flujo calibración explicado

**🧪 1. Platt Scaling (CalibratedClassifierCV con method='sigmoid')**

**📚 Librería: sklearn.calibration**

**🔁 Flujo detallado:**

1. **Modelo base entrenado** (clf): Antes de calibrar, se entrena un modelo tradicional (p.ej., RandomForestClassifier) en X\_train, y\_train.
2. **Definición del calibrador**:
3. platt\_clf = CalibratedClassifierCV(base\_estimator=clf, method='sigmoid', cv='prefit')
   * cv='prefit': indica que el clf ya ha sido entrenado.
   * method='sigmoid': aplica un ajuste tipo **sigmoide** como en Platt Scaling clásico.
4. **Ajuste de la calibración**:
5. platt\_clf.fit(X\_val, y\_val)
   * Utiliza las predicciones del clf sobre X\_val, junto con y\_val, para ajustar la función sigmoide que reescala las probabilidades.
6. **Predicción calibrada**:
7. calibrated\_probs = platt\_clf.predict\_proba(X\_test)[:, 1]
   * Devuelve la probabilidad calibrada del modelo base, reescalada por la función ajustada.

**🧾 Resultado: Probabilidades calibradas ∈ [0,1] para cada muestra de test.**

**🧪 2. Isotonic Regression (CalibratedClassifierCV con method='isotonic')**

**📚 Librería: sklearn.calibration**

**🔁 Flujo detallado:**

1. **Definición del calibrador**:
2. isotonic\_clf = CalibratedClassifierCV(base\_estimator=clf, method='isotonic', cv='prefit')
   * method='isotonic': usa una función creciente por tramos (piecewise), no paramétrica.
3. **Entrenamiento**:
4. isotonic\_clf.fit(X\_val, y\_val)
   * Aprende un mapeo creciente entre las probabilidades crudas del modelo base y las verdaderas etiquetas.
5. **Predicción**:
6. calibrated\_probs = isotonic\_clf.predict\_proba(X\_test)[:, 1]

**🧾 Resultado: Probabilidades calibradas, pero más flexibles que el sigmoide (puede sobreajustar si hay pocos datos).**

**🧪 3. Beta Calibration**

**📚 Librería: betacal**

**🔁 Flujo detallado:**

1. **Obtención de las probabilidades del modelo base**:
2. probs\_val = clf.predict\_proba(X\_val)[:, 1]
   * Se extraen las probabilidades crudas.
3. **Inicialización del calibrador**:
4. from betacal import BetaCalibration
5. beta\_clf = BetaCalibration()
6. **Entrenamiento**:
7. beta\_clf.fit(probs\_val, y\_val)
   * Ajusta una transformación basada en la función beta:

f(p)=11+e−(a⋅logit(p)+b⋅logit(1−p)+c)f(p) = \frac{1}{1 + e^{-(a \cdot \text{logit}(p) + b \cdot \text{logit}(1-p) + c)}}

1. **Aplicación a test**:
2. prob\_test = clf.predict\_proba(X\_test)[:, 1]
3. calibrated\_probs = beta\_clf.predict(prob\_test)

**🧾 Resultado: Array 1D de probabilidades calibradas.**

**🧪 4. BBQ (Bayesian Binning into Quantiles)**

**📚 Librería: netcal.binning**

**🔁 Flujo detallado:**

1. **Obtención de probabilidades**:
2. probs\_val = clf.predict\_proba(X\_val)[:, 1]
3. **Inicialización del calibrador**:
4. from netcal.binning import BBQ
5. bbq = BBQ(n\_bins=10)
   * Divide las probabilidades en n\_bins cuantiles.
   * Ajusta distribuciones bayesianas dentro de cada bin.
6. **Entrenamiento**:
7. bbq.fit(probs\_val, y\_val)
   * Ajusta la probabilidad empírica de cada bin y la suaviza bayesianamente.
8. **Aplicación a test**:
9. prob\_test = clf.predict\_proba(X\_test)[:, 1]
10. calibrated\_probs = bbq.transform(prob\_test)

**🧾 Resultado: Probabilidades calibradas con binning y suavizado bayesiano (array 1D).**

**🧪 5. Venn-Abers**

**📚 Librería: venn\_abers**

**🔁 Flujo detallado:**

1. **Inicialización**:
2. from venn\_abers import VennAbersCalibrator
3. va = VennAbersCalibrator()
4. **Entrenamiento**:
5. va.fit(X\_val, y\_val)
   * Se entrena un modelo de predicción interno con los datos de validación.
   * Genera un conjunto de calibradores tipo "Venn predictor" que garantizan intervalos válidos.
6. **Predicción**:
7. prob\_intervals = va.predict\_proba(X\_test)
   * Devuelve una lista de tuplas: [(p0, p1), ...], donde p0 ≤ true prob ≤ p1.
8. **Conversión a punto único (opcional)**:
9. calibrated\_probs = np.mean(prob\_intervals, axis=1)
   * Esto se hace si queremos comparar con calibradores puntuales.

**🧾 Resultado: Intervalos de probabilidad → transformables en probas puntuales si se desea.**

¿Quieres que estos pasos los represente también en un gráfico tipo **diagrama de flujo vertical**, para cada método? Puedo crear uno más extenso y técnico que el anterior, con cada bloque claramente separado.