Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Eléctrica

Tarea 1

Clasificador bayesiano

Integrantes: Lukas Pavez
Profesor: Javier Ruiz del Solar
Auxiliar: Patricio Loncomilla
Ayudantes: Francisco Leiva C.
Gabriel Andrés Azócar Cárcamo
Giovanni Pais L.
Nicolas Cruz Brunet

Fecha de entrega: 14 de Abril de 2019
Santiago, Chile

Índice de Contenidos

Índice de Figuras

1. Introducción

Se tiene como objetivo implementar un clasificador de *air showers*generados por rayos gammas primarios v/s hadrones. Se dispone de un set de datos entregados en la tarea que tiene datos de las características de ambas clases.

Para esto se quiere entrenar un clasificador bayesiano que determine si un vector de características corresponde a un hadrón. Esto se realizará de 2 formas: primero con naive bayes, donde se asume que las características no están correlacionadas, y se utilizaran histogramas para obtener las verisimilitudes de las características. El segundo método se realizara con una gaussiana multidimensional para el calculo de las verosimilitudes.

2. Marco Teórico

Primero se trabajará usando el modelo de histogramas, donde se asume naive bayes, por lo que el calculo de la Ecuación 1 se simplifica, ya que al no estar relacionadas, se puede obtener la verosimilitud como la multiplicación de la probabilidad de cada característica (Ecuación 2), para trabajar con éste método, se toma un set de entrenamiento, se separa por clase, y se generan histogramas por cada característica, como en este caso se tenían 10 características por clase, se calculan 20 histogramas, estos histogramas deben normalizarse, esto es para que cuando se pregunte por una característica, de su probabilidad, así se puede calcular la verosimilitud para cada vector de características del set de prueba.

El clasificador usando el modelo gaussiano, asume que hay relaciones entre las características, por lo ya no se puede usar la Ecuación 2 para calcular la verosimilitud, sino que se tiene que calcular una gaussiana multidimensional para obtenerla, esto se hace con la Ecuación 3.

El set de datos debe separarse en un set de entrenamiento y otro de prueba, esto es porque si se tiene un clasificador y traza una separación entre 2 clases, siempre pueden haber más casos que la separación no toma en cuenta, por esto, se toma una muestra representativa del set de datos y se deja como prueba, con este set de datos se puede calcular diferentes cosas referentes al clasificador, tal como la tasa de verdaderos positivos, la de falsos positivos, la presición, entre otros, y así se puede describir el clasificador que se va a utilizar.

3. Desarrollo

3.1. División de la base de datos

Para dividir la base de datos se tienen 2 funciones: separate_by_class y split_data, la primera función toma los datos y entrega 2 matrices donde cada una representa los datos de una clase, la segunda toma un dataset y entrega un set de entrenamiento y otro de prueba, el código es el siguiente:

```
def separate_by_class(data):
    g = np.array(range(11))  # this is needed for vstack function
    h = np.array(range(11))
    for x in data:
        if x[10] == 1:
            h = np.vstack((h, x))
        else:
            g = np.vstack((g, x))

    g = g[1:]  # delete the first row (range(11))
    h = h[1:]

    return g, h

def split_data(data):
    # g: 0
    # h: 1
    g, h = separate_by_class(data)

    # shuffle data
    np.random.shuffle(g)
    np.random.shuffle(h)

    test_g_len = len(g) * 20 // 100
    test_h_len = len(h) * 20 // 100

    # separate between test set and train set for each class (20 % - 80 %)
    test_g = np.array(g[:test_g_len])
    train_g = np.array(g[test_g_len:])
    test_h = np.array(h[:test_h_len])
    train_h = np.array(h[test_h_len:])

    train_set = np.vstack((train_g, train_h))  # combine the sets of each train set class into one
     ↪ train set
    test_set = np.vstack((test_g, test_h))  # combine the sets of each test set class into one test set
    return test_set, train_set
```

Empezando por la función separate_by_class, se parte creando dos arreglos de tamaño 11, esto es porque se irán agregando filas con la función vstack de numpy, y esta función necesita que todas las filas sean del mismo largo, por lo que no se puede empezar de una fila vacía, luego de agregar cada fila a su arreglo correspondiente (separando por el 11-avo elemento de cada fila) se retorna desde el segundo elemento hasta el final de cada uno de los arreglos (para eliminar la primera fila).

La función split_data parte separando los datos por clase con la función separate_by_class, luego desordena las matrices con np.random.shuffle(matriz), calcula cuanto es el largo del 20 % de cada una, y luego crea una matriz de test y otra de train para cada clase, esto con los largos calculados, donde la matriz de test es desde el primer dato hasta el largo, y la de train es el resto de los datos, luego se juntan las matrices de test de ambas clases en una sola matriz test con la función vstack(), lo mismo se hace con las matrices de train.

Esta forma de dividir los datos es representativa, ya que a cada clase le saco su 20 % por separado y después junto las matrices de ambas clases, por lo que la razón entre (hadrón/gamma) se mantiene del set original a los set de entrenamiento y prueba.

3.2. Modelo naive bayes

Esta parte tiene una función principal llamada naive_bayes que recibe un número de bins un set de prueba, uno de entrenamiento y un set de thetas. Esta función calcula la curva ROC y la deja ploteada (no la muestra, para mostrarla se debe usar plt.show() después de llamar a la función). Para lograr esto, hay otras 2 funciones importantes: histogram(), que recibe un set de datos y un número de bins y entrega una lista de alturas (hist) y una lista de los bordes de los bins (edges), y get_histogram_rates, que recibe un set de prueba, 2 listas de histogramas, 2 listas de bordes de histogramas y un valor de theta y retorna una lista con 2 valores: el false positive rate y el true positive rate. Las funciones son las siguientes:

```
def find_bin(x, bin_edges):
    res = 0
    for edge in bin_edges[1:len(bin_edges) - 1]:
        if x < edge:
            return res
        res += 1
    return res

def histogram(data, bins):
    m = min(data)
    M = max(data)
    size = len(data)
    step = (M - m) / bins
    bin_edges = [m]
    for i in range(1, bins + 1):
        bin_edges.append(bin_edges[i - 1] + step)

    hist = np.zeros(bins)
    for value in data:
        b = find_bin(value, bin_edges)
        hist[b] += 1

    for i in range(len(hist)):
        hist[i] = hist[i] / size
```

```
25      return hist, bin_edges
26
27  def get_histogram_rates(test, hist_g, edges_g, hist_h, edges_h, theta):
28      TP = 0
29      FN = 0
30      FP = 0
31      TN = 0
32
33      for x in test:
34          p0 = p1 = 1
35          for caract in range(10):
36              p0 = p0 * get_prob(hist_g[caract], edges_g[caract], x[caract])
37          for caract in range(10):
38              p1 = p1 * get_prob(hist_h[caract], edges_h[caract], x[caract])
39
40          if p1 / p0 >= theta:  # classifier: hadron
41              if x[10] == 1:  # real: hadron
42                  TP += 1
43              else:  # real: gamma
44                  FP += 1
45          else:  # classifier: gamma
46              if x[10] == 1:  # real: hadron
47                  FN += 1
48              else:  # real: gamma
49                  TN += 1
50
51      TPR = TP / (TP + FN)
52      FPR = FP / (FP + TN)
53      return [FPR, TPR]
54
55  def naive_bayes(bins, test, train, thetas):
56      hist_g = []
57      edges_g = []
58      hist_h = []
59      edges_h = []
60
61      g, h = separate_by_class(train)
62      for i in range(10):
63          hist, edges = histogram(g[:, i], bins)
64          hist_g.append(hist)
65          edges_g.append(edges)
66          hist, edges = histogram(h[:, i], bins)
67          hist_h.append(hist)
68          edges_h.append(edges)
69
70      roc = []
71      for theta in thetas:
72          roc.append(get_histogram_rates(test, hist_g, edges_g, hist_h, edges_h, theta))
73      xs = [x[0] for x in roc]
74      ys = [x[1] for x in roc]
75      plt.plot(xs, ys, label="histogram model, {} bins".format(bins))
```

La función histogram parte por definir el rango de los valores que toma cada bin, esto se hace tomando el mínimo y el máximo del set de datos, restarlos y dividir el resultado por el número de bins, el resultado de la operación es llamado step, para definir los bordes de cada bin, basta con tomar el mínimo de los datos e ir sumando el step calculado hasta llegar al máximo.
Para calcular las alturas, se parte con una lista de 0 para todos los bins, para cada dato se busca el índice del bin en el que cae con la función find_bin, que va recorriendo la lista de bordes hasta que el dato es mayor que el borde, luego de encontrar el bin en el que cae el dato se le suma 1 a su altura.
Finalmente, cada altura de la lista de alturas se divide por la cantidad de datos para normalizar el histograma.

La función get_histogram_rates parte seteando los True Positive (TP), False Negative (FN), False Positive (FP), True Negative (TN) en 0, luego para cada dato de el set de prueba hace lo siguiente: inicia las verosimilitudes (p1 y p0) en 1, luego para cada característica, multiplica la verosimilitud por la probabilidad de que esa característica sea gamma (p0) o hadrón (p1), esa probabilidad se obtiene con la función get_prob que se le da el histograma (altura y bordes) de la característica y el dato que se quiere buscar, y la función busca en el histograma el valor, la función verifica que si la probabilidad da 0 o el dato se sale del rango del histograma (dato menor que el menor de los bordes o mayor que el mayor borde), se le asigna la probabilidad 0.00000000000001, esto es para evitar divisiones por 0, y se asume que si un dato no esta en el histograma su probabilidad es 0.
Luego de calcular las verosimilitud, se hace la comparación $\frac{p1}{p0} \geq \theta$, si es verdadero, entonces se pregunta si el dato es en realidad de la clase hadrón, si es así se le suma 1 a TP, si no, se le suma 1 a FP. En el caso de la desigualdad sea falsa, se pregunta si el dato es en realidad de la clase hadron, si es así se le suma 1 a FN, si no, se le suma 1 a TN.
Luego de haber calculado los valores de TP, FP, TN, FN con todos los valores del set de prueba, se calcula el True Positive Rate (TPR) y False Positive Rate (FPR) con las fórmulas $TPR = TP/(TP + FN)$ y $FPR = FP/(FP + TN)$. Finalmente se retornan ambos valores.

Finalmente, la función naive_bayes, parte creando los histogramas, para esto separa el set de entrenamiento por clase, y luego para cada característica calcula los histogramas para ambos sets, por lo que al terminar de calcular se tienen 4 listas de largo 10: 2 listas de las alturas de los histogramas y 2 listas de los bordes de los bins de cada histograma.
Después de calcular todos los histogramas necesarios, comienza el cálculo de la curva roc, para esto recorre la lista de thetas, y para cada uno se llama a la función get_histogram_rates dandole el set de prueba, las listas de los histogramas y un theta, eso entrega el FPR y TPR, después se grafican los puntos.

3.2.1. Cambio en la cantidad de bins

Se hicieron pruebas para 100 thetas y distintas cantidades de bins, y se vio que para 60 bins se tiene una curva aceptable, viendo la Figura 1, aquí se tiene que las curvas más aceptables son las de 60, 90 y 100 bins, luego en la Figura 2, las 3 son muy parecidas, ya que las de 60 y 90 son casi iguales, y ambas con la de 100 hay un rango en que las de 60 y 90 son mejor (esto para thetas más estrictos), después la de 100 pasa a ser mejor.

3.3. Modelo gaussiano

Esta parte tiene una función principal llamada gauss_bayes que recibe un set de prueba, uno de entrenamiento y un set de thetas. Esta función calcula la curva ROC y la deja ploteada (no la muestra, para mostrarla se debe usar plt.show() después de llamar a la función). Para lograr esto, utiliza otra función importantes (además de las de numpy): get_gaussian_rates, que recibe un set de prueba, 2 listas de promedios, 2 matrices de covarianza y un valor de theta y retorna una lista con 2 valores: el false positive rate y el true positive rate. Las funciones son las siguientes:

```
def gaussian(x, mu, cov):
    pi = math.pi
    k = len(x)
    diff = x - mu
    inv = np.linalg.inv(cov)
    det = np.linalg.det(cov)
    exp = -1 * 0.5 * ((diff.transpose()).dot(inv)).dot(diff)
    den = math.sqrt(math.pow(2 * pi, k) * det)
    return math.exp(exp) / den

def get_parameters(data):
    mu = np.array([])
    for i in range(10):
        mu = np.hstack((mu, np.mean(data[:, i])))
    cov = np.cov(data, rowvar=False)
    return mu, cov

def get_gaussian_rates(test, mu_g, cov_g, mu_h, cov_h, theta):
    TP = 0
    FN = 0
    FP = 0
    TN = 0

    for x in test:
        p0 = gaussian(x[:10], mu_g, cov_g)
        p1 = gaussian(x[:10], mu_h, cov_h)
        if p0 == 0: p0 = 0.00000000000001
        if p1 == 0: p1 = 0.00000000000001
        if p1 / p0 >= theta:  # classifier: hadron
            if x[10] == 1:  # real: hadron
                TP += 1
            else:  # real: gamma
                FP += 1
        else:  # classifier: gamma
            if x[10] == 1:  # real: hadron
                FN += 1
            else:  # real: gamma
                TN += 1

    TPR = TP / (TP + FN)
    FPR = FP / (FP + TN)
    return [FPR, TPR]
```

```
43
44 def gauss_bayes(test, train, thetas):
45     g, h = separate_by_class(train)
46
47     mu_h, cov_h = get_parameters(h[:, :10])  # delete last column
48     mu_g, cov_g = get_parameters(g[:, :10])
49     roc = []
50     for theta in thetas:
51         roc.append(get_gaussian_rates(test, mu_g, cov_g, mu_h, cov_h, theta))
52     xs = [x[0] for x in roc]
53     ys = [x[1] for x in roc]
54     plt.plot(xs, ys, label="gauss model")
```

La función get_gaussian_rates parte seteando los True Positive (TP), False Negative (FN), False Positive (FP), True Negative (TN) en 0, luego calcula las verosimilitudes p0 y p1 con la función gaussian, que recibe un vector de características x, un vector de promedios mu y una matriz de covarianza, y entrega la verosimilitud calculada con la fórmula de la gaussiana multidimensional (Ecuación 3).
Luego de calcular las verosimilitud, se hace la comparación $\frac{p1}{p0} \geq \theta$, si es verdadero, entonces se pregunta si el dato es en realidad de la clase hadrón, si es así se le suma 1 a TP, si no, se le suma 1 a FP. En el caso de la desigualdad sea falsa, se pregunta si el dato es en realidad de la clase hadron, si es así se le suma 1 a FN, si no, se le suma 1 a TN.
Luego de haber calculado los valores de TP, FP, TN, FN con todos los valores del set de prueba, se calcula el True Positive Rate (TPR) y False Positive Rate (FPR) con las fórmulas $TPR = TP/(TP + FN)$ y $FPR = FP/(FP + TN)$. Finalmente se retornan ambos valores.

La función gauss_bayes parte por obtener el vector de promedios y la matriz de covarianza para cada clase, para esto, parte separando por clase el set de entrenamiento, y luego llama a la función get_parameters, esta función calcula el promedio por columna con np.mean() y los guarda en una lista, también calcula la matriz de covarianza con np.cov(), teniendo en cuenta de hay que dar False en el parámetro rowbar, esto es porque np.cov toma en cuenta que cada fila de la matriz entregada es una característica diferente, y cada columna es una observación, per en el caso de esta tarea, se trabaja con que las columnas representan una característica diferente y cada fila es una observación, y el parámetro rowbar arregla eso, otra forma de arreglarlo era darle la matriz traspuesta a np.cov(). La función se llama 2 veces, una vez dandole el set de entrenamiento de la clase hadrón y la otra de la clase gamma, con esto se tienen 2 listas de 10 promedios y 2 matrices de covarianza.
Después de obtener los parámetros necesarios, comienza el cálculo de la curva roc, para esto recorre la lista de thetas, y para cada uno se llama a la función get_gaussian_rates dandole el set de prueba, las listas de los promedios, las matrices de covarianza y un theta, eso entrega el FPR y TPR, después se grafican los puntos.

3.4. Comparación de ambos modelos

Se setearon 60 bins para trabajar con el modelo de histogramas, esto ya que en la comparación de bins la roc con 60 bins estaba entre las mejores junto con la de 100, pero se eligió la de 60 ya que con menos bins aportaba a la velocidad de la creación de los histogramas. Se hicieron curvas según el número de thetas, se trabajó principalmente en el rango de 0 a 100, y se notó que la mayoría de los puntos se juntaban en la parte izquierda del gráfico, como se puede ver en la Figura 3, hasta que se comenzó a agregar puntos entre 0 y 1, después de eso las curvas comenzaron a parecerse a la Figura 4, esto es ya que los thetas con los que se probaba eran demasiado estrictos, lo que hacía que ambos rates fueran muy bajos, al agregar valores menos estrictos (entre 0 y 1) arregló el problema.

Se hicieron distintas pruebas con thetas entre 1 y 100 variando el número de thetas que hay entre 0 y 1, en todas ambas curvas salían bastante juntas, pero siempre la curva del modelo gaussiano estaba por encima de la curva del método de histogramas, se puede ver esto en las Figuras 5, 6 y 7. Esto significa que asumir que las características no están relacionadas entre ellas no es correcta.

Sobre cada método, empezando con naive_bayes, al asumir que las características no están relacionadas hace que el calculo de las verosimilitudes sea más rápido, ya que se ahorra gran parte del calculo, donde solo debe multiplicarse la probabilidad dada por los histogramas, mientras que con el modelo gaussiano, al tener que multiplicar matrices el proceso se hace mucho más lento.

4. Conclusiones

En esta tarea se trabajó con un clasificador bayesiano, donde primero se asumió naive bayes, donde las características no están relacionadas, y luego se utilizó el modelo con gaussiana multidimensional. En los resultados se pudo ver que en este caso el modelo gaussiano predice de mejor manera que el modelo de histogramas, esto puede ser porque hay características que dependen de otras. También se aprendió que al calcular las curvas roc puede tomar bastante tiempo dependiendo de cuantos valores de θ se usen para la curva y que modelo se está usando.
Los resultados obtenidos eran los esperados, ya que para un dataset como el entregado se espera que haya alguna relación entre las características, por lo tanto el modelo gaussiano funcionaría mejor que asumir naive bayes.

5. Anexo

5.1. Ecuaciones

$$\frac{P(características|hadrón)}{P(características|no\_hadrón)} \geq \theta \tag{1}$$

$$P(características|hadrón) = P(x_1|hadrón) * ... * P(x_n|hadrón) \tag{2}$$

$$f(\vec{x}) = \frac{\exp\left\{-\frac{1}{2} * (x-\mu)^T \Sigma^{-1} (x-\mu)\right\}}{\sqrt{(2\pi)^k |\Sigma|}} \tag{3}$$

5.2. Figuras

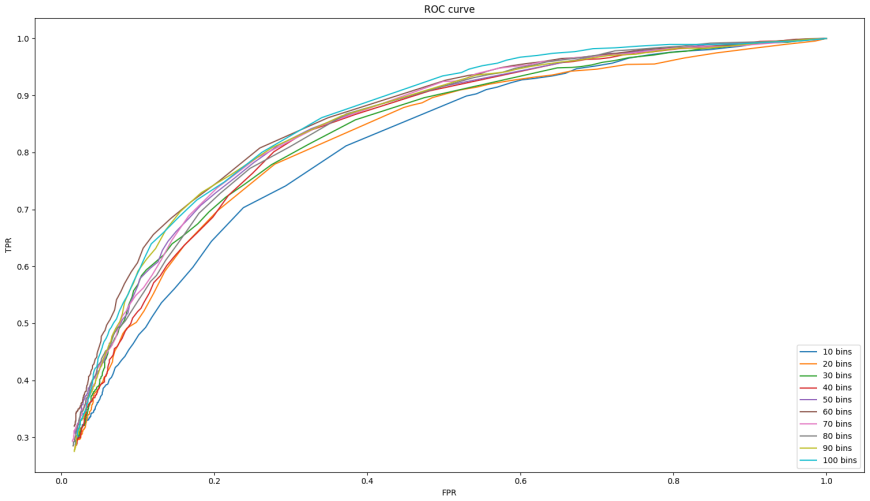

Figura 1: comparación de del número de bins

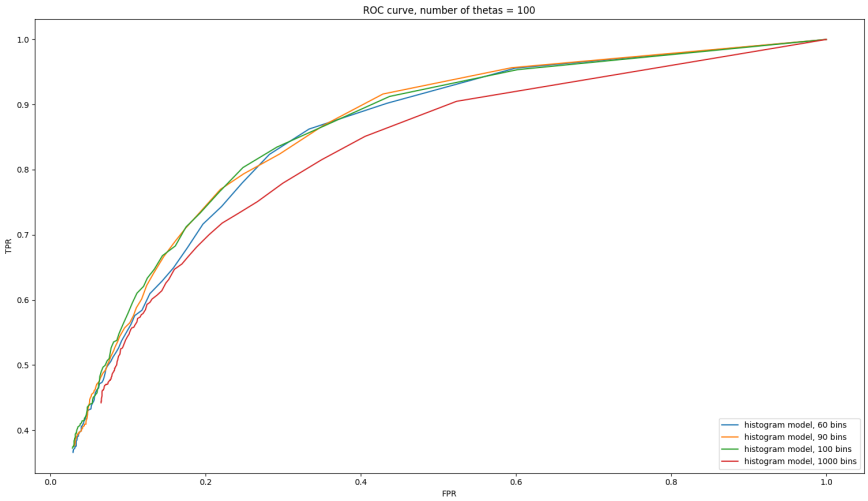

Figura 2: comparación de del número de bins

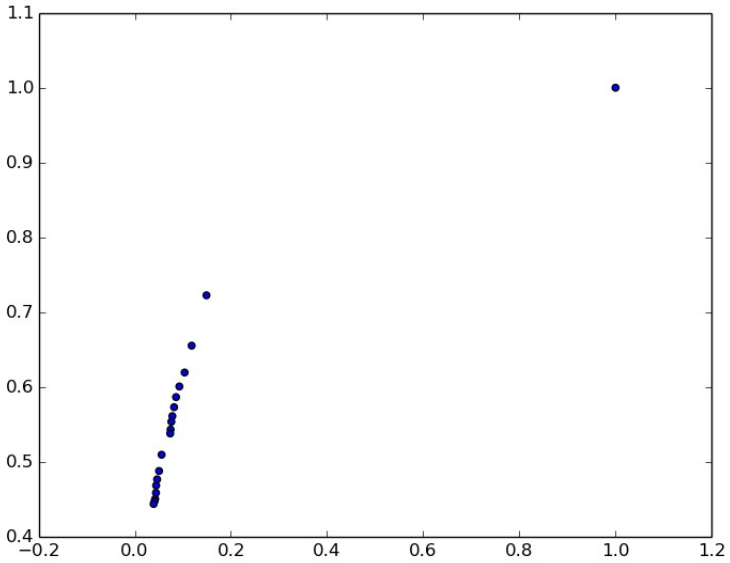

Figura 3: roc con puntos agrupados a la izquierda

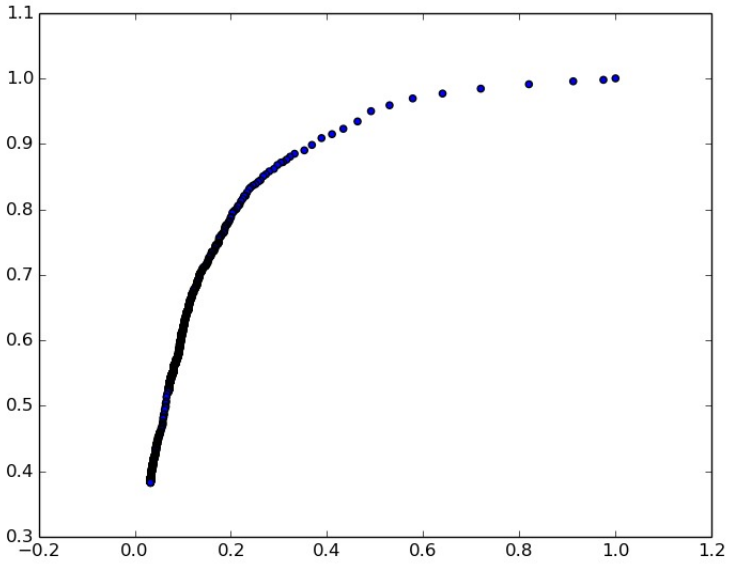

Figura 4: roc normal

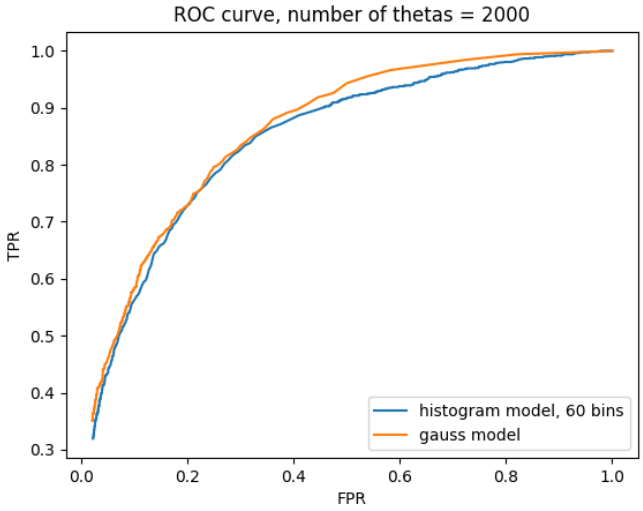

Figura 5: comparación de modelos, 2000 thetas

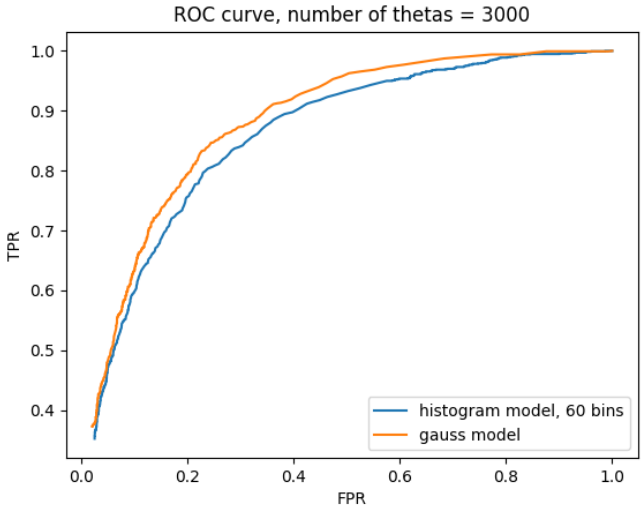

Figura 6: comparación de modelos, 3000 thetas

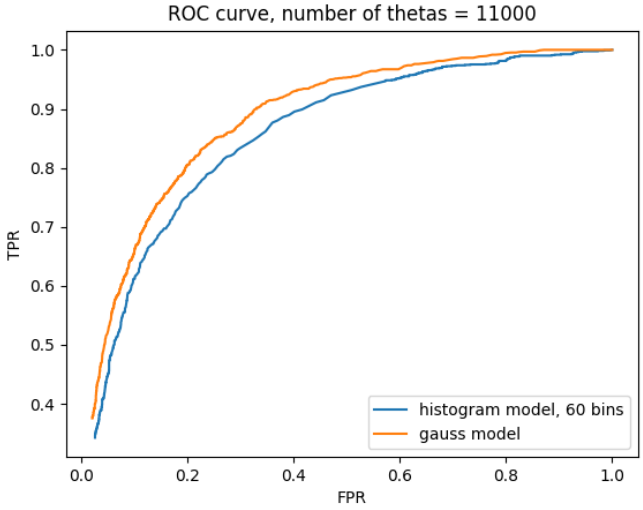

Figura 7: comparación de modelos, 11000 thetas