{desafío}
latam_

Ensambles Secuenciales: Boosting _

Sesión Presencial 1

Itinerario

Activación de conceptos

Desarrollo Desafío

Panel de discusión

{desafío}
latam_

Activación de conceptos

¿Cuáles son los pasos de Bagging?

1. Entrenamos una serie de modelos $h \in \mathcal{H}$, generamos predicciones, aleatorizamos y ponderamos.
2. Dado un conjunto de datos, se muestra dentro de éstos con Bootstrap y posteriormente entrenamos, generamos predicciones y asignamos valor.
3. Entrenamos iterativamente un modelo $h \in \mathcal{H}$, corrigiendo por error de clasificación.

¿Podemos implementar Bagging con `sklearn.linear_model.LogisticRegression`?

- Si podemos, de hecho genera mejores resultados que con un árbol de decisión.
- No podemos.
- Si podemos, pero Bagging funciona mejor con modelos que presenten alta varianza.

¿Cuál es el principal problema de Bagging?

- No presenta problema alguno, dado que minimiza la varianza y estabiliza el sesgo.
- Tiene una tendencia a correlacionar las predicciones de cada modelo en el ensamble.
- Presenta el problema de maximizar la varianza de los modelos en el ensamble

¿Cómo resuelve Random Forest el problema de modelos correlacionados?

- De manera adicional al muestreo de datos, selecciona atributos de forma aleatoria en cada modelo.
- Genera un muestreo de los datos en el conjunto de entrenamiento.
- Penaliza las estimaciones del modelo mediante ℓ_1 o ℓ_2.

¿Cuál es una de las principales desventajas de Random Forest?

- Es sensible a la escala de los atributos.
- Tiene una baja interpretabilidad dado la selección aleatorizada de atributos.
- Es ineficiente en espacios N-dimensionales.

No Free Lunch

- Wolpert y Macready (1997): No existe un algoritmo que presente soluciones satisfactorias para todos los problemas.
- Implementar múltiples algoritmos es la mejor manera de encontrar la solución óptima.

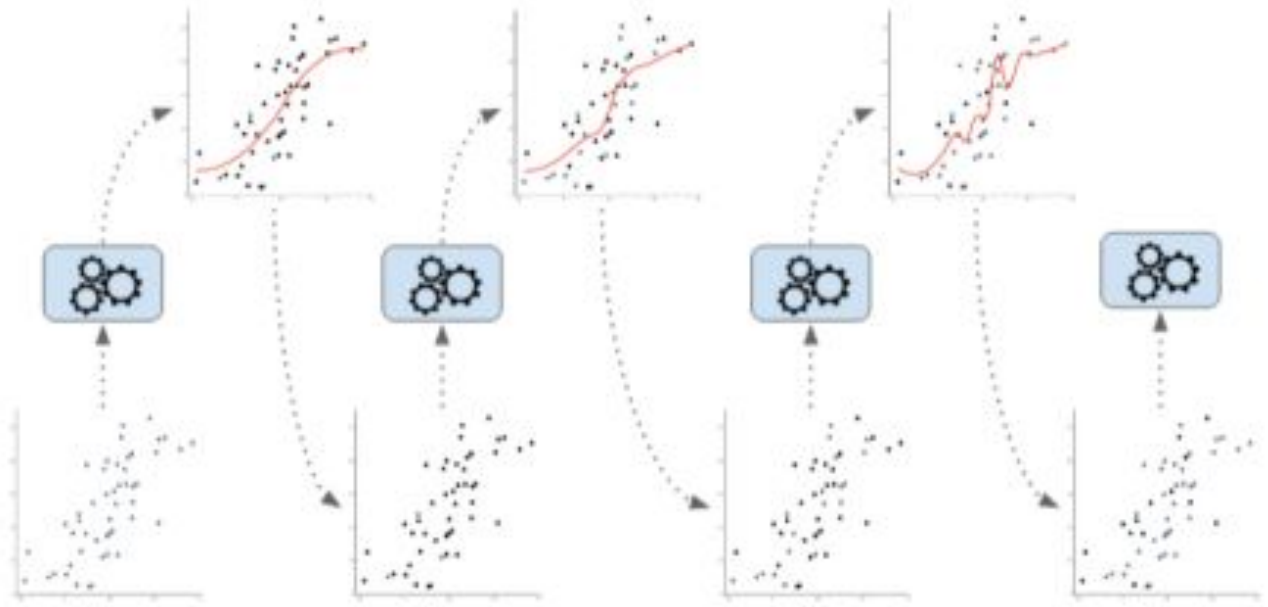
Razones para implementar múltiples soluciones:

1. Necesidades de la industria
2. Pureza de los datos
3. Interpretabilidad de los modelos

¿Qué sabemos hasta ahora?

- Modelos de instancia específica:
  - Regresión Lineal
  - Logística
  - Algoritmos generativos (Naive Bayes, LDA)
  - Máquinas de Soporte Vectorial
  - Árboles…
- Modelos de ensamble paralelo:
  - Bagging
  - Random Forest

Ensambles secuenciales



AdaBoost: Boosting Adaptativo

Rudimentos I

- Dado un vector objetivo $y \in \mathcal{Y} = \{-1, 1\}$ y una matriz de atributos **X**, entrenamos un clasificador débil
- Con un ensamble **H** de clasificadores $h(\cdot)$ podemos estimar la tasa de error de una observación en específico:

$$\bar{\varepsilon}_i = \frac{\sum_{i=1}^{N} \mathbb{I}\left(y_i \neq h(x_i)\right)}{N}$$

Rudimentos II

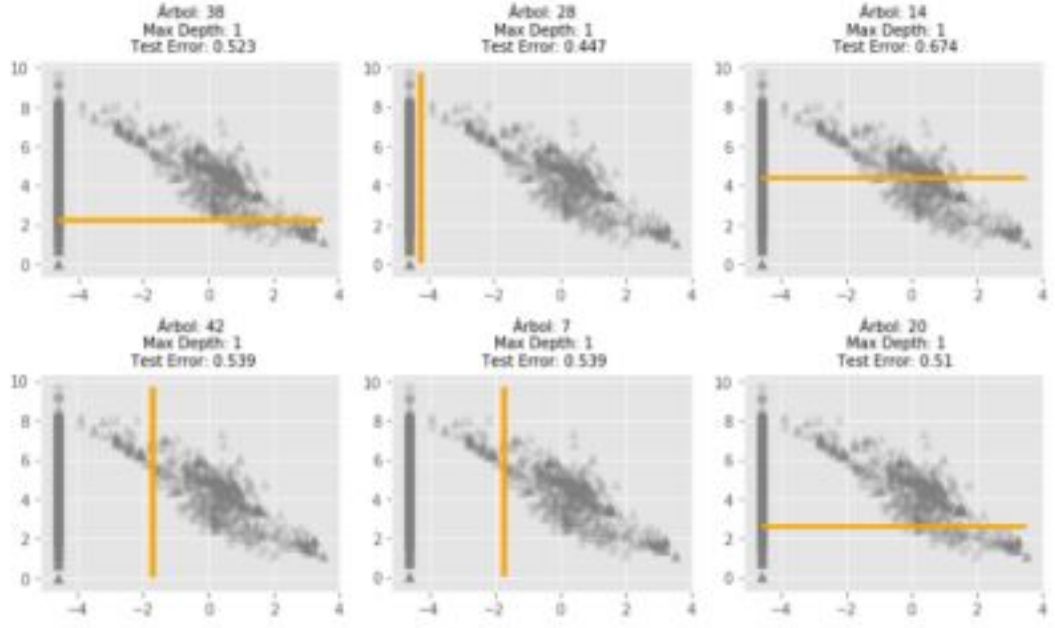
- Predicción de una clase en un ensamble:

$$H_m = \text{sign}\left(\sum_{i=1}^{M} \alpha_m h_m(x) \right)$$

- Objetivo: Ponderar con α_m en base a la tasa de error de los clasificadores para una observación.
- El algoritmo busca predecir, estimar y el error y actualizar los pesos recorriendo todos los modelos.

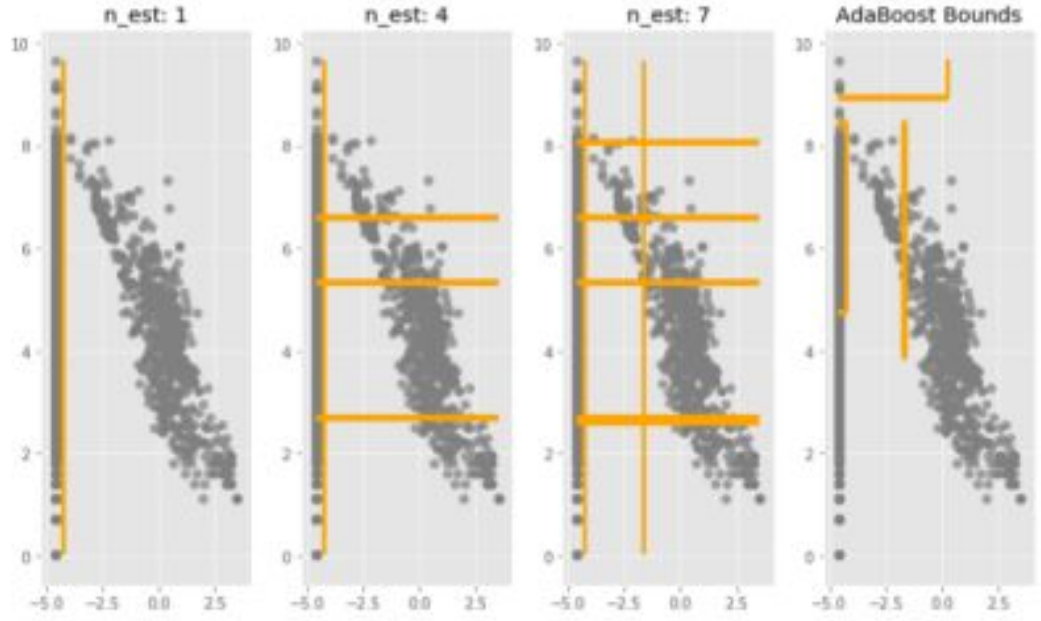
AdaBoost como secuencia de clasificadores

In [25]:
```
np.random.seed(42)
afx.adaboost_weak_learner_behavior()
```



AdaBoost como concatenación de clasificadores

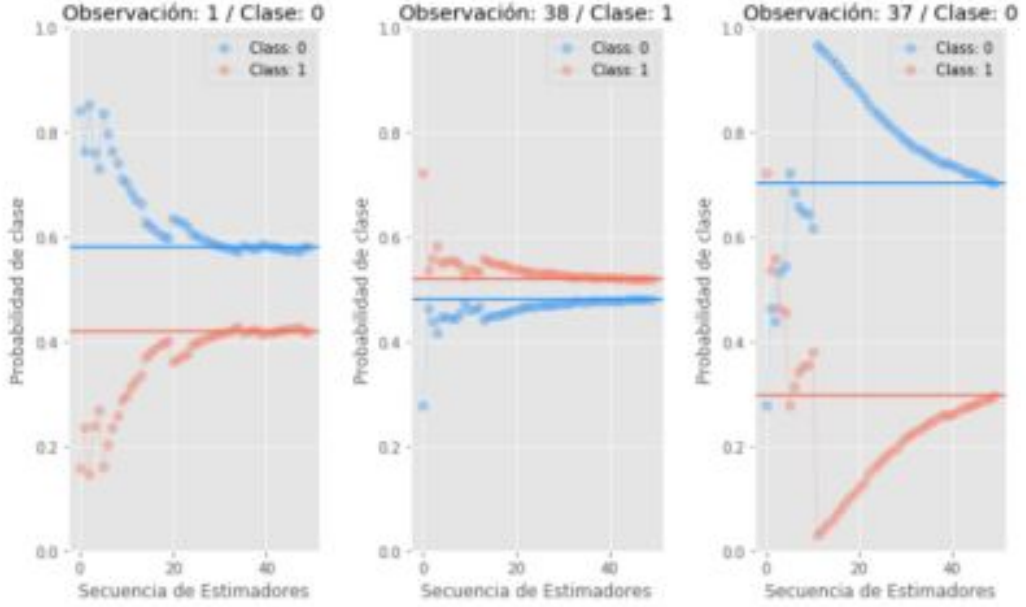
```
In [26]: afx.adaboost_adaptive_behavior()
```



{desafío} latam_

Comportamiento del ponderador

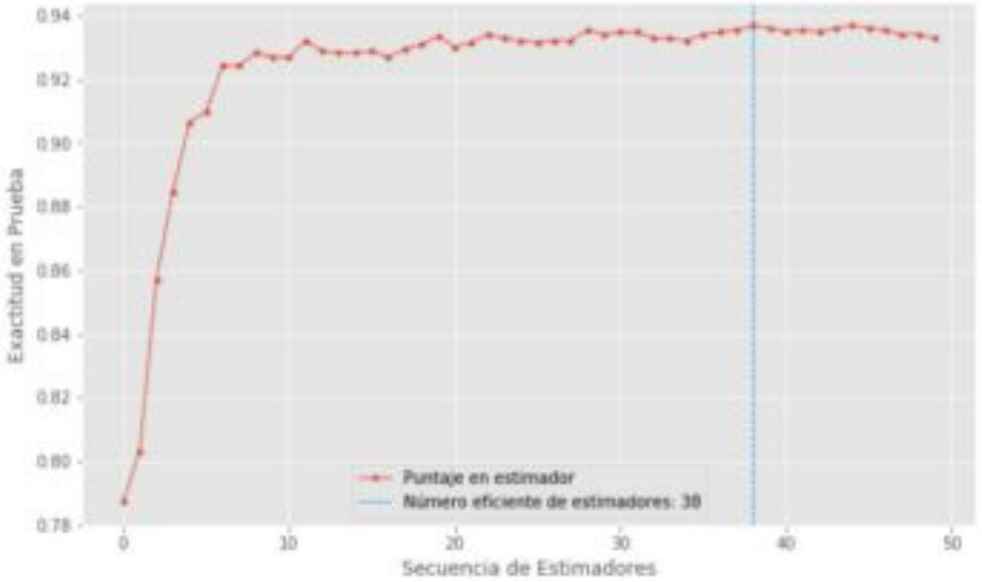
```
In [27]: adaboost_weighting_behavior()
```



{desafío} latam_

Comportamiento en la exactitud general

```
In [28]: adaboost_performance_behavior()
```



Implementación con `sklearn.ensemble.AdaBoostClassifier`

- Generamos el *decision stump*

```
In [29]: decision_stump = DecisionTreeClassifier(max_depth=1, random_state=11238)
```

- Generamos AdaBoost

```
In [30]: adaboost_classifier = AdaBoostClassifier(base_estimator=decision_stump, random_state=1123
         8)
```

- Guardamos predicciones de clase

```
In [31]: adaboost_yhat = adaboost_classifier.fit(X_train, y_train).predict(X_test)
         decision_stump_yhat = decision_stump.fit(X_train, y_train).predict(X_test)
```

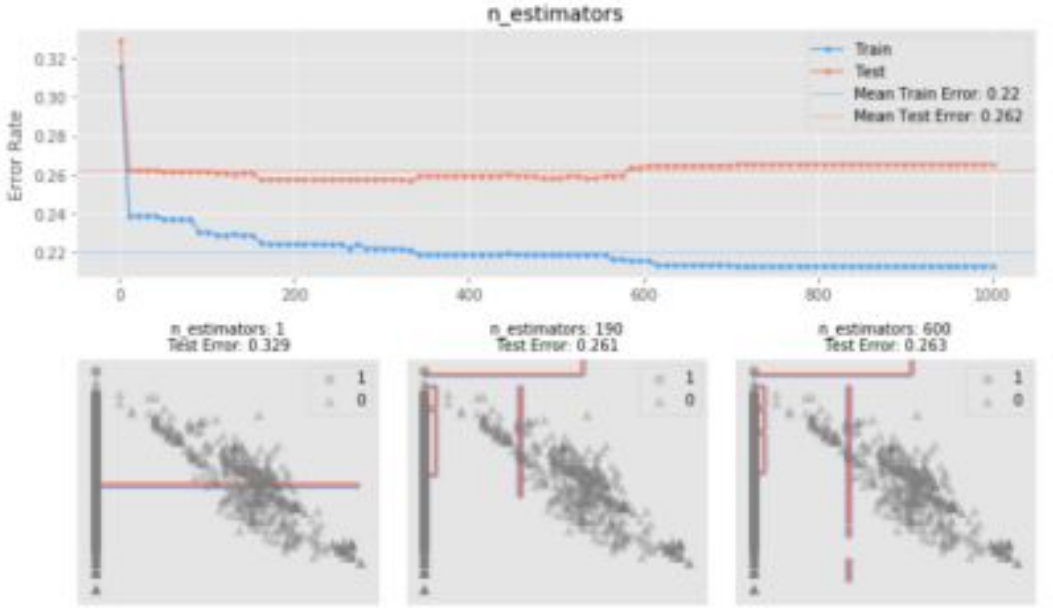
Desempeño de AdaBoost

In [32]:
```
print("Adaboost Performance: \n", get_metrics(y_test, adaboost_y_hat))
print("Decision Stump Performance: \n", get_metrics(y_test, decision_stump_yhat))
```

```
Adaboost Performance:
 [0.933, 0.915]
Decision Stump Performance:
 [0.787, 0.745]
```

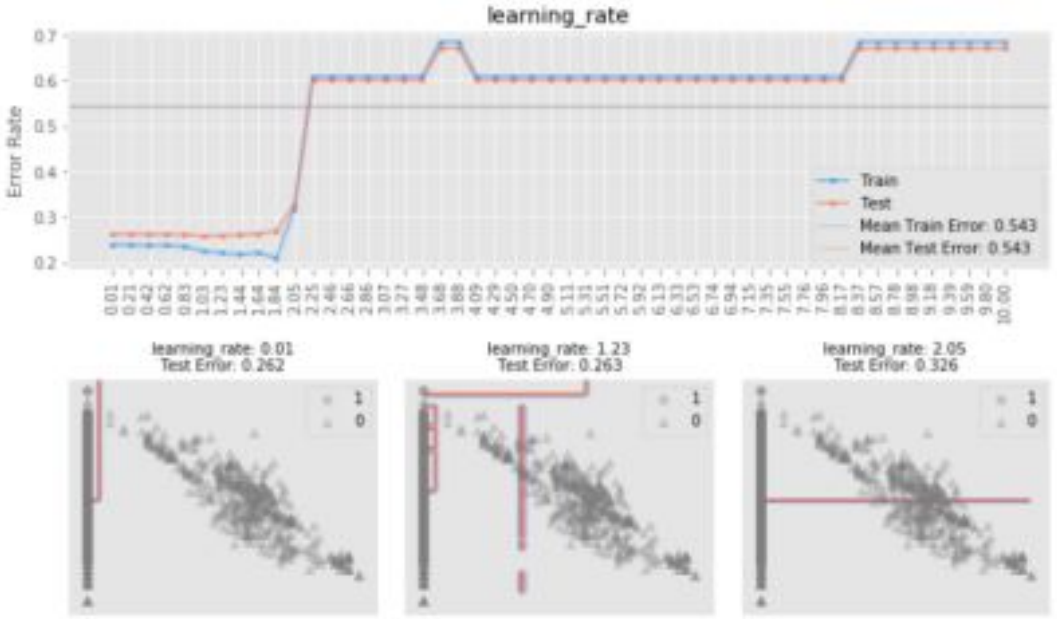
Hiperparámetros: Cantidad de modelos

```
In [33]: afx.n_estimators_hyperparams()
```



Hiperparámetros: Tasa de aprendizaje

```
In [34]: afx.learning_rate_hyperparams()
```



GBoost: Gradient Boosting

Rudimentos I

- Sabemos que AdaBoost se entrena de forma **secuencial** en los errores de clasificación.
- Gradient Boosting se entrena de forma **secuencial** en los errores residuales del modelo previo.
- Debemos optimizar la siguiente función de pérdida:

$$f_0(\mathbf{x}) = \underset{\gamma \in \Gamma}{\operatorname{argmin}} \sum_{i=1}^{N} \ell(y_i, \phi(\mathbf{x}_i, \gamma))$$

Rudimentos II: Funciones de pérdidas

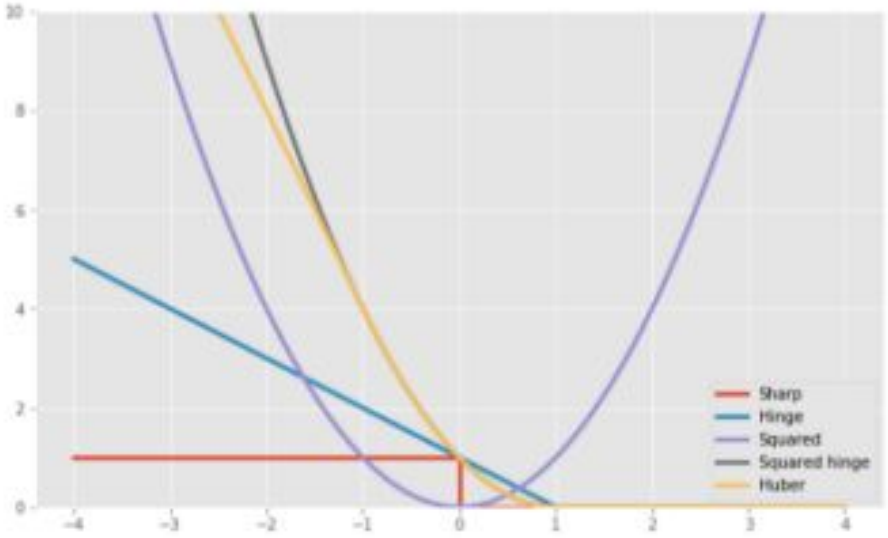
Elección de la función de pérdida = Naturaleza del Vector Objetivo **+** Métrica a minimizar

Existen múltiples variantes (para ver mpas detalle, refiérase a Freund y Schapire, 2012)

Tipo	Función	Algoritmo
Cuadrática	$\frac{1}{2}(y_i, \phi(\mathbf{x}_i, \gamma)^2)$	L2Boosting
Absoluta	$\lvert y_i, \phi(\mathbf{x}_i, \gamma))\rvert$	Gradient Boosting
Exponencial	$\exp(-\tilde{y}_i \phi(\mathbf{x}_i, \gamma))$	AdaBoost
Logloss	$\log(1 + \exp(-\tilde{y}_i \phi_i)$	LogitBoost

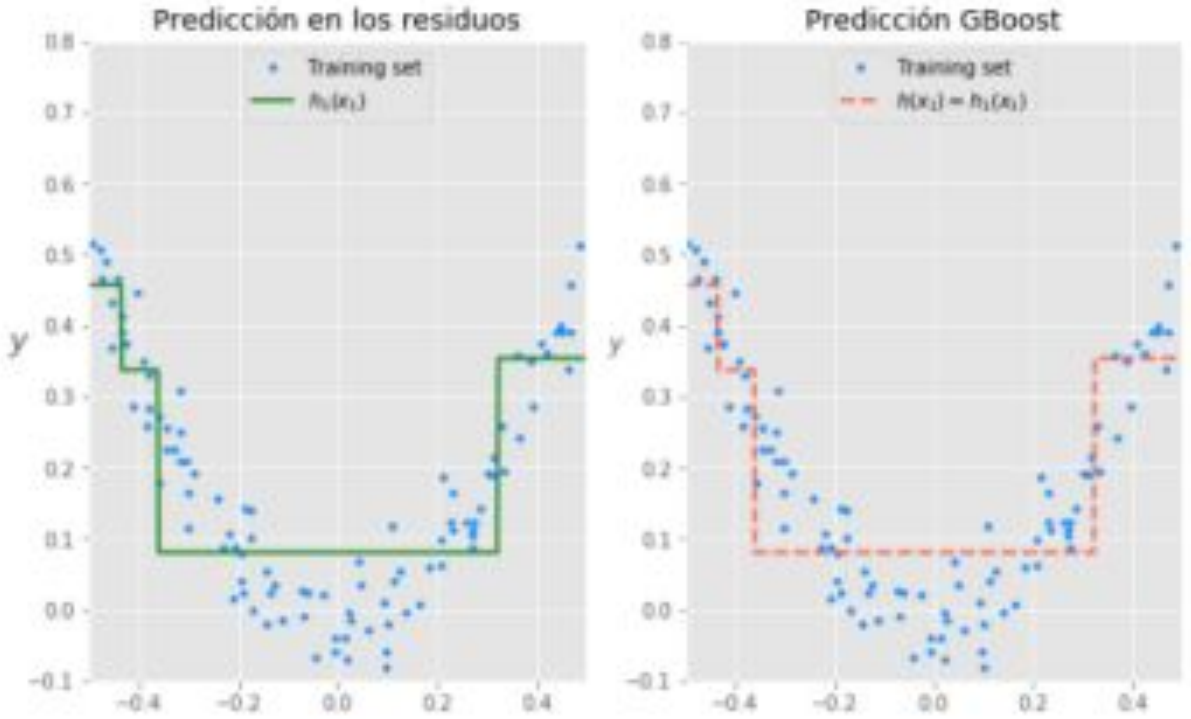
Visualización de la función de pérdida

```
In [35]: afx.loss_functions()
```

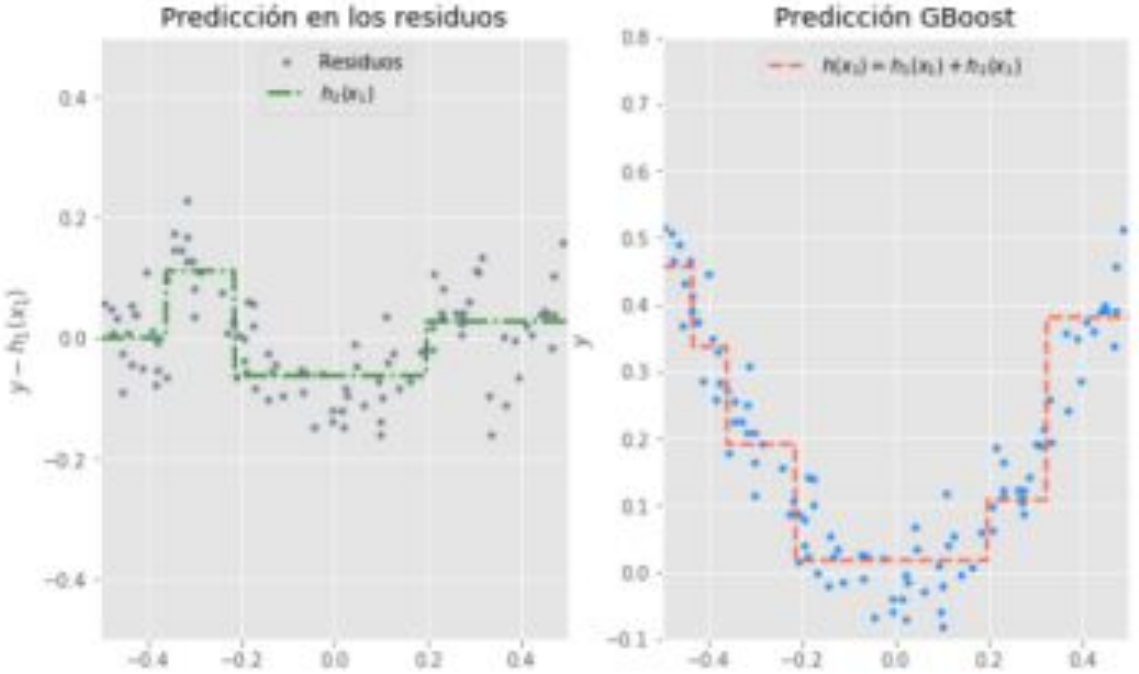


Gradient Boosting
como entrenamiento en los residuos

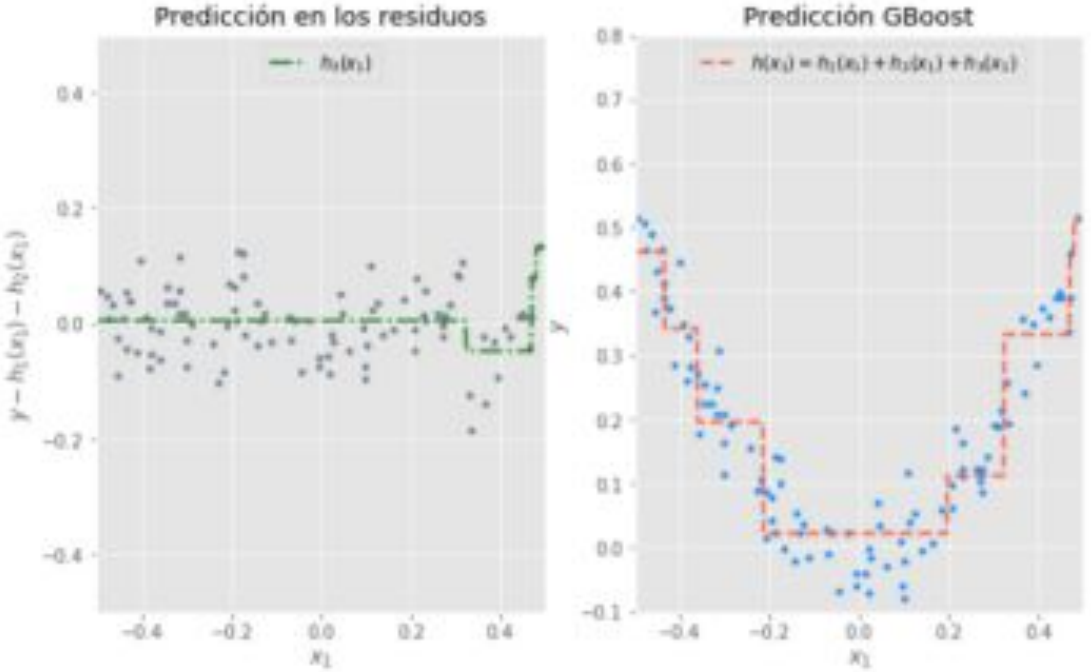
In [36]:
```
afx.gboost_stage_one()
```



In [37]:
```
afx.gboost_stage_two()
```



In [38]:
```
afx.gboost_stage_three()
```



Implementación con `sklearn.ensemble.GradientBoostingClassifier`

```
In [39]: gradient_boost_model = GradientBoostingClassifier().fit(X_train, y_train)

In [40]: gboost_y_hat = gradient_boost_model.predict(X_test)

In [41]: print("Decision Stump Performance: \n", get_metrics(y_test, decision_stump_yhat))
         print("Adaboost Performance: \n", get_metrics(y_test, adaboost_y_hat))
         print("Gradient Boosting Performance: \n", get_metrics(y_test, gboost_y_hat) )

Decision Stump Performance:
 [0.787, 0.745]
Adaboost Performance:
 [0.933, 0.915]
Gradient Boosting Performance:
 [0.949, 0.936]
```

Elementos internos de Gradient Boosting

```
In [42]: gradient_boost_model.estimators_[0][0]
```

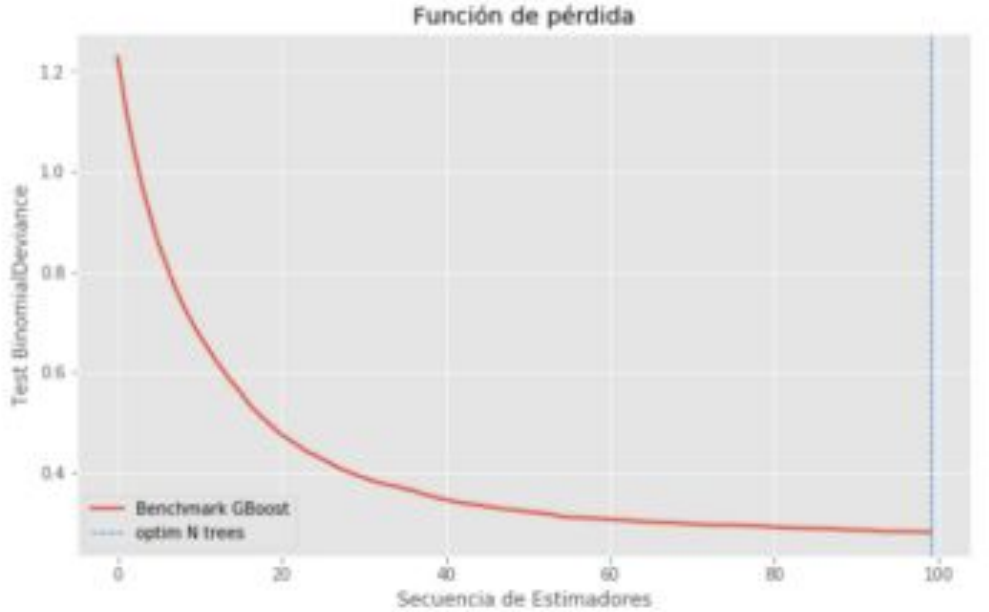
```
Out[42]: DecisionTreeRegressor(criterion='friedman_mse', max_depth=3,
                               max_features=None, max_leaf_nodes=None,
                               min_impurity_decrease=0.0, min_impurity_split=None,
                               min_samples_leaf=1, min_samples_split=2,
                               min_weight_fraction_leaf=0.0, presort='auto',
                               random_state=<mtrand.RandomState object at 0x104cc62d0>,
                               splitter='best')
```

```
In [43]: print(type(gradient_boost_model.loss_))
```

```
<class 'sklearn.ensemble.gradient_boosting.BinomialDeviance'>
```

Comportamiento de la función de pérdida

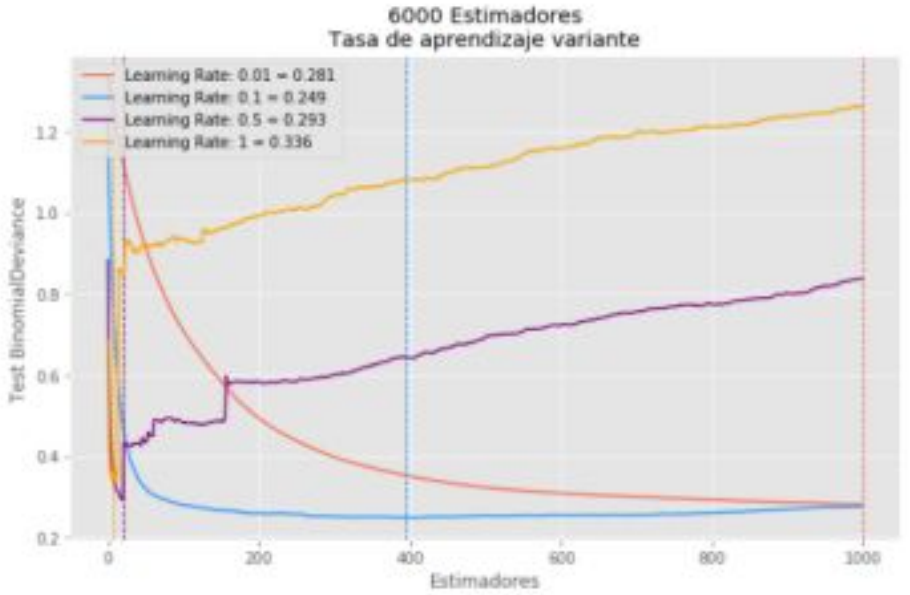
```
In [44]: gboost_loss_function()
```



{desafío} latam_

Hiperparámetro: Tasa de aprendizaje

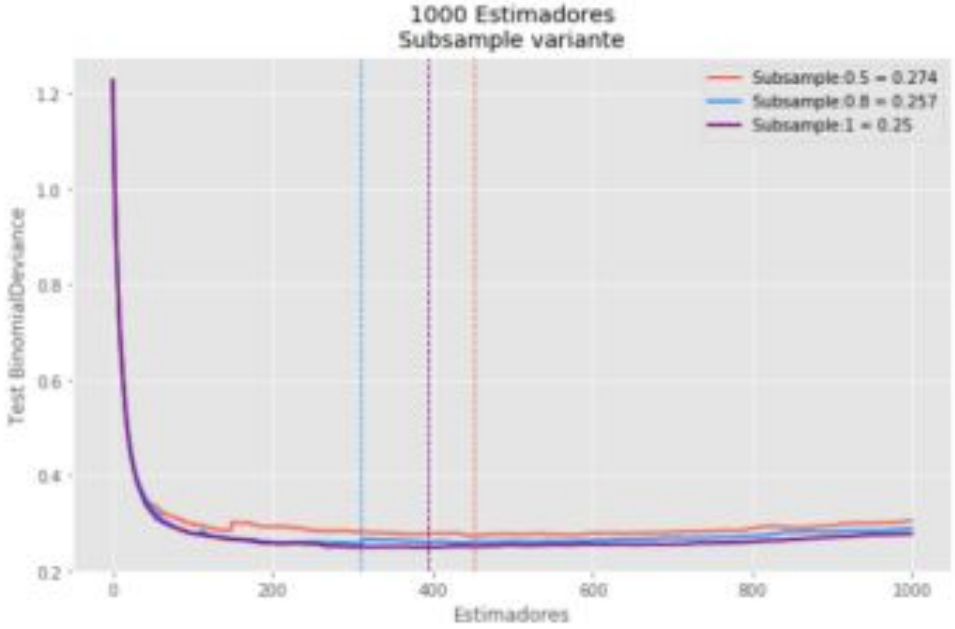
```
In [45]: afx.gboost_learning_hyperparams(X_train, X_test, y_train, y_test)
```



{desafío} latam_

Hiperparámetro: Subsample de entrenamiento

```
In [46]: afx.gboost_sampling_hyperparams(X_train, X_test, y_train, y_test)
```



{desafío} latam_

/* Desafío*/

Panel de discusión

{desafío} latam_

Academia de talentos digitales

www.desafiolatam.com