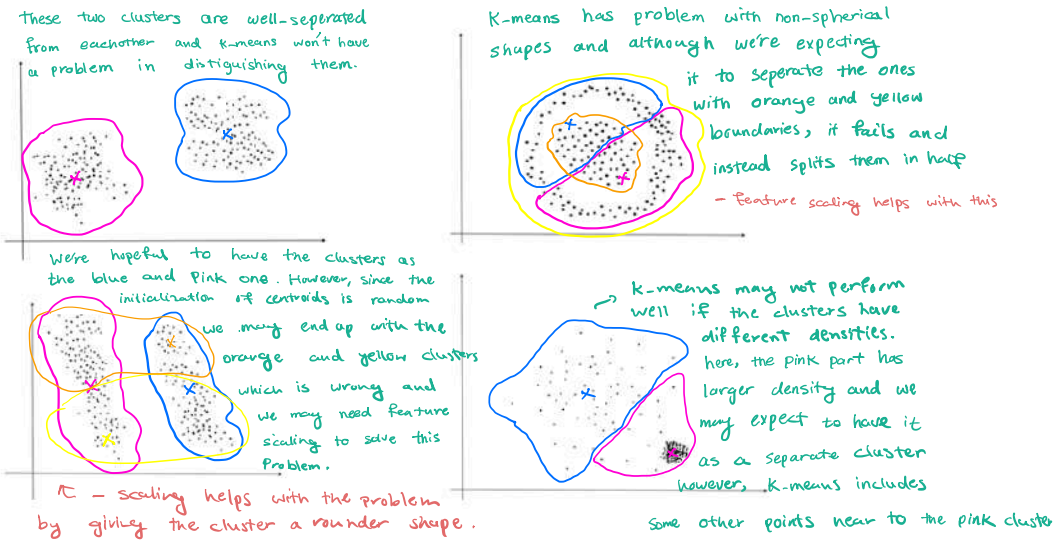
1. Clustering and K_means

1.1 Intuitive understanding: clusters in the examples blew are seperated with blue and pink colors.



- K-means uses cluster centers which each are obtained as the arithmetic mean of all the points belonging to that specific cluster. Also, each point is closer to its own cluster center than to other cluster centers.

Centroids are usually initialized to random but after running the algorithm several times and assigning data points to the nearest cluster, centroids stop changing after a while. However, the result may vary depending on the random initialization of the centroids.

In the first one, K-means result is almost the same as what we as humans may expect. However, in the second one, we may recognize the clusters by orange and yellow boundaries. Whereas, K-means usually goes by splitting it in half and although it's still minimizing the within cluster sum of squares, it fails to cluster the points correctly. (however by using the polar coordination, we can solve this problem.)

- Usually if variables show different variances, it is a good idea to standardize before K-means. Leaving variances unequal, is equivalent to putting more weight on variables with smaller variance so clusters will tend to be separated along variables with

greater variance. Feature scaling might be helpful with the non-spherical shapes problem and the third one

1.2 Finding proper value of K

- Inertia measures how well a dataset was clustered by K-means. It is calculated by measuring the distance between each data point and its centroid, squaring this distance and summing these squares across one cluster. A good model is one with low interia and a low number of clusters (K).

The Silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette ranges from -1 to +1. where a high value indicates that the object is well matched to its own cluster. and poorly matched to neighboring clusters.

Hence, they both provide valuable information for clustering analysis and it may be a good idea to use both plots just to make sure that you select the most optimal number of clusters. Inertia (elbow) method is easier to implement but it only calculates the euclidean distance whereas silhouette takes into account variables such as variance, skewness, high-low differences, etc.

Some suggest silhouette score over inertia since it uses inter and intra cluster distances in its scoring function while the elbow method only uses intra-cluster distances. Morever, the inertia graph is always decreasing (we have lower scores as k decreases) but this doesn't mean we have the optimal value for k (silhouette is not like this and is better this way).

- When using silhouette score for determining the optimal value for K, we should consider diagrams that all the clusters' plot is beyond average silhouette score, with mostly uniform thickness and do not have wide fluctuations in the size.

With this explanation, K=3 and K=6 are a bad pick for the data due to the presence of clusters

with below average silhouette scores. Moreover, K=4 has larger fluctuations in the size of the plots (cluster 1 is much larger than others) and thus, K=5 seems like an optimal choice for K in this example where all the plots have almost the same thickness.

1.3 Applications of Clustering

Active learning is a special case of machine learning in which a learning algorithm can interactively query a user to label data with desired outputs. We use clustering to label the data more efficiently. User can label the points in each cluster and continue to do this until the data for train is ready

Semi-supervised learning is a type of machine learning. It refers to a learning problem that involves a small portion of labeled examples and a large number of unlabeled examples from which a model must learn and make predictions on new examples.

One idea is to use clustering on all the data points and then use its results to label the examples with higher importance. For example we can choose the points which are closest to their cluster centroids as a candidate point of that cluster. We can label them and use them to train our model.

Active learning is a form of semi-supervised learning. Unlike fully supervised learning, the ML algorithm is only given an initial subset of human-labeled data out of a larger, unlabeled dataset. The algorithm processes that data and provides a prediction with a certain confidence level. However, AL (active learning) and SSL (semi-supervised learning) are quite different. In each iteration, SSL collects data with greatest confidence label (the label is the output of the model trained with high-confidence labels). However, in each iteration, AL collects data with least confidence output, query their label from, say, human experts (output comes from the model trained with previously labeled data and newly labeled data which is labeled by human experts).

Sources used: wikipedia_ medium_ virtualflux - machinelearningmastery

2 Whitening using PCA

By whitening we want to make the features less correlated with one another and give all of the features the same variance. We do this by 1. projecting the dataset onto the eigenvectors (to rotate the data so that there is no correlation between the components) 2. Normalize the dataset to have a variance of one (this is done by simply dividing each component by the square root of its eigenvalue).

- First, we want to make sure that our data has zero mean. We can achieve this by subtracting mean from each example: $\mu = \frac{1}{N}\sum x$, $X := X - \mu$
- Next we compute the covariance matrix Σ as follows: $\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)})(x^{(i)})^T$
- Then PCA computes the eigenvectors of Σ, to do this we can make use of svd decomposition and obtain U matrix, where $\Sigma = U\Lambda U^T$ and $U = \begin{bmatrix} | & | & & | \\ u_1 & u_2 & \cdots & u_n \\ | & | & & | \end{bmatrix}$ contains the eigenvectors of Σ (u_1 is the principal vector corresponding to the largest eigenvalue). Morever, the diagonal entries of matrix Λ will contain the corresponding eigenvalues.
- Next we compute x_{rot} as $x_{rot} = U^T x$ which represents our data in $(u_1, u_2, \ldots, u_n)$-basis. and somhow rotates our data.

- Afterwards in order to reduce the dimesion of $x \in \mathbb{R}^n$ to a k dimensional representation $\tilde{x} \in \mathbb{R}^k$ (where $k < n$), we would take the first k components of x_{rot} which correspond to the top k directions of variation. $\tilde{x} = \begin{bmatrix} x_{rot,1} \\ \vdots \\ x_{rot,k} \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ and the final n-k components of $\tilde{x}$ will be defined to zero. Hence, there is no need to keep these zeros around and we can define $\tilde{x}$ as a k-dimensional vector
- To make each of the input features have a variance equal to one, we can rescale each feature $x_{rot,i}$ by $\frac{1}{\sqrt{\lambda_i}}$. As a result, we can define our whitened data $x_{PCAwhite} \in \mathbb{R}^n$ as:

$$x_{PCAwhite,i} = \frac{x_{rot,i}}{\sqrt{\lambda_i}}$$

Which is equivalent to $X_{PCA} \rightarrow \Lambda^{\frac{-1}{2}} \cdot U^T \cdot X$

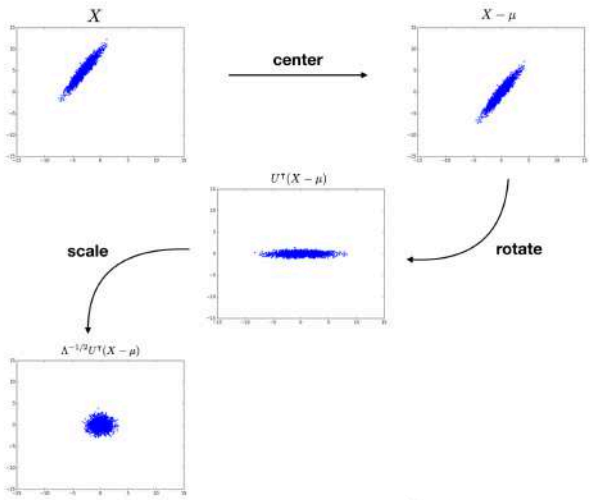
Now we show that this matrix has a covariance of 1:

$$X_{PCA}\, X_{PCA}^T = \Lambda^{-\frac{1}{2}} U^T X X^T U (\Lambda^{-\frac{1}{2}})^T \xrightarrow{\Lambda \text{ is diagonal}} \Lambda^{-\frac{1}{2}} U^T \underbrace{X X^T}_{\Sigma} U \Lambda^{-\frac{1}{2}} = \Lambda^{-\frac{1}{2}} \underbrace{U^T U}_{I} \Lambda \underbrace{U^T U}_{I} \Lambda^{-\frac{1}{2}}$$

$$= I$$

Hence, we have proved that it has identity matrix as its covariance matrix.

Also, since we had subtracted the mean from each feature in the initial step, the result also remains with a mean equal to zero.



3 Linear Regression

3.1 Lagrange Multipliers

3.2 Ridge Regression : We know that in ridge regression we use the power 2 of the weight in regularization and the goal in this model is to minimize the following function : $\sum_{i=1}^{m}(y - Xw)^2 + \alpha \sum_{j=1}^{p} w_j^2$ which is also equivalent to $L(w) = \|Xw - y\|_2^2 + \lambda \|w\|_2^2$. In order to minimize it, we take its derivative with respect to w:

$$L(w) = (Xw - y)^T(Xw - y) + \frac{\lambda}{2} w^T w = (w^T X^T - y^T)(Xw - y) + \lambda w^T w$$
$$= w^T X^T X w - w^T X^T y - y^T X w + y^T y + \lambda w^T w$$

using the results of the question 1 in the previous HW, we know that: $\frac{\partial x^T A x}{\partial x} = x^T(A + A^T)$. Also we know that $\frac{\partial Ax}{\partial x} = A$ and $\frac{\partial x^T A}{\partial x} = A^T$

Hence, we have:

$$\frac{\partial L(w)}{\partial w} = 2w^T X^T X - y^T X - y^T X + 0 + 2\lambda w^T$$
$$\longrightarrow w^T X^T X - y^T X + \lambda w^T = 0$$
$$\longrightarrow w^T(X^T X + \lambda I) = y^T X$$
$$\longrightarrow w^T = y^T X (X^T X + \lambda I)^{-1}$$
$$\longrightarrow w^* = (X^T X + \lambda I)^{-1} X^T y$$

4. Generalization Error

- Dimensionality reduction reduces overfitting. When having fewer features, the complexity of model decreases and hence, this helps mitigate overfitting. This way, DR algorithms such as PCA usually remove the noise in the data and keep the most important features, resulting in a simpler model and less overfitting

- Data augmentation methods:

* Some data augmentation techiques for images:

- Position Augmentation → the pixel positions of an image is changed.

1. Scaling: Image is resized to a given size 2. Cropping 3. Flipping 4. Padding 5. Rotation

5. translation (ex. Affine transformation which preserves points, straight lines and planes.

6. Distortions (which should be representations of the type of noise/distortions in the test set. Usually it does not help to add purely random/meaningless noise to the data.)

- Color augmentation: Change the brightness, contrast, suturation, hue. Convert to grayscale

- Some advanced methods such as generative adversarial networks and neural style trasfer methods are also used for data augmentation.

* Data augmentation for speech: we can add noisy background or do:

1. Time warping 2. Frequency masking 3. Time masking

* Data augmentation for NLP:

1. Back translation: translate the text data to some language and then translate it back to the original language.

2. Easy data augmentation → Synonym replacement - Random Insertion. Random Swap - Random deletion.

3. NLP Albumentation → Shuffle sentence transform - Exclude duplicate transform -

sources used: iq_opengenus - towards data science - neptune.ai

5 Kernels

5.1 Feature Space

We can rewrite $(1 + \cos(x,y))^2$ as follows using the formula $\cos(x,y) = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}| \cdot |\vec{y}|}$:

$$(1 + \cos(x,y))^2 = 1 + \cos^2(x,y) + 2\cos(x,y) = 1 + \frac{(\vec{x} \cdot \vec{y})^2}{|\vec{x}|^2 \cdot |\vec{y}|^2} + 2\frac{\vec{x} \cdot \vec{y}}{|\vec{x}||\vec{y}|}$$

$$= 1 + \frac{(\sum x_i y_i)(\sum x_i y_i)}{|x|^2 \, |y|^2} + 2\frac{\sum x_i y_i}{|x||y|}$$

In order to obtain the result above in our dot production, we build the feature space vectors as follows:

$$\phi(x) = \begin{bmatrix} 1 \\ \frac{\sqrt{2}x_1}{|x|} \\ \vdots \\ \frac{\sqrt{2}x_n}{|x|} \\ \frac{x_1 x_1}{|x|^2} \\ \vdots \\ \frac{x_n x_n}{|x|^2} \end{bmatrix} \qquad \phi(y) = \begin{bmatrix} 1 \\ \frac{\sqrt{2}y_1}{|y|} \\ \vdots \\ \frac{\sqrt{2}y_n}{|y|} \\ \frac{y_1 y_1}{|y|^2} \\ \vdots \\ \frac{y_n y_n}{|y|^2} \end{bmatrix}$$

the first component is 1

the next n values are in the form of $\frac{\sqrt{2}z_i}{|z|}$

and the next n^2 value are $\frac{z_i z_j}{|z|^2}$ for i and j in the range of 1 to n.

$$\phi(x)\,\phi(y) = \begin{bmatrix} 1 \\ \frac{2x_1 y_1}{|x||y|} \\ \vdots \\ \frac{2x_n y_n}{|x||y|} \\ \frac{x_1 x_1 \, y_1 y_1}{|x|^2 \, |y|^2} \\ \vdots \\ \frac{x_n x_n \, y_n y_n}{|x|^2 \, |y|^2} \end{bmatrix} \rightarrow 1 + 2\frac{\sum x_i y_i}{|x||y|} + \frac{(\sum x_i y_i)(\sum x_i y_i)}{|x|^2 \, |y|^2} \Rightarrow \text{which gives us the same result}$$

Setting $n=2$ we have:

$$\phi(x) = \begin{bmatrix} 1 \\ \frac{\sqrt{2}x_1}{|x|} \\ \frac{\sqrt{2}x_2}{|x|} \\ \frac{x_1 x_1}{|x|^2} \\ \frac{x_1 x_2}{|x|^2} \\ \frac{x_2 x_1}{|x|^2} \\ \frac{x_2 x_2}{|x|^2} \end{bmatrix}$$

5.2 Kernel Matrix

5.2.1 We that in order to have a semi-definite matrix such as K, we can prove that for every vector $\underline{u}$, we should have: $u^T K u \geq 0$

So suppose that we have an arbitary vector $u = [u_1, \dots, u_m]^T$ then we should prove that the following multiplication result is positive:

$$[u_1, \dots, u_m] \; K \begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix} = [u_1, \dots, u_m] \begin{bmatrix} K(x^{(1)}, x^{(1)}) & K(x^{(1)}, x^{(2)}) & \cdots & K(x^{(1)}, x^{(m)}) \\ K(x^{(2)}, x^{(1)}) & \ddots & & \\ \vdots & & & \\ K(x^{(m)}, x^{(1)}) & & & K(x^{(m)}, x^{(m)}) \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix}$$

$$= \left[\sum_{i=1}^{m} K(x^{(i)}, x^{(1)}) u_i, \dots, \sum_{i=1}^{m} (x^{(i)}, x^{(m)}) u_i \right] \begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix} = \sum_{j=1}^{m} \sum_{i=1}^{m} K(x^{(i)}, x^{(j)}) u_i u_j$$

$$= \sum_{j=1}^{m} \sum_{i=1}^{m} \phi(x_j)^T \phi(x_i) u_i u_j = \left(\sum_{j=1}^{m} \phi(x_j)^T u_j\right) \cdot \left(\sum_{i=1}^{m} \phi(x_i) u_i\right) = \left(\sum_{j=1}^{m} \phi(x_j) u_j\right)^T \cdot \left(\sum_{i=1}^{m} \phi(x_i) u_i\right)$$

$$= \left\| \sum_{i=1}^{m} \phi(x_i) u_i \right\|_2^2$$

which is a positive number.

5.2.2 We want to show that $K_{i,j} = \phi(x_i)\, \phi(x_j)$ (for a specific ϕ)

We know that K is a symmetric matrix; hence we can write it in the form of, $K = PDP^T$ where P is an orthogonal matrix consisting of the eigenvectors of K and D is a diagonal matrix consisting of its eigenvalues. Also we know that all the eigen values in a semi-definite matrix are non-negative.

Using the facts above we define a feature mapping $\phi(x^i) = \begin{bmatrix} \sqrt{\lambda_1}(\vec{V_1})_i \\ \sqrt{\lambda_2}(\vec{V_2})_i \\ \vdots \\ \sqrt{\lambda_m}(\vec{V_m})_i \end{bmatrix}$ where V_is are the eigenvectors in $P = [\vec{V_1}, \vec{V_2} \dots, \vec{V_m}]$

To show that $K_{i,j} = \phi(x_i)\,\phi(x_j)$, we expand K using $K = PDP^T$:

$$PD = \left[\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_m\right]\begin{bmatrix} \lambda_1 & & 0 \\ & \lambda_2 & \\ 0 & & \ddots \\ & & & \lambda_m \end{bmatrix} = \left[\lambda_1 \vec{v}_1, \lambda_2 \vec{v}_2, \ldots, \lambda_m \vec{v}_m\right]$$

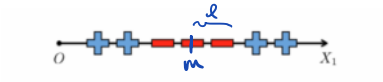
$$PD_{(i,k)} = \lambda_k (v_k)_i \rightarrow PDP^T{}_{(i,j)} = \boxed{\sum_{k=1}^{m} \lambda_k (v_k)_i (v_k)_j}$$

This term is equal to $\phi(x^i) \cdot \phi(x^j) = \begin{bmatrix} \sqrt{\lambda_1}(\vec{v}_1)_i \\ \sqrt{\lambda_2}(\vec{v}_2)_i \\ \vdots \\ \sqrt{\lambda_m}(\vec{v}_m)_i \end{bmatrix} \cdot \begin{bmatrix} \sqrt{\lambda_1}(\vec{v}_1)_j \\ \sqrt{\lambda_2}(\vec{v}_2)_j \\ \vdots \\ \sqrt{\lambda_m}(\vec{v}_m)_j \end{bmatrix} = \sum_{k=1}^{m} \lambda_k (v_k)_i (v_k)_j$

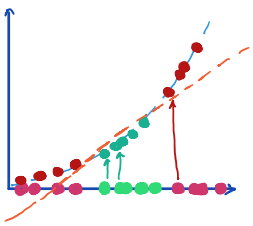
Thus, we proved that $K_{(i,j)} = \phi(x^i) \cdot \phi(x^j) \rightarrow$ K is a dot product

6 Feature Expansion

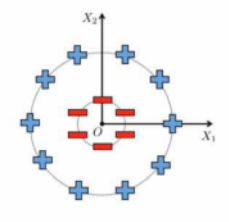
6.1



for this one, we can use the power two of each element (x_1^2)

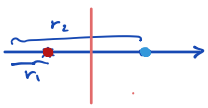


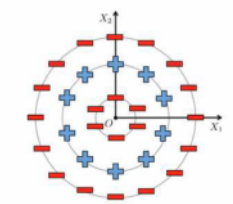
(The other solution is to use $|X_1 - m|$ and then set a threshold (ℓ) to separate the points)



Clearly, we can use the equation of a circle and we can seperate the red and blue circles.

→ $X_1^2 + X_2^2$ → we will end up with two values (points) for one and thus they can be separated easily.





first use the trick in previous one $x_1^2 + x_2^2$ →  → next use the trick in the first example ✓

6.2

By expanding this equation we have:

$$(x_1 - a)^2 + (x_2 - b)^2 - r^2 = 0$$

$$x_1^2 + a^2 - 2ax_1 + x_2^2 + b^2 - 2bx_2 - r^2 = 0$$

$$x_1^2 + x_2^2 - 2ax_1 - 2bx_2 + (a^2 + b^2 - r^2) = 0$$

By using the feature space (x_1, x_2, x_1^2, x_2^2) and the coeffients of $(-2a, -2b, 1, 1)$ and intercept $a^2 + b^2 - r^2$, we can show that circulem regions are linearly separable in this feature space.

7 SVM Decision Boundaries

We can easily see that the only possible linear kernels can belong to the figures (b), (c) and (f). The reason for this is that (a), (d) and (e) clearly can not be linear and hence are RBF kernels.

Now note that the given equation used in linear kernels is as follows:

$$\min_{w,b,\varepsilon} \ \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{N}\varepsilon_i$$

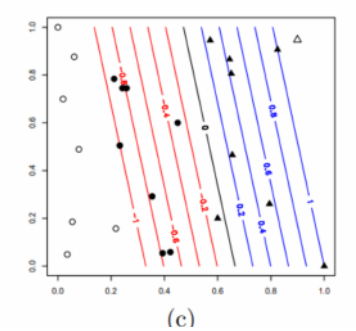
$$\text{s.t.} \quad y^{(i)}(w^T x^{(i)} + b) \geq 1 - \varepsilon_i$$

$$\varepsilon_i \geq 0 \qquad \forall\, i \in \{1,2,\dots,N\}$$

In the optimization problem above, $\varepsilon_1,\dots,\varepsilon_n$ are the slack variables and are added to the inequality constraint to transform it to an equality. Our goal is to mimize the error values which show how far each point is from the margin and the hyperplane.

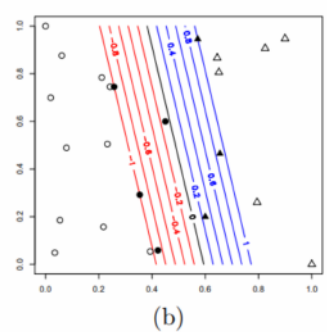
Mirever, C is a hyperparameter used for regularization and controls the trade off between bias and variance, the larger C is, we are more interested in having smaller errors and the margin becomes smaller since the models becomes more sensitive to not miss-classify points. On the other hand, when C is larger, margin becomes bigger.

With this explanations, the corresponding values for linear kernels are as below:



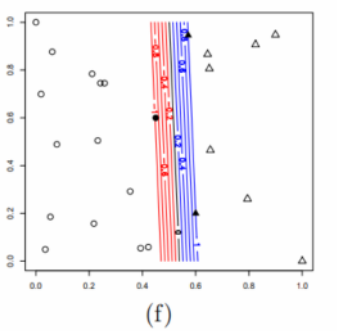
(c)

C = 0.1



(b)

C = 1



(f)

C = 10

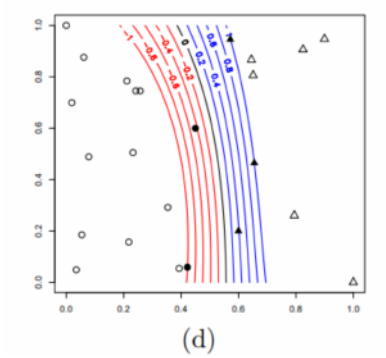
Also for the RBF kernels, the influence of the parameters C and Gamma are described below:

The gamma parameter defines how far the influence of a single training example reaches. with low values meaning 'far' and high values meaning 'close'. The lower values of gamma result in models with lower accuracy and model is too contrained and cannot capture the complexity or "shape" of the data (The resulting model will behave like similarly to a linear model). Intermediate values of gamma give a model with good decision boundaries. Morever, large values of gamma result in overfitting and the radius of the area of influence of the support vectors only include the support vector itself.

The regularization parameter C is the same as before and cotrols the penalty of missclassification

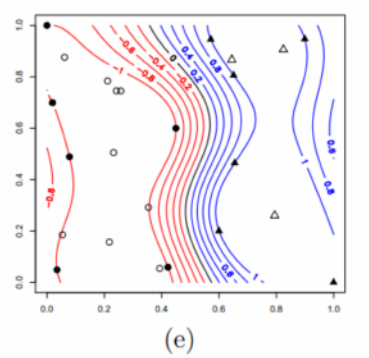
C↑ : tolerance to missclassification ↓ : margin ↑

C↓ : tolerance to missclassification ↑ : margin ↓



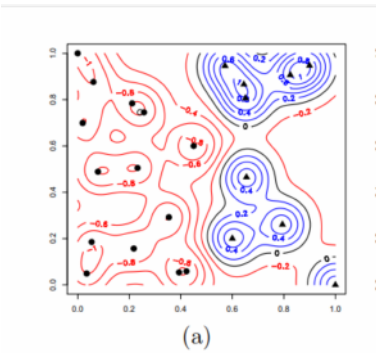
(d)

C = 15 γ = 0.1



(e)

C = 3 γ = 1



(a)

C = 1 γ = 10

8 Monte Carlo Cross Validation

8.1 The larger n_t is in MCCV, the lower the bias and the higher the variance. When the training set size is large, we have more duplicates of each point in the iterations and hence, it gets overfitted on the training data.

8.2 K_fold divides the data points into K mutually exclusive subsets of equal size. Next, uses only one of these K subsets in each iteration as the test set. (it has K iterations) Certainly, when using MCCV, we have more variation in the way these points are partitioned and included in the train and test sets. (although we usually don't have all the $\binom{N}{n_t}$ partitions.) However, in k_folds, we can be sure that each data point is included (exactly once) in the test set

Usually we expect a result with higher bias and lower variance in MCCV (after running it for a decent amount of iterations). Averaging the K results obtained in k_fold algorithm gives us a good estimate of its performance but may have a higher variance.