{desafío}
latam_

Unidad 2:
Máquinas de Soporte Vectorial_

{desafío}
latam_

Alcances de la lectura asignada

- Comprender el principio de la maximización del márgen como un clasificador.
- Diferenciar entre la clasificación estrcita y flexible de márgen.
- Entrenar modelos mediante `Pipelines`.
- Implementar modelos SVC.
- Identificar el kernel radial-basis-function y su implementación en SVC.
- Manipular hiperparámetros mediante la técnica GridSearchCV.

Activación de Conceptos

- En la unidad anterior aprendimos sobre Análisis del Discriminante Lineal.
- ¡Póngamos a prueba lo aprendido!

Cuál de las siguientes frases es incorrecta respecto a los modelos generativos

- Un modelo generativo aprende características a partir de $\Pr(x, y) \quad \forall x \in \mathbf{X}^{\mathbb{R}}$.
- Un modelo generativo asigna clases $\Pr(y|x)$ mediante el teorema de Bayes.
- Un modelo generativo asigna clases mediante una función objetivo $f : \mathbf{X} \mapsto y$

¿Cuál es la diferencia entre LDA y QDA?

- LDA es generativo y QDA es discriminativo.
- En LDA Σ se asume constante, mientras que en QDA $\Sigma_{k \in \mathcal{K}}$
- LDA busca maximizar el márgen, mientras que QDA busca maximizar la elipsoide.

¿Qué diferencia existe entre la regresión logística y LDA?

- La regresión logística es más exigente en supuestos que LDA.
- Si se cumplen condiciones de normalidad multivariada, LDA es asinóticamente eficiente.
- No existen diferencias, ambos entregan el mismo resultado.

¿Cuál es el proceso de LDA?

- Extraer $\boldsymbol{\mu}$,$\boldsymbol{\Sigma}$ → estimar Multivarnorm($\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$)→ Extraer discriminante lineal.
- Extraer $\boldsymbol{\mu}_k$,$\boldsymbol{\Sigma}_k$ → estimar Multivarnorm($\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$)→ Extraer discriminante lineal.
- Extraer $\boldsymbol{\mu}_k$,$\boldsymbol{\Sigma}_k$ → Extraer discriminante lineal. → Asignar clases Pr($y \in \mathbb{Y}|x$)

Clasificadores de Máximo Márgen

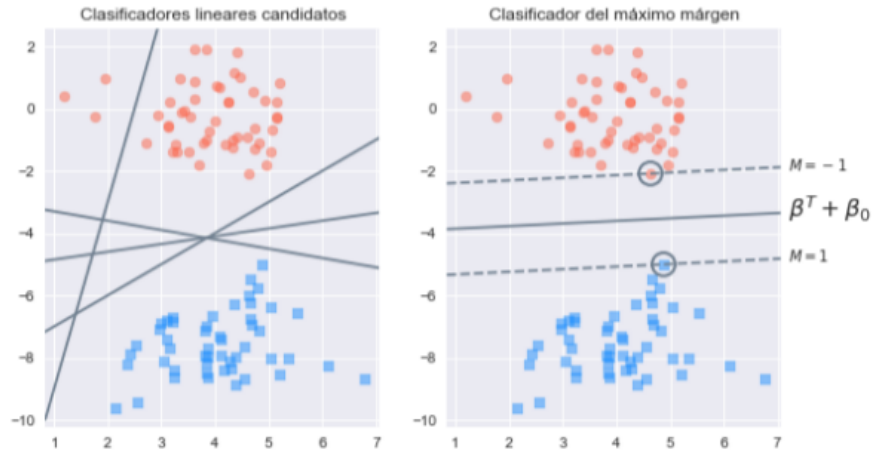
{desafío}
latam_

Motivación

- Existen casos donde no hay clasificadores únicos (más de un modelo llega a la misma conclusión).
- También casos donde los clasificadores fallan en el plano lineal.
- SVM obtiene un clasificador en base a la distancia de los atributos entre cada clase ⇝ **Principio de maximización de márgenes**.

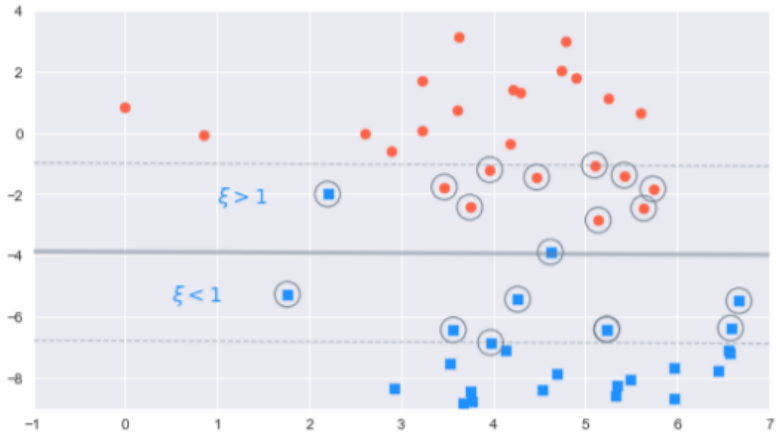
Caso separable

In [2]:
```
plt.figure(figsize=(9,5));afx.setup_svm_problem()
```



Caso no separable

In [3]:
```
plt.figure(figsize=(9, 5));afx.svm_non_separable(plot_slacks=True, plot_xi=True)
```



Clasificación estricta y flexible

- SVM es un problema de clasificación flexible ⇝ *permitimos un rango de valores que se pueden situar dentro de los márgenes.*

- **Slacks**:
  - $(\xi > 1)$: Cercano al márgen contrario.
  - $(\xi < 1)$: Cercano al márgen correcto.

- La estrategia de optimización es minimizar la tasa de $(\xi > 1)$

Implementación con sklearn

{desafío}
latam_

Consideraciones

- Dado que SVM se basa vectores de soporte, las escalas de los atributos afectan a la solución ⇝ Implementamos una estrategia de transformación.
- Podemos estandarizar ⇝ Todas las observaciones tendrán una distribución $\mathcal{N}(0, \sigma_i^2)$.

In [4]:
```
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
```

`sklearn.pipeline`

- Resulta que en la medida que nuestros modelos requieren de más pasos, nuestro código necesita ordenarse entorno al procedimiento.
- `Pipeline` ⇝ concatenación de preprocesamiento y procesos de un estimador.

In [5]:
```
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
pipeline_model = Pipeline([('scale', StandardScaler()),
                ('model', SVC())])
```

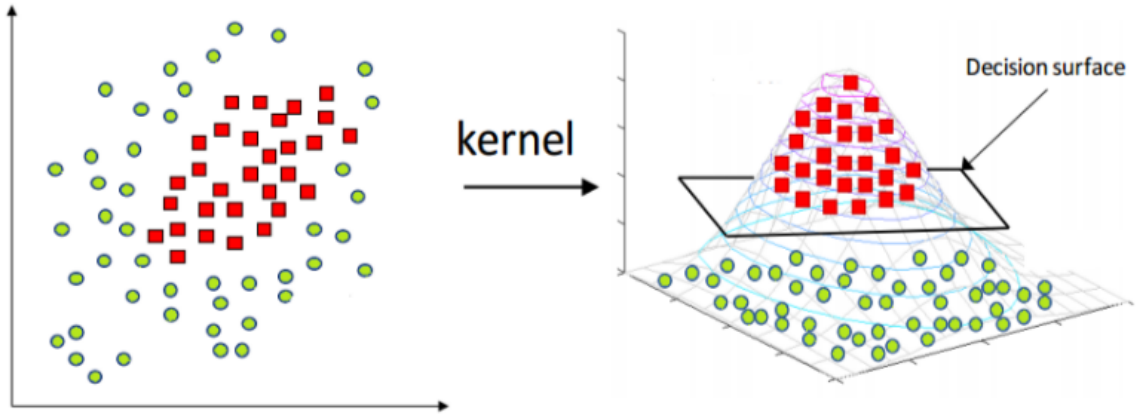
- Posteriormente se pueden aplicar funciones clásicas de estimadores`sklearn`.

Kernelización

{desafío}
latam_

¿Qué es la kernelización?

- Reexpresar nuestra matriz $\mathbf{X}$ en un espacio donde sean separables.



Tipos de kernels

Kernel	Función
(Gaussian) Radial Basis Function [`rbf`]	$\kappa(x, x^{\mathsf{T}}) = \exp(-\gamma \lvert\lvert x - x^{\mathsf{T}} \rvert\rvert^2)$
Polinomiales de orden-n [`poly`]	$\kappa(x, x^{\mathsf{T}}) = (1 + \langle x, x^{\mathsf{T}} \rangle)^d$
Sigmoide [`sigmoid`]	$\kappa(x, x^{\mathsf{T}}) = \tanh(k_1 \langle x, x^{\mathsf{T}} \rangle + k_2)$

Hiperparámetros

{desafío}
latam_

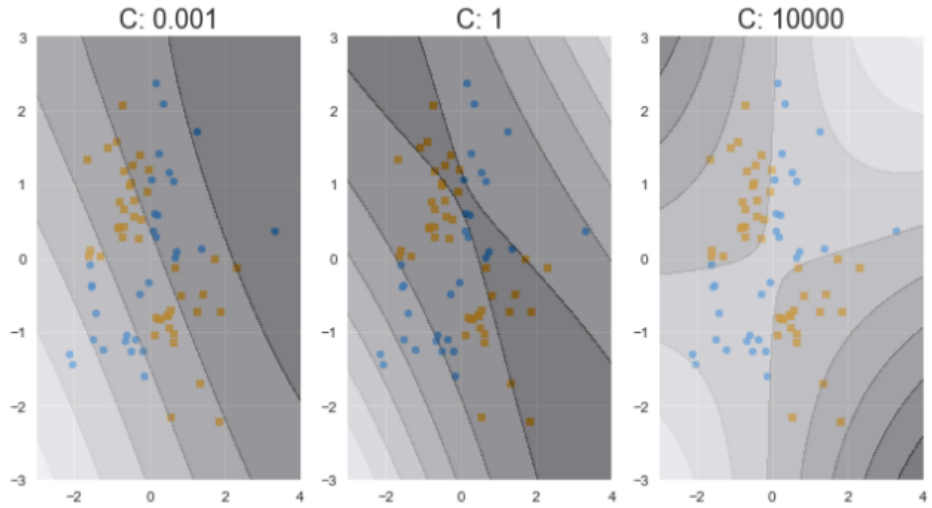
Hiperparámetros en SVM

- Debemos fijarnos en minimización del error y en la especificidad del modelo respecto a los datos de entrenamiento.

C

- Penalización en observaciones incorrectamente clasificadas.
- En términos visuales, refleja el ajuste del kernel en los datos de entrenamiento.
- Valores bajos: Mayor sesgo.
- Valores altos: Mayor varianza.

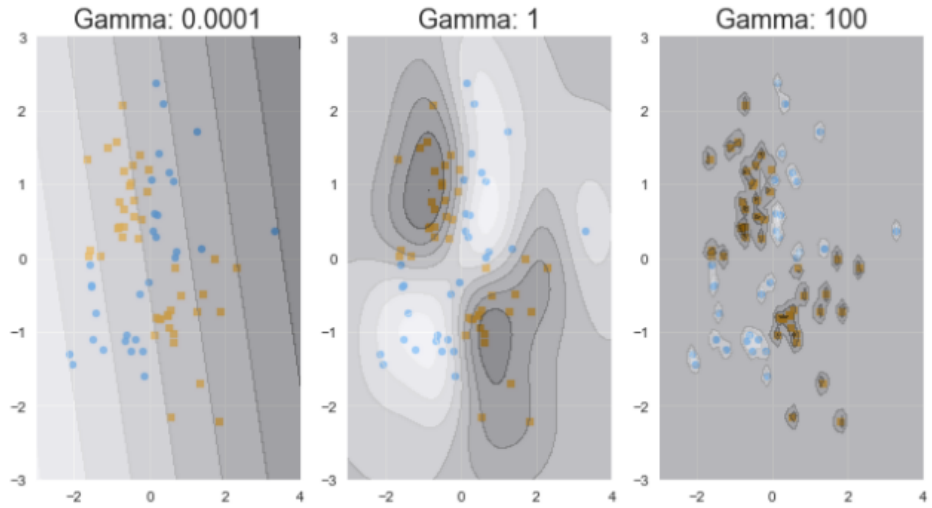
In [4]:
```
X, y = afx.svm_logical_xor_data(nsize=75); plt.figure(figsize=(9,5));
afx.svm_c_hyperparameter(X, y, c_range=[0.001, 1, 10000])
```



Gamma

- Controla la influencia de un punto específico en la función de decisión.
- En términos visuales, refleja el ajuste de la función de decisión en los datos de entrenamiento.
- Valores bajos: Mayor sesgo.
- Valores altos: Mayor varianza.

In [5]:
```
plt.figure(figsize=(9,5));afx.svm_gamma_hyperparameter(X, y, gamma_range=[0.0001, 1, 100
])
```



Búsqueda en Grilla con CV

- Cantidad de modelos estimables:

$$\#\text{Modelos Estimables} = \left(\prod_{i=1}^{P} \theta_i \right) \times \#\text{CV}$$

- **Objetivo**: tener una matriz entre validaciones (filas) y combinaciones de hiperparámetros (columnas).

- Computacionalmente demandante.
- Se pueden especificar la cantidad de núcleos del procesador a implementar mediante `n_jobs`. `n_jobs=-1` ocupa todos los núcleos.

Elección de parámetros en SVC (con `GridSearchCV`)

- Estandarizamos

In [8]:
```
X = StandardScaler().fit_transform(df)
```

- Definimos una grilla de valores

In [9]:
```
params = {'C':[1, 10, 100, 1000, 10000, 100000],
          'gamma': [0.00001, 0.0001, 0.001, 0.01, 0.1, 1]}
```

- Instanciamos el modelo

In [10]:
```
from sklearn.model_selection import GridSearchCV
# lo hice con menos validaciones cruzadas
estimatecv = GridSearchCV(SVC(kernel='rbf'),params, cv=3)
```

- Entrenamos

In [12]:
```
estimatecv.fit(X_train, y_train)
```

Out[12]:
```
GridSearchCV(cv=3, error_score='raise',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=
0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
```

Objetos relevantes en `GridSearchCV`

Mejor combinación de hiperparámetros

In [13]:
```
print("Hyperparams combination: {}".format(estimatecv.best_params_))
```

```
Hyperparams combination: {'C': 10, 'gamma': 0.1}
```

Desempeño en el Test Set

In [14]:
```
print("Test score (given best combination): {}".format(round(estimatecv.best_score_, 3)))
```

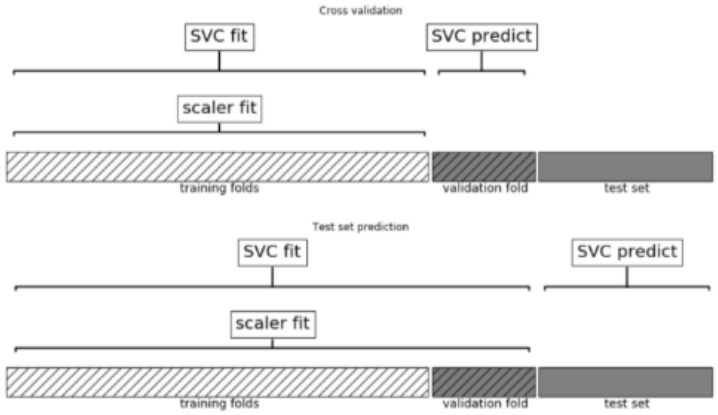
```
Test score (given best combination): 0.768
```

Valores de cada modelo estimable

In [15]:
```
estimatecv.cv_results_['mean_test_score'][:10]
```

Out[15]:
```
array([0.6394761 , 0.6394761 , 0.70565257, 0.74494485, 0.76424632,
       0.75689338, 0.6394761 , 0.70519301, 0.74678309, 0.75413603])
```

Hold-out set


Cross validation
SVC fit
SVC predict
scaler fit
training folds
validation fold
test set
Test set prediction
SVC fit
SVC predict
scaler fit
training folds
validation fold
test set

Estimación en el hold-out set

- `best_estimator_` permite entrenar el modelo con la mejor combinación de hiperparámetros.

In [16]:
```
estimatecv.best_estimator_.fit(X_train, y_train)
```

Out[16]:
```
SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

- Posteriormente podemos comparar el desempeño en el hold-out set

In [17]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, estimatecv.predict(X_test)))
```

```
             precision    recall  f1-score   support

          0       0.81      0.85      0.82      1330
          1       0.73      0.67      0.70       815

avg / total       0.78      0.78      0.78      2145
```