1_Maximal_Margin_Classifiers

March 8, 2021

```
<IPython.core.display.HTML object>
```

1 Week 8 - Support Vector Machines

Dr. David Elliott

1. Introduction
2. What is a Hyperplane?
3. Maximal Margin Classifier

Common Notation

- 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.

2 1. Introduction

The term Support Vector Machines (SVM's) is sometimes used loosely to refer to three methods, each an extension of the previous method[1]:

- Maximal margin classifier,
- Support vector classifier,
- Support vector machine.

SVM's are a common supervised discriminative algorithm, well suited to *complex* small- to medium sized datasets[2].

They can be used for both **classification** and regression.

2.1 Dataset Example: Iris

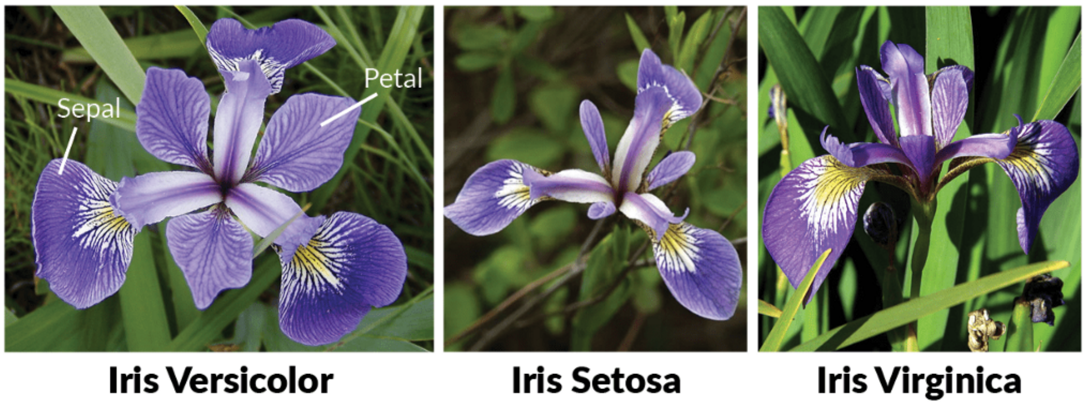
To demonstrate SVM's I'll be using Fisher's (or Anderson's) Iris flowers dataset[3].

The dataset consists of 50 samples each from three species of Iris (Iris setosa, Iris virginica and Iris versicolor).

Notes

- *“…This data comes from a famous experiment on a series of measurements of three species of iris flowers. R A Fisher, a statistically minded thinker in the early twentieth centure used this dataset in his 1936 paper The Use of multiple measurements in taxonomic problems, published in the Annals of Eugenics.”*[9]

`Figure 1: Iris Flowers`



Five attributes were collected for the 150 records.

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2
```

```
{0: 'Setosa', 1: 'Versicolor', 2: 'Virginica'}
```

```
   Sepal length (cm)  Sepal width (cm)  Petal length (cm)  Petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

  Species
0  Setosa
1  Setosa
2  Setosa
3  Setosa
4  Setosa
```

Notes

- Species is the typical target for its use in classification

Figure 2: Iris Attributes

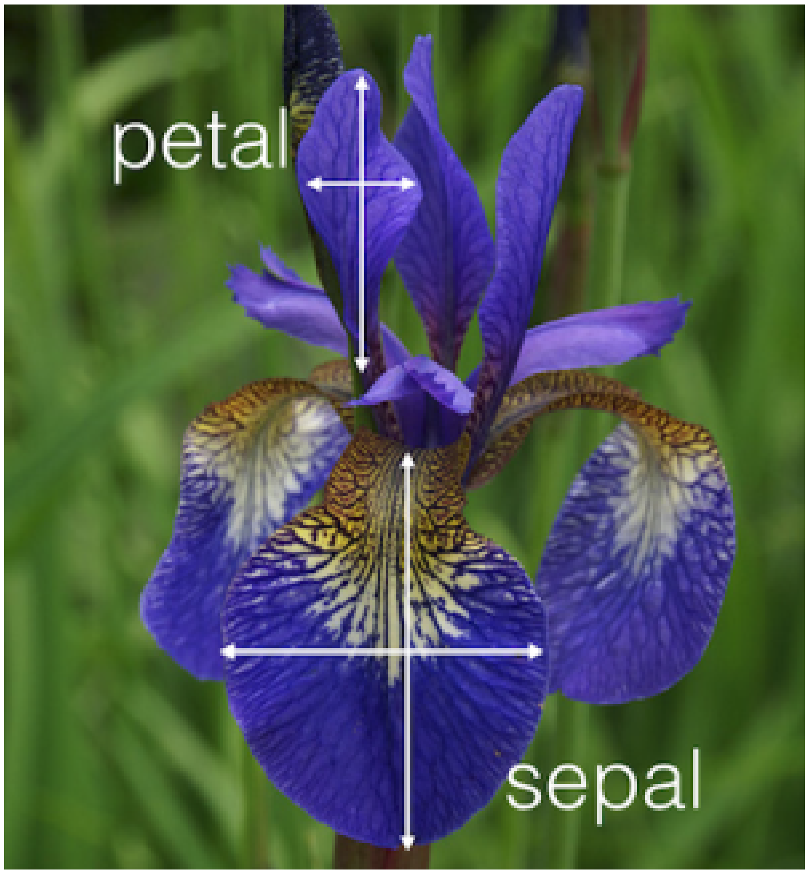


Figure 3: Iris Flower Dataset Pairplot

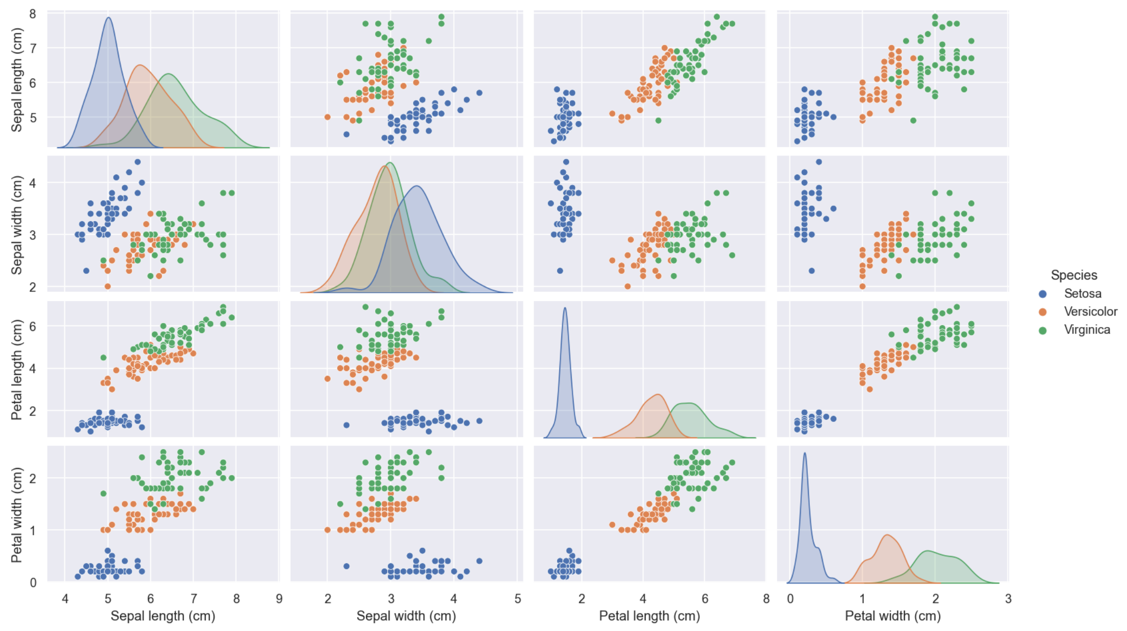


Figure 4: Petal length (cm) and Petal width (cm) of Setosa and Versicolor flowers

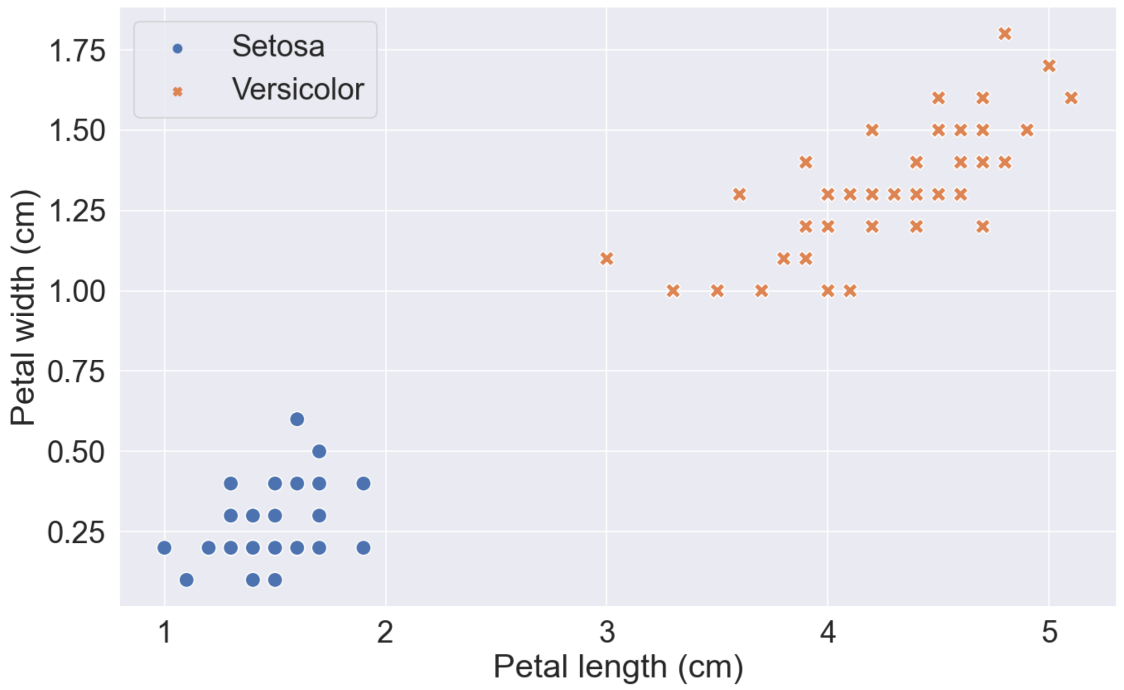
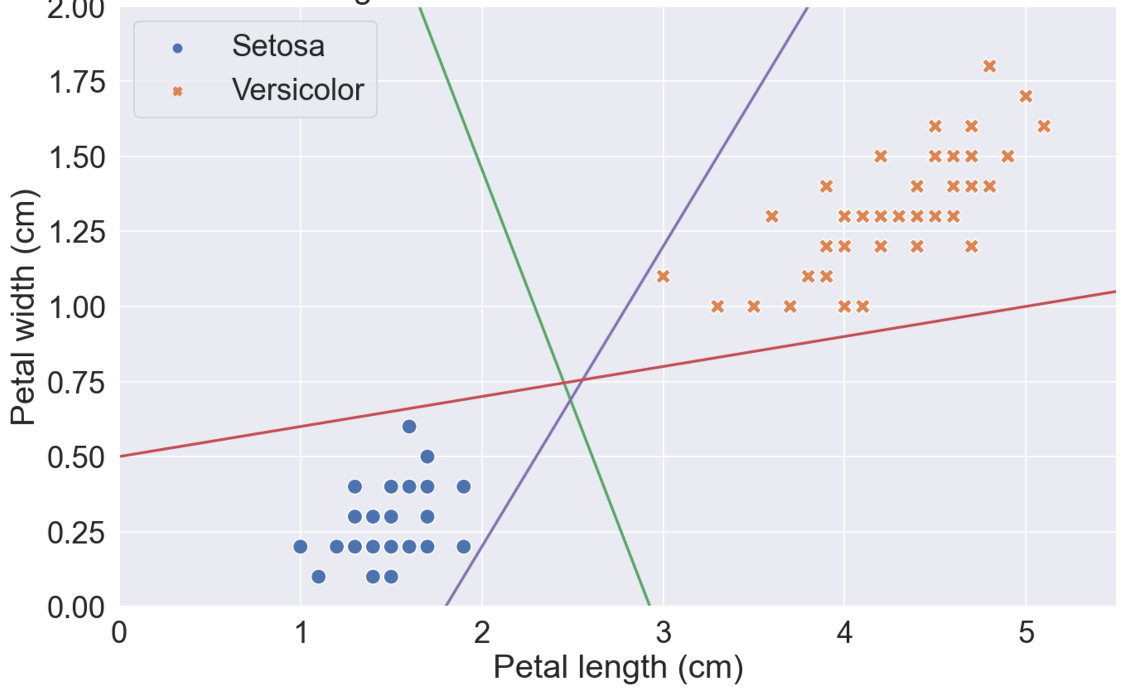


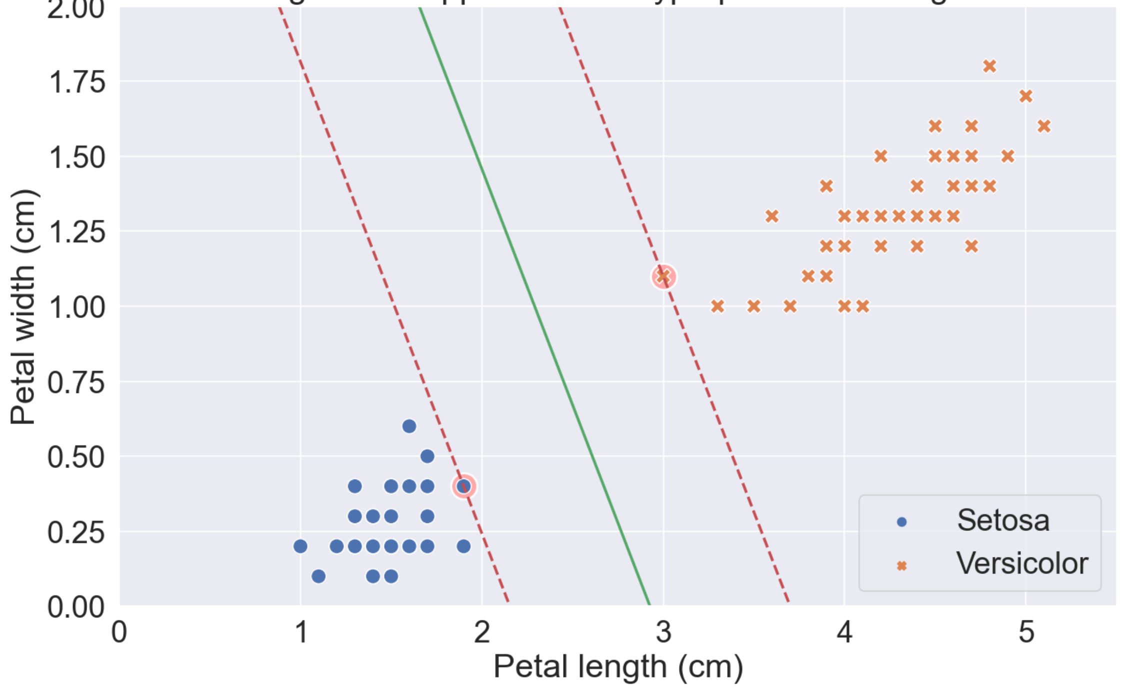
Figure 5: Potential Decision Boundaries



Notes

- We could use any one of the decision boundries above to separate the flower classes, but which one if the "best"?

Figure 6: Support Vector Hyperplane and Margin



Notes

- The middle thick line is a hyperplane and the dashed outer lines are the edges of the margin.
- The circled points are the class examples that fall on the margin. These will later be termed *support vectors*.

3 2. What is a Hyperplane?

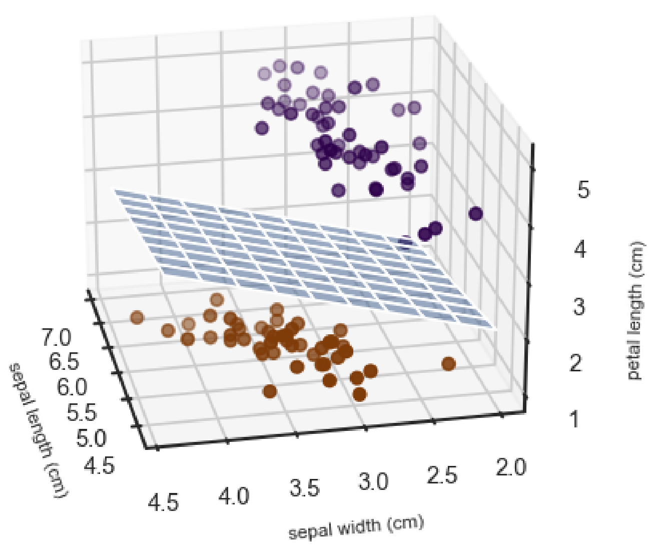
In p-dimensional space, a hyperplane is a flat affine subspace of dimension $p - 1$.

- *Two-dimensions*: A flat one-dimensional line
- *Three-dimensions*: A two-dimensional subspace
- *p-dimensions*: A $p - 1$ dimensional subspace

Notes

- "Affine" just means *"does not need to pass through the origin"*
- *Two-dimensions*: Figures 5 & 6
- *Three-dimensions*: Figure 7
- *p-dimensions*: …you'll have to use your imagination
- If you want to interact with the 3D plot use `%matplotlib notebook`

Figure 7: Support Vector Hyperplane in Three Dimensions



In more detail[1]:

Two-dimensions: $\beta_0 + \beta_1 \mathbf{x}_1 + \beta_2 \mathbf{x}_2 = 0$

Three-dimensions: $\beta_0 + \beta_1\mathbf{x}_1 + \beta_2\mathbf{x}_2 + \beta_3\mathbf{x}_3 = 0$

P-dimensions: $\beta_0 + \beta_1\mathbf{x}_1 + \beta_2\mathbf{x}_2 + ... + \beta_p\mathbf{x}_p = 0$

If $x = (\mathbf{x}_1, ..., \mathbf{x}_p)$ satisfies above, then it is a point on the hyperplane.

If $\beta_0 + \beta_1\mathbf{x}_1 + \beta_2\mathbf{x}_2 + ... + \beta_p\mathbf{x}_p > 0$, it lies on one side of the hyperplane,

so if, $\beta_0 + \beta_1\mathbf{x}_1 + \beta_2\mathbf{x}_2 + ... + \beta_p\mathbf{x}_p < 0$, its on the other side.

Notes

- In-other-words, p-dimensional space is dividided into two halves.

3.0.1 Classifying data

We aim to classify an $n \times p$ matrix of n observations in p dimensional space, with these observations falling into two classes $y_1, ..., y_n \in \{-1, 1\}$.

If we were to perfectly separate the classes the hyperplane would have the property that[1]:

$\beta_0 + \beta_1\mathbf{x}_{i1} + \beta_2\mathbf{x}_{i2} + ... + \beta_p\mathbf{x}_{ip} > 0 \quad \text{if } y_i = 1,$

$\beta_0 + \beta_1\mathbf{x}_{i1} + \beta_2\mathbf{x}_{i2} + ... + \beta_p\mathbf{x}_{ip} < 0 \quad \text{if } y_i = -1.$

For a new test observations x^*, we would look at the sign of:

$$f(x^*) = \beta_0 + \beta_1 x_1^* + \beta_2 x_2^* + ... + \beta_p x_p^*.$$

We would assign it to class 1 if $f(x^*)$ is positive and class -1 if negative.

Furthermore, we could use the magnitude of $f(x^*)$ to indicate how far the point lies from the hyperplane.

4 3. Maximal Margin Classifier

We need a reasonable way of constucting a hyperplane, out of the possible choices.

Maximal margin hyperplanes look at getting the hyperplane that is the furthest from the training observations - we compute the perpendicular distance from each training observation to a given separating hyperplane.

The maximal margin hyperplane is the separating hyperplane for which the margin is largest.

We hope the classifier with a large margin on the training data will generalise well to unseen test observations.

Another way of defining our hyperplane is:

$$\mathbf{w} \cdot \mathbf{x} + b = 0,$$

which uses the dot product between a weight vector, $\mathbf{w}$, and our input vector $\mathbf{x}$, plus our bias, b (sometimes defined as w_0).

In statistics the dot product of two n-dimensional vectors $\mathbf{a}$ and $\mathbf{b}$ is often noted as $\mathbf{a} \cdot \mathbf{b}$. The dot product is a type of *inner product*, often denoted as $\langle \mathbf{a}, \mathbf{b} \rangle$.

Notes

- $\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} \mathbf{a}_i \mathbf{b}_i$

In Machine Learning, vectors are typically represented as *column vectors*, so the dot product is defined as $\mathbf{a}^{\mathrm{T}}\mathbf{b}$. Using this notation, our hyperplane is[10]:

$$\mathbf{w}^{\mathrm{T}}\mathbf{x} + b = 0.$$

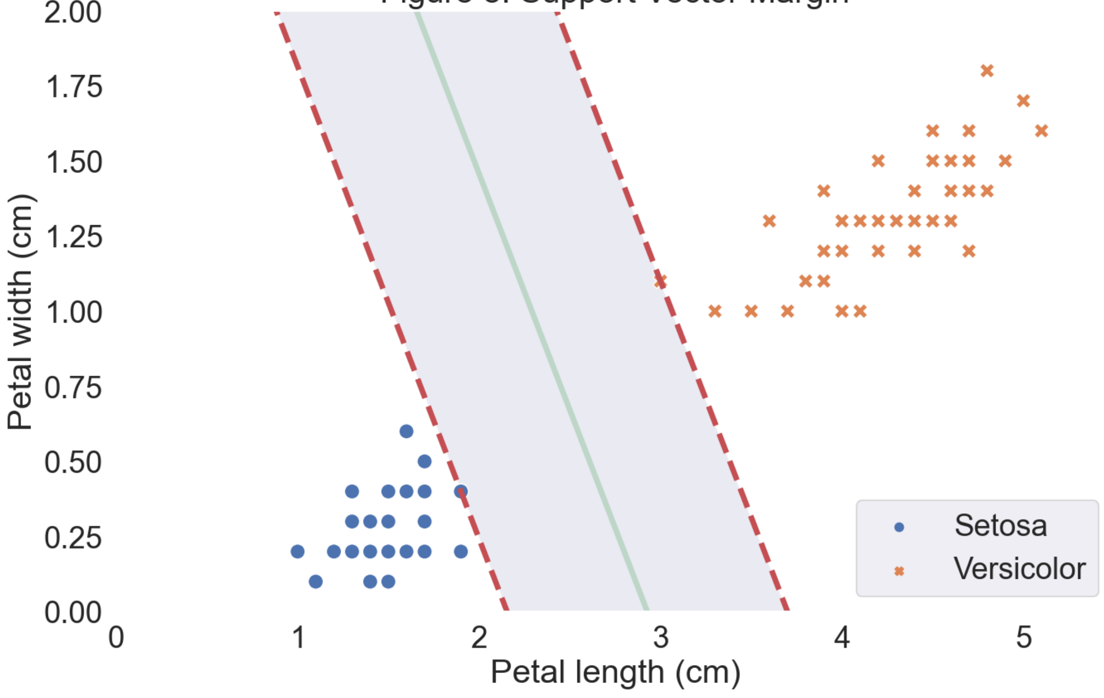
Our margins, where our labels $y \in \{-1, 1\}$, are then:

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

which can be more compactly written as:

$$y_i \left(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b\right) \geq 1 \quad \forall_i$$

Figure 8: Support Vector Margin



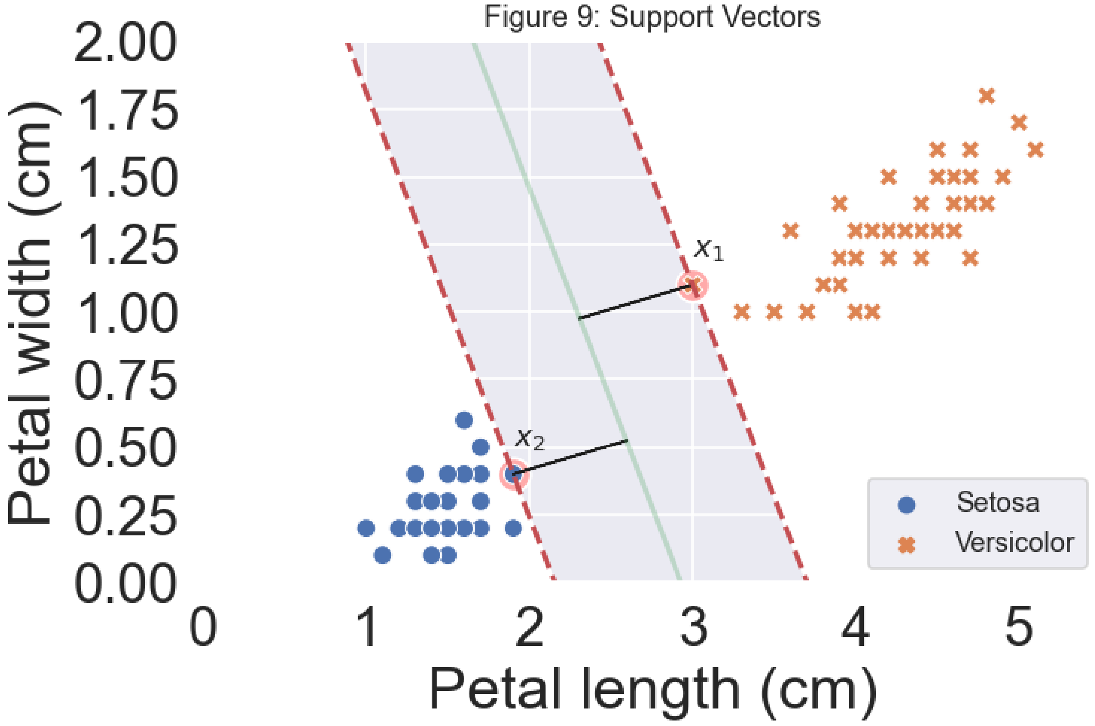
There are two equidistant points from the maximal margin hyperplane, lying on the dashed lines. These observations are called *Support Vectors*.

We can call these two points, x_1 and x_2 as below:

$$\mathbf{w}^{\mathrm{T}}x_1 + b = 1,$$

$$\mathbf{w}^{\mathrm{T}}x_2 + b = -1.$$

Figure 9: Support Vectors



If we move our support vectors our hyperplane will too.

This is because the maximal margin hyperplane only depends on these support vectors.

Other data points could be moved without the hyperplane moving.

```
<__main__.MatplotlibFigure at 0x20719ee57f0>
```

We want to maximise the distance between the margin lines, on which the points lie[10].

$$x_1 - x_2$$

$$\frac{\mathbf{w}^{\mathrm{T}}x_1 + b = 1 - \mathbf{w}^{\mathrm{T}}x_2 + b = -1}{\mathbf{w}^{\mathrm{T}}(x_1 - x_2) = 2}$$

We can normalise this equation by the length (Euclidean norm) of the vector **w**:

$$||\mathbf{w}|| = \sqrt{\sum_{j=1}^{p} w_j^2}$$

$$\frac{\mathbf{w}^{\mathrm{T}}(x_1 - x_2)}{||\mathbf{w}||} = \frac{2}{||\mathbf{w}||}$$

Notes 1. we want the distance between x_1 and x_2 2. our b cancels out 3. we want to remove $\mathbf{w}^{\mathrm{T}}$ 4. we do this with norm of $\mathbf{w}^{\mathrm{T}}$. However we cannot remove $||\mathbf{w}||$, so we are left to maximise $\frac{2}{||\mathbf{w}||}$.

```
<__main__.MatplotlibFigure at 0x2071a4152e0>
```

We want to $\max\frac{2}{||\mathbf{w}||}$, while classifying everything correctly, $y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \geq 1 \quad \forall_i$.

Or, instead:

$\min\limits_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2 \quad \text{s.t.} \quad y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \geq 1 \quad \forall_i,$

which is easier because this is a *convex quadratic optimisation problem* which is efficiently solvable using quadratic programming algorithms.

Notes

- $\frac{1}{2}$ is added for convience.

4.0.1 Primal Problem

This requires a *Lagrangian* formulation of the problem so we introduce Lagrange multipliers, $\alpha_i, i = 1, \ldots, n$:

$$\min L_P = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{n} \alpha_i y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) + \sum_{i=1}^{n} \alpha_i \qquad \text{s.t.} \quad \forall_i \alpha_i \geq 0$$

4.0.2 Duel Problem[2,6,7]

Removes the dependence on $\mathbf{w}$ and b,

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

To achive this we first need to set a constaint,

$$\sum_{i}^{n} \alpha_i y_i = 0.$$

Now we can find the vector $\hat{\alpha}$ that maximises the equasion using a *QP solver*.

Notes

- The duel problem allows us later to use the kernel trick.

- $\sum_i \alpha_i y_i = 0$ removes our b.
- There is a Lagrange multiplier α_i for every training point. Points for which $\alpha_i > 0$ are called "support vectors", and lie on one of the margins, with all other training points having $\alpha_i = 0$.
- α_i will only be zero if all training points have the same class.

Knowing our $\hat{\alpha}_i$ means we can find the weights, $\widehat{\mathbf{w}}$, which are a linear combination of the training inputs, $\mathbf{x}_i$, training outputs, y_i, and the values of α,

$$\widehat{\mathbf{w}} = \sum_{i \in s} \hat{\alpha}_i y_i \mathbf{x}_i,$$

where s is the collection of indicies of support vectors[6]. We only need the support vectors as they are the only ones with $\hat{\alpha}_i > 0$.

Out bias term, $\hat{b}$, is then determined by[2]:

$$\hat{b} = \frac{1}{n_s} \sum_{i=1}^{n} \left(y_i - \widehat{\mathbf{w}}^{\mathrm{T}} \mathbf{x}_i \right)$$

where n_s is the number of support vectors.

Notes

- We only need to focus on those with $\alpha_i > 0$.
- Solving the SVM problem is equivalent to finding a solution to the Karush-Kuhn-Tucker (KKT) conditions.
  - it is through the use of using the following Karush-Kuhn-Tucker (KKT) condition we can find b:
$$\alpha_i(y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1) = 0 \quad \forall i.$$
- It is numerically safer to take the average value over all support vectors, $\frac{1}{N_S}$.
- For more reading on quadratic programming for support vector machines, I recommend you read:
  - Appendix C in Géron, A. (2017). Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.".
  - Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. Data mining and knowledge discovery, 2(2), 121-167.
  - https://www.adeveloperdiary.com/data-science/machine-learning/support-vector-machines-for-beginners-duality-problem/

4.1 Inner products, similarity, and SVMs[8]

Question: But why bother doing this? That was a lot of effort, why not just solve the original problem?

Answer: Because this will let us solve the problem by computing the just the inner products ($\mathbf{x}_i^{\mathrm{T}} \mathbf{x}_k$) which will be important when we want to solve non-linearly separable classification problems.

- Inner products provide a measure of *"similarity"*
- Inner product in 2D between 2 vectors of unit length returns the cosine of the angle between them. In otherwords, how *"far apart"* they are.

- if they are parallel their inner product is 1 (completely similar).
- If they are perpendicular (completely unlike) their inner product is 0 (so should not contribute to the correct classifier).

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

Case 1 Two observations $\mathbf{x}_i$, $\mathbf{x}_k$ are completely *dissimilar* (orthogonal), so their dot product is 0. They don't contribute to L.

Case 2 Two observations $\mathbf{x}_i$, $\mathbf{x}_k$ are similar and predict the *same* output value y_i (ie. both $+1$ or -1). This means $y_i \times y_k = 1$ and the value $\alpha_i \alpha_k y_i y_k \mathbf{x}_i \mathbf{x}_k$ is positive. However this *decreases* the value of L, due to subtracting from the first term sum, $\sum_i^n \alpha_i$, so the algorithm downgrades similar feature vectors that make the *same* prediction.

Case 3 : Two observations $\mathbf{x}_i$, $\mathbf{x}_k$ predict opposite predictions about the output value y_i (ie. one $+1$ and the other -1), but are otherwise similar. The value $\alpha_i \alpha_k y_i y_k \mathbf{x}_i \mathbf{x}_k$ is negative and since we are subtracting it, this adds to the sum. These are the examples that maximise the margin width.

```
<__main__.MatplotlibFigure at 0x207192b84c0>
```

4.2 Recap[1]

The solution to the maximal margin classifier problem involves only the *inner products* of the observations:

$$\langle x_i, x_{i'} \rangle = \sum_{j=1}^p x_{ij} x_{i'j}.$$

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:

$$f(x) = \sum_{i=1}^n \hat{\alpha}_i \langle x^*, x_i \rangle + \hat{b}.$$

In the above case, for estimating the parameters $\hat{\alpha}_1 ..., \hat{\alpha}_n$ and $\hat{b}$, we need the $n(n-1)/2$ inner products $\langle x_i, x_{i'} \rangle$ between all pairs of training observations.

However, $\hat{\alpha}$ is nonzero only for support vectors, so if we have a collection of their indicies, s, we can do the following instead:

$$f(x^*) = \sum_{i \in s} \hat{\alpha}_i \langle x^*, x_i \rangle + \hat{b}.$$

4.3 Limitations

- If we have a large number of features, this approach often leads to overfitting.
- Maximal Margin Classifiers are sensitive to outliers.

```
<__main__.MatplotlibFigure at 0x20719e7a100>
```

Notes

- In figure 13 above, we can see that the reliance on a small number of observations means there is now a small margin.
- We want to be confident that a distance from the hyperplane is a measure of our confidence in its classification, and that we have no overfit to our training data.

In other cases, no exact linear separating hyperplane exists. Therefore we may want to use a hyperplane that *almost* separates the two classes, allowing some errors, using a *soft margin* (Support Vector Classifier).

```
<__main__.MatplotlibFigure at 0x207169c4c40>
```

5 Associated Exercises

Now might be a good time to try exercises 1-4.

6 References

1. James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning. 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. Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. Annals of eugenics, 7(2), 179-188.
4. Murphy, K. P. (2012). Machine learning: a probabilistic perspective. MIT press.
5. Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. Data mining and knowledge discovery, 2(2), 121-167.
6. Fletcher, R. Practical Methods of Optimization. John Wiley and Sons, Inc., 2nd edition, 1987
7. http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf
8. https://bookdown.org/koehlerj/qr_book/introducing-the-iris-dataset.html
9. Raschka, Sebastian, and Vahid Mirjalili. "Python Machine Learning: Machine Learning and Deep Learning with Python." Scikit-Learn, and TensorFlow. Second edition ed (2017).

```
[NbConvertApp] Converting notebook 1_Maximal_Margin_Classifiers.ipynb to html
[NbConvertApp] Writing 3742148 bytes to 1_Maximal_Margin_Classifiers.html
[NbConvertApp] Converting notebook 1_Maximal_Margin_Classifiers.ipynb to pdf
[NbConvertApp] Support files will be in 1_Maximal_Margin_Classifiers_files\
[NbConvertApp] Making directory .\1_Maximal_Margin_Classifiers_files
[NbConvertApp] Making directory .\1_Maximal_Margin_Classifiers_files
[NbConvertApp] Writing 41705 bytes to notebook.tex
[NbConvertApp] Building PDF
```

```
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | b had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 885418 bytes to 1_Maximal_Margin_Classifiers.pdf
[NbConvertApp] Converting notebook 1_Maximal_Margin_Classifiers.ipynb to slides
[NbConvertApp] Writing 3920087 bytes to 1_Maximal_Margin_Classifiers.slides.html
```