_svm_clf__degree	param_svm_clf__coef0	mean_test_score	std_test_score
387	0.125	0.5	3	8	0.983333	0.014907
514	0.125	0.007812	4	64	0.983333	0.014907
1545	32	0.03125	3	16	0.983333	0.014907
906	2	0.125	3	4	0.983333	0.014907
506	0.125	0.125	3	64	0.983333	0.014907

	param_svm_clf__C	param_svm_clf__gamma	mean_test_score	std_test_score
29	0.5	8	0.983333	0.014907
27	0.5	0.5	0.983333	0.014907
28	0.5	2	0.983333	0.014907
37	2	0.5	0.983333	0.014907
18	0.125	2	0.983333	0.014907

Figure 27: Best Non-Linear SVM Model Decision Boundary Using GridSearch



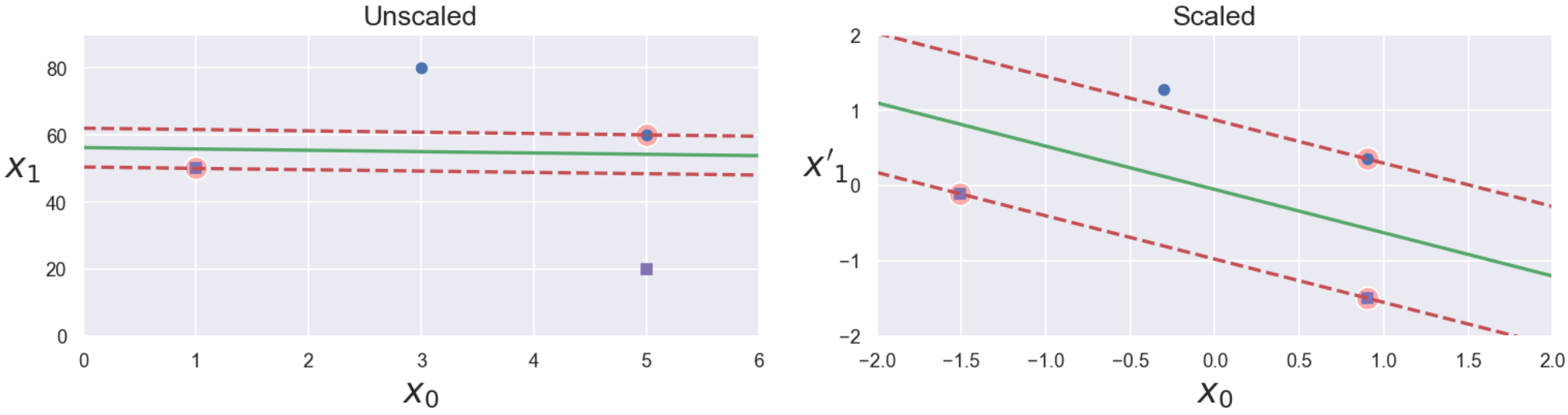


Standardization[11]

SVMs, along with many other machine learning estimators (e.g. l1 and l2 regularizers of linear models), are sensitive to feature scales.

If a feature has a variance orders of magnitude larger than others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected.

Figure 28: Sensitivity to Feature Scales



Notes

- We can see that in the unscaled plot that $\mathbf{x}_1$ is much larger than $\mathbf{x}_0$ so the widest street is very close to the horizontal.
- After scaling we have a much better decision boundary.

Standardization is especially important for RBF kernels.

These models assume that all features look like standard normally distributed data: Gaussian with zero mean and unit variance.

Dataset Example: Breast Cancer

We can see below for example that the features in the Breast Cancer dataset are of completely different orders of magnitude[16].

Figure 29: Feature Maginitudes for the Breast Cancer Dataset

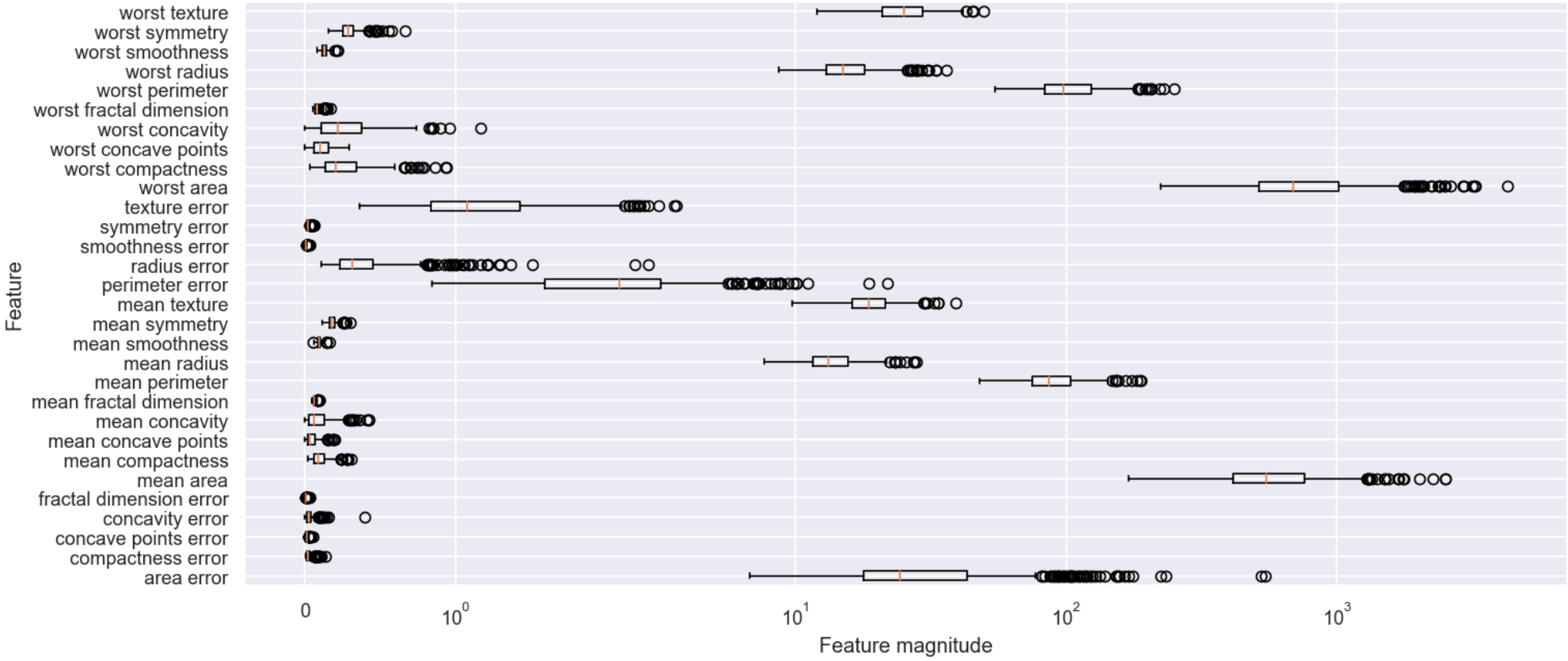
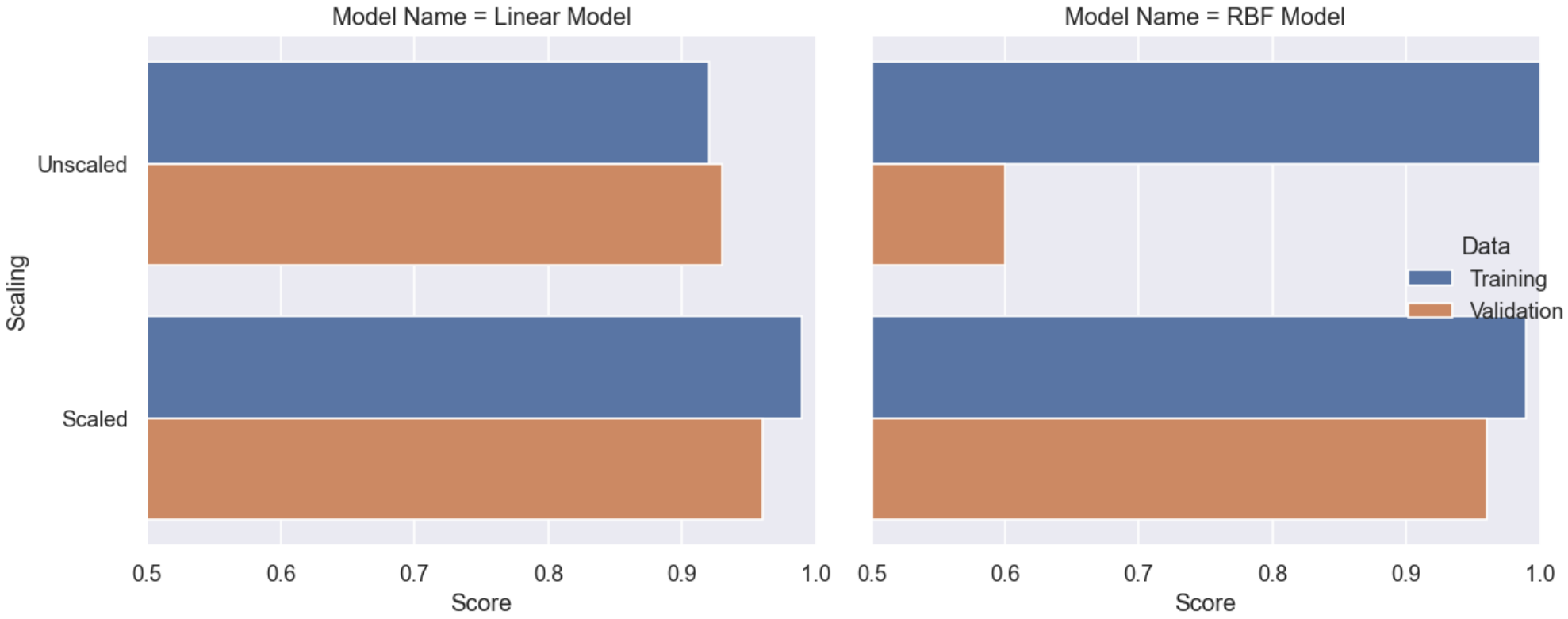


Figure 30: Effects of Scaling on SVM Overfitting



Notes

- In this example, the RBF model overfits quite substantially, with a perfect score on the training set and only 60% accuracy on the test set.
- Also you see the linear model also has a minor improvement in accuracy if scaled.

Recap[1]

The kernel approach is an efficient computational approach to enlarge our feature space to accommodate a non-linear boundary.

Assume we have a new point x^*. If wanted to compute $f(x^*)$ using our linear classifier we would need to the inner product between x^* and each training point x_i:

$$f(x) = \sum_{i \in s} \alpha_i \left\langle x^*, x_i \right\rangle + b.$$

Instead of actually calculating the inner product, we could instead use a *generalisation*, $K(x, x_{i'})$, where K is a *kernel*. We can now define the classifier as:

$$f(x) = \sum_{i \in s} \alpha_i K\left(x^*, x_i\right) + b.$$

A kernel is a function that quantifies the similarity of two observations. For example, for a *linear kernel* we could use:

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j},$$

where we quantifiy the similarity of pairs of observations using Pearson (standard) correlation.

However, we could use other forms of kernel to fit the support vector classifier in a higher-dimensional space, such as a *polynomial kernel*:

$$K(x_i, x_{i'}) = \left(\gamma \sum_{j=1}^{p} x_{ij} x_{i'j} + r\right)^d.$$

Another popular choice is the *radial kernel*:

$$K(x_i, x_{i'}) = \exp\left(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2\right).$$

Associated Exercises

Now might be a good time to try exercises 5-8.

References

1. James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning (Vol. 112, p. 18). New York: springer.
2. Géron, A. (2017). Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.".
3. Zheng, A., & Casari, A. (2018). Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists. " O'Reilly Media, Inc.".
4. Raschka, 2016
5. Cortes, C. and Vapnik, V. Support vector networks. Machine Learning, 20:273–297, 1995
6. Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. Data mining and knowledge discovery, 2(2), 121-167.
7. Murphy, K. P. (2012). Machine learning: a probabilistic perspective. MIT press.
8. Burkov, A. (2019). The hundred-page machine learning book (Vol. 1). Canada: Andriy Burkov.
9. Zhang, J. (2015). A complete list of kernels used in support vector machines. Biochem. Pharmacol.(Los Angel), 4, 2167-0501.
10. Raschka, Sebastian, and Vahid Mirjalili. "Python Machine Learning: Machine Learning and Deep Learning with Python." Scikit-Learn, and TensorFlow. Second edition ed (2017).
11. https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing
12. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
13. Al-Mejibli, I. S., Alwan, J. K., & Abd Dhafar, H. (2020). The effect of gamma value on support vector machine performance with different kernels. International Journal of Electrical and Computer Engineering, 10(5), 5497.


```
[NbConvertApp] Converting notebook 2_Support_Vector_Machines.ipynb to pdf
[NbConvertApp] Support files will be in 2_Support_Vector_Machines_files\
[NbConvertApp] Making directory .\2_Support_Vector_Machines_files
[NbConvertApp] Making directory .\2_Support_Vector_Machines_files
[NbConvertApp] Making directory .\2_Support_Vector_Machines_files
[NbConvertApp] Making directory .\2_Support_Vector_Machines_files
[NbConvertApp] Making directory .\2_Support_Vector_Machines_files
[NbConvertApp] Making directory .\2_Support_Vector_Machines_files
[NbConvertApp] Making directory .\2_Support_Vector_Machines_files
[NbConvertApp] Making directory .\2_Support_Vector_Machines_files
[NbConvertApp] Making directory .\2_Support_Vector_Machines_files
```

```
[NbConvertApp] Making directory .\2_Support_Vector_Machines_files
[NbConvertApp] Making directory .\2_Support_Vector_Machines_files
[NbConvertApp] Making directory .\2_Support_Vector_Machines_files
[NbConvertApp] Making directory .\2_Support_Vector_Machines_files
[NbConvertApp] Making directory .\2_Support_Vector_Machines_files
[NbConvertApp] Making directory .\2_Support_Vector_Machines_files
[NbConvertApp] Writing 53142 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] CRITICAL | x failed: xelatex notebook.tex -quiet
notebook.tex:508:

notebook.tex:766:

notebook.tex:779:
Traceback (most recent call last):
  File "C:\Users\delliot2\.conda\envs\mlp_pip\Scripts\jupyter-nbconvert-script.py", line 10, in <module>
    sys.exit(main())
  File "C:\Users\delliot2\.conda\envs\mlp_pip\lib\site-packages\jupyter_core\application.py", line 254, in launch_instance
    return super(JupyterApp, cls).launch_instance(argv=argv, **kwargs)
  File "C:\Users\delliot2\.conda\envs\mlp_pip\lib\site-packages\traitlets\config\application.py", line 845, in launch_instance
    app.start()
  File "C:\Users\delliot2\.conda\envs\mlp_pip\lib\site-packages\nbconvert\nbconvertapp.py", line 350, in start
    self.convert_notebooks()
  File "C:\Users\delliot2\.conda\envs\mlp_pip\lib\site-packages\nbconvert\nbconvertapp.py", line 524, in convert_notebooks
    self.convert_single_notebook(notebook_filename)
  File "C:\Users\delliot2\.conda\envs\mlp_pip\lib\site-packages\nbconvert\nbconvertapp.py", line 489, in convert_single_notebo
k
    output, resources = self.export_single_notebook(notebook_filename, resources, input_buffer=input_buffer)
  File "C:\Users\delliot2\.conda\envs\mlp_pip\lib\site-packages\nbconvert\nbconvertapp.py", line 418, in export_single_notebook
    output, resources = self.exporter.from_filename(notebook_filename, resources=resources)
  File "C:\Users\delliot2\.conda\envs\mlp_pip\lib\site-packages\nbconvert\exporters\exporter.py", line 181, in from_filename
    return self.from_file(f, resources=resources, **kw)
  File "C:\Users\delliot2\.conda\envs\mlp_pip\lib\site-packages\nbconvert\exporters\exporter.py", line 199, in from_file
    return self.from_notebook_node(nbformat.read(file_stream, as_version=4), resources=resources, **kw)
  File "C:\Users\delliot2\.conda\envs\mlp_pip\lib\site-packages\nbconvert\exporters\pdf.py", line 183, in from_notebook_node
    self.run_latex(tex_file)
  File "C:\Users\delliot2\.conda\envs\mlp_pip\lib\site-packages\nbconvert\exporters\pdf.py", line 153, in run_latex
    return self.run_command(self.latex_command, filename,
  File "C:\Users\delliot2\.conda\envs\mlp_pip\lib\site-packages\nbconvert\exporters\pdf.py", line 141, in run_command
    raise raise_on_failure(
nbconvert.exporters.pdf.LatexFailed: PDF creating failed, captured latex output:
Failed to run "xelatex notebook.tex -quiet" command:
notebook.tex:508:
```

```
notebook.tex:766:

notebook.tex:779:
```