CEFET/RJ
Programa de Pós-graduação em Ciência da Computação
CIC1215 - Aprendizado de Máquina - Trabalho 02

Prof. Eduardo Bezerra (ebezerra@cefet-rj.br)

7/2018

Conteúdo

1 Parte 1 - Agrupamento

Neste exercício, você implementará o algoritmo K-means. Você irá executar esse algoritmo sobre um conjunto de dados 2D que ajudará você a obter uma intuição de como o K-means funciona.

1.1 Implementando K-means

O algoritmo K-means é um método para agrupar automaticamente exemplos de dados semelhantes. Concretamente, o algoritmo recebe um conjunto de treinamento $\{x^{(1)}, \ldots, x^{(m)}\}$ (onde $x^{(i)} \in \Re^n$) e agrupa os dados em K grupos (*clusters*) coesos. A intuição por trás do K-means é um procedimento iterativo que começa por definir os centroides iniciais de cada grupo e, em seguida, refina essa solução atribuindo repetidamente exemplos a seus centróides mais próximos e, em seguida, recalculando os centróides com base nas atribuições. O algoritmo K-means é o seguinte:

Listing 1: Função `run_kmeans`.

```
# Inicia os centroides
centroids = kmeans_init_centroids(X, K)

for iter in range(max_iters):
        # Passo 1 (atribuição a grupos): associar cada ponto de dado ao centroide mais
            ↪ próximo
        # idx(i) corresponde ao índice do centroide associado ao exemplo i
        idx = find_closest_centroids(X, centroids)
        # Passo 2 (atualização dos centroides): Computar médias com base nas atribuições
            ↪ a centróides
        centroids = compute_means(X, idx, K)
end
```

O laço interno do algoritmo executa repetidamente duas etapas: (i) associação de cada exemplo de treinamento $x^{(i)}$ ao centróide mais próximo, e (ii) Atualização da média de cada centróide usando os pontos atualmente atribuídos a ele. O K-means sempre convergirá para algum conjunto final de médias para os centróides. Observe que a solução final pode nem sempre ser ideal e depende da configuração inicial dos centróides. Portanto, na prática, o algoritmo K-means é executado algumas vezes com diferentes inicializações aleatórias. Uma maneira de escolher entre estas diferentes soluções a partir de diferentes inicializações aleatórias é escolher aquele com o menor valor de função de custo (distorção). Você implementará as duas fases do algoritmo K-means separadamente nas próximas seções.

1.2 Encontrando centróides mais próximos

No passo de *atribuição de grupos* do K-means, o algoritmo atribui todos os exemplos de treinamento $x^{(i)}$ ao seu centróide mais próximo, dadas as posições correntes dos centróides. Especificamente, para cada exemplo $x^{(i)}$, definimos

$$c^{(i)} \leftarrow j \text{ que minimiza } \left\| x^{(i)} - \mu_j \right\|$$

onde $c^{(i)}$ é o índice do centróide mais próximo de $x^{(i)}$, e μ_j é o posição (valor) do j-ésimo centróide. Note que $c^{(i)}$ corresponde a `idx(i)` no trecho de código acima. Sua tarefa é completar o código da função `find_closest_centroids`. Esta função recebe a matriz de dados X e as localizações de todos os centróides (argumento `centroids`) e deve produzir uma matriz unidimensional `idx` em que a i-ésima entrada contém o índice (um valor em $\{0, \ldots, K-1\}$, onde K é o número total de centróides) do centróide mais próximo ao exemplo correspondente do conjunto de treinamento. Você pode completar a implementação dessa função usando um loop sobre todos os exemplos de treinamento e todo centróide, embora haja formas mais eficientes de implementação.

Depois de ter concluído o código em `find_closest_centroids`, execute script `Parte1/main.py` para produzir a saída para os três primeiros exemplos. Você deverá ver a saída [0 2 1], correspondendo às atribuições aos grupos para os 3 primeiros exemplos. Além disso, a função `main` deve produzir o gráfico da Figura 1, que apresenta a atribuição inicial de pontos a grupos.

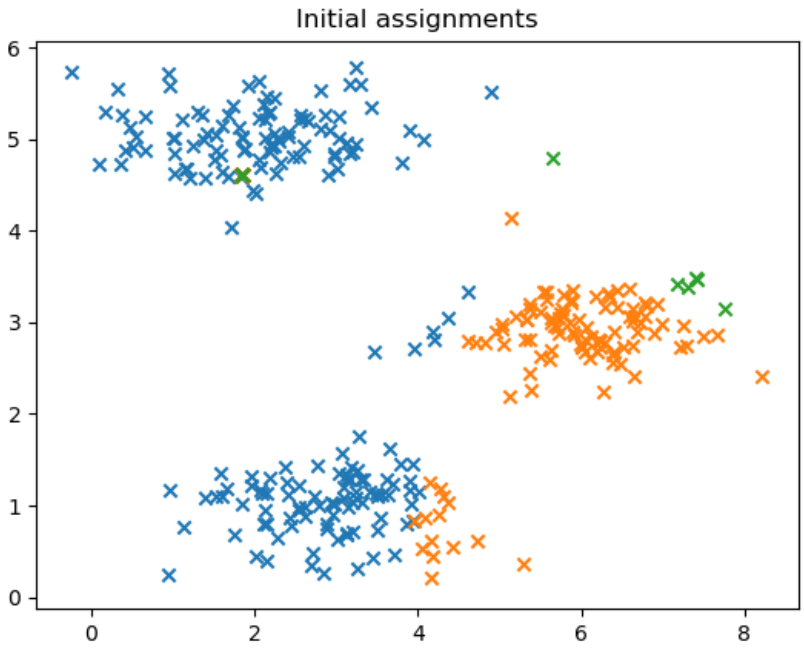

Figura 1: Atribuições iniciais de pontos a grupos.

1.3 Atualização dos centróides

Dadas as atribuições de cada ponto a um centróide, a segunda fase do algoritmo recomputa, para cada centróide, a média dos pontos que foram atribuídos a ele. Especificamente, para cada centroide k, definimos

$$\mu_k \leftarrow \frac{1}{|C_k|} \sum_{i \in C_k} x^{(i)}$$

onde C_k é o conjunto de exemplos que são atribuídos ao centróide k. Concretamente, se dois exemplos (digamos) $x^{(3)}$ e $x^{(5)}$ são atribuídos ao centróide $k = 2$, então você deve atualizar $\mu_2 = \frac{1}{2}(x^{(3)} + x^{(5)})$.

Agora você deve completar o código da função `compute_centroids`. Você pode implementar esta função usando um loop sobre os centróides. Você também pode usar um loop sobre os exemplos. Entretanto, se você realizar uma implementação vetorizada, seu código pode ser executado mais rápido.

Depois de ter concluído a implementação de `compute_centroids`, a execução do script `Parte1/main.py` irá gerar os centroides resultantes após 10 iterações do K-means.

1.4 K-means aplicado ao conjunto de dados de exemplo

Após ter concluído as implementações das funções `find_closest_centroids` e `compute_centroids`, sua próxima tarefa é executar o algoritmo K-means em um conjunto de dados 2D para ajudá-lo a entender como o K-means funciona. Suas funções são chamadas dentro da função `run_kmeans` que por sua vez é chamada na função `main`. Você deve analisar a função `run_kmeans` para entender como ela funciona. Observe que o código chama as duas funções que você implementou em um loop. Além disso, se você definir a variável `plot_progress` como `True`, essa função irá apresentar graficamente a evolução das configurações dos centróides. Nesse caso, essa função produzirá uma sequência de visualizações que ilustra o progresso do algoritmo em cada iteração. Perceba como cada etapa do K-means altera os centróides e as atribuições de pontos aos grupos. Ao final da 10 iterações, a visualização deve ser similar àquela exibida na Figura 2.

1.5 Inicialização aleatória

As atribuição inicial de centróides para o conjunto de dados de exemplo feita na primeira parte da função `main` foi projetada de tal forma que você veja a mesma convergência toda vez que executa o script. Em casos práticos, uma boa estratégia para inicializar os centróides é selecionar exemplos aleatórios do conjunto de treinamento. Nesta parte do exercício, você deve utilizar a função já fornecida `kmeans_init_centroids`:

Listing 2: Função `kmeans_init_centroids`.

```
def kmeans_init_centroids(X, K):
    return X[np.random.choice(X.shape[0], K, replace=False)]
```

O código acima primeiro aleatoriamente seleciona k índices dos exemplos (usando a função `np.random.choice`). Em seguida, seleciona K exemplos baseados na seleção aleatória dos índices. Repare que essa implementação permite que os exemplos sejam selecionados aleatoriamente sem o risco de selecionar o mesmo exemplo mais de uma vez.

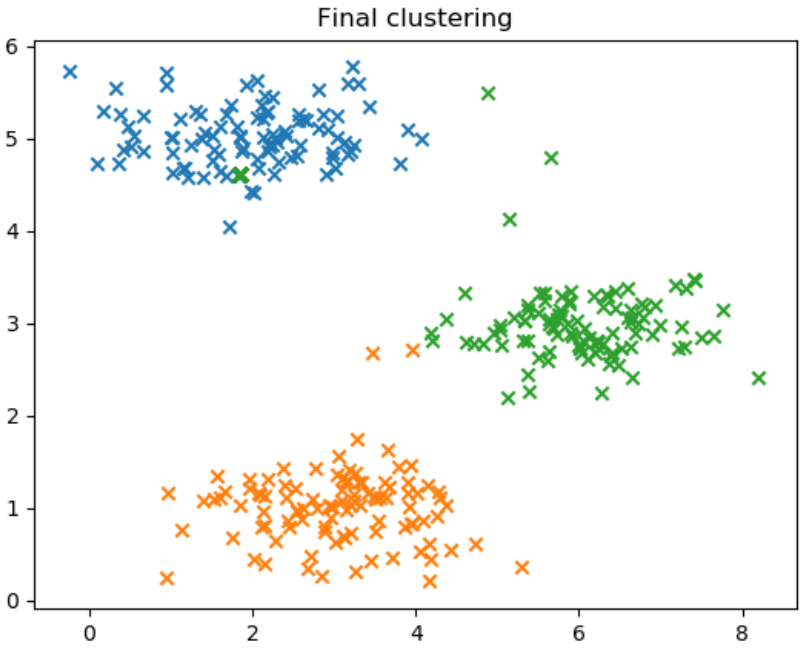

Figura 2: Grupos detectados pelo K-means após 10 iterações

2 Parte 2 - Redução de Dimensionalidade

Nesta parte, você usará a análise de componentes principais (PCA) para executar redução de dimensionalidade. Você primeiro irá realizar experimentos com um conjunto de dados em 2D para obter intuição sobre como o PCA funciona.

2.1 Conjunto de dados de exemplo

Para ajudá-lo a entender como o PCA funciona, você irá aplicá-lo a um conjunto de dados 2D que tem uma direção de grande variação e uma de menor variação. O script `Parte2/main.py` plotará os dados de treinamento (Figura 3). Nesta parte do exercício, você visualizará o que acontece quando você usa PCA para reduzir o dados de 2D para 1D. Em problemas reais, você pode se deparar com um conjunto de dados de 256 dimensões e desejar reduzir para 50 dimensões, digamos. Entretanto, usar dados dimensionais menores neste exemplo nos permite visualizar o efeito da execução do algoritmo.

2.2 Implementando o PCA

Antes de usar o PCA, é importante primeiro normalizar os dados subtraindo o valor médio de cada característica (*feature*) do conjunto de dados e dimensionar cada uma delas para que estejam no mesmo intervalo. No script fornecido `Parte2/main.py`, esta normalização foi realizada para você usando a função `normalize_features`.

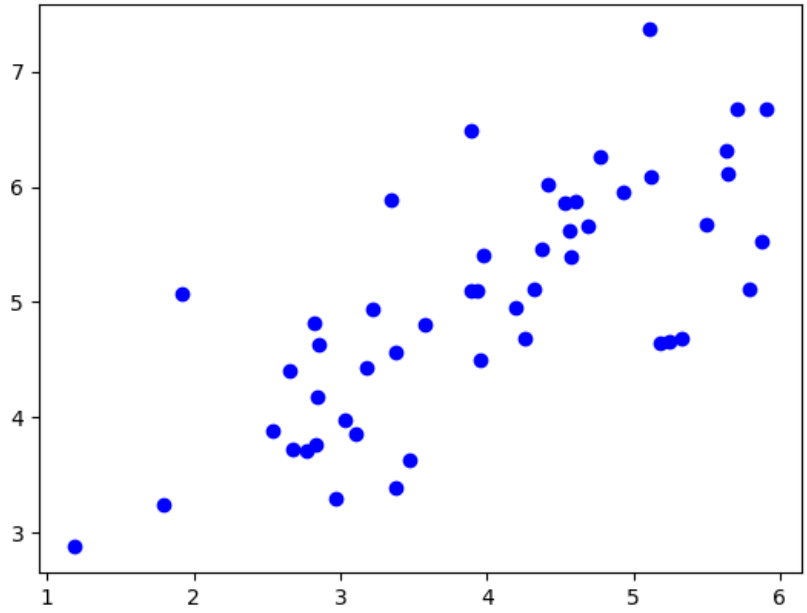

Figura 3: Conjunto de dados original.

Após a normalização dos dados, o PCA pode ser executado para calcular os componentes principais. Nesta parte do trabalho, você irá completar a implementação do PCA fornecida em `Parte2/main.py`. Em particular, você deve implementar a função `pca` contida neste arquivo.

O PCA consiste de duas etapas computacionais. Primeiro, a matriz de covariância dos dados deve ser calculada. A matriz de covariância é dada por:

$$\Sigma = \frac{1}{m} X X^T$$

onde X é a matriz de dados, com exemplos nas linhas, e m é o número de exemplos. Note que Σ é uma matriz $n \times n$ e não o operador de somatório.

Depois de calcular a matriz de covariância, você pode decompor essa matriz por meio da técnica denominada *decomposição por valores singulares* (SVD) para computar os componentes principais. Em Python, você pode usar o numpy: `U, S, V = np.linalg.svd(Sigma)`, onde `U` irá conter os componentes principais e `S` é uma matriz diagonal. As colunas da matriz U contêm os autovetores $U_1, U_2, \ldots, U_n$. Estes corresponderão aos componentes principais de variação nos dados.

Depois de concluir a implementação da função `pca`, você deve analisar o restante do script `Parte2/main.py` para entender a ocorre a aplicação do PCA sobre o conjunto de dados de exemplo e a apresentação dos dados projetados em 1D. A próxima seção irá guiar você nessa tarefa.

2.3 Redução de Dimensionalidade com PCA

Depois de computar os componentes principais, você pode usá-los para reduzir a dimensão do conjunto de dados, projetando cada exemplo para um espaço de dimensão menor, $x^{(i)} \rightarrow z^{(i)}$ (por exemplo, projetando os dados de 2D para 1D). Nesta parte do exercício, você usará os autovetores retornados pelo PCA para projetar o conjunto de dados em um espaço unidimensional. Na prática, se você estivesse usando um algoritmo de aprendizado como a regressão linear ou talvez redes neurais, agora você poderia usar os dados projetados no lugar dos dados originais. Usando os dados projetados, você pode treinar seu modelo mais rápido, pois há menos dimensões na entrada.

2.3.1 Projetando os dados nos componentes principais

Agora você deve analisar o funcionamento da função `project_data`. Especificamente, essa função recebe um conjunto de dados X, os componentes principais U, e o número desejado de dimensões para reduzir, K. A função projeta cada exemplo em X para os K componentes principais em U. Note que os K componentes principais em U são dados pelas primeiras K colunas de U: `U_reduce = U[:, 0:K]`.

Depois de ter analisado e entendido o código em `project_data`, você deve projetar o primeiro exemplo de X na primeira dimensão. Ao fazer isso, você deve ver um valor de aproximadamente 1.481 (ou possivelmente -1.481, se você tiver $-U_1$ em vez de U_1). Forneça o trecho de código que você usou para isso em seu relatório.

2.3.2 Reconstruindo uma aproximação dos dados

Depois de projetar os dados no espaço de menor dimensão, é possível recuperar os dados, projetando-os de volta para o espaço original. Analise o código da função `recover_data` que projeta cada exemplo em Z de volta ao espaço original e retorna a aproximação recuperada em `X_rec`. Depois de ter analisado e entendido o código em `recover_data`, use essa função para recuperar uma aproximação do primeiro exemplo em X. Você deve obter um valor próximo de [-1,047 -1,047]. Forneça o trecho de código que você usou para isso em seu relatório.

2.3.3 Visualizando as projeções

O script `Parte2/main.py` finalmente apresenta em um mesmo gráfico os dados originais e os dados projetados. Na Figura 4, os pontos de dados originais são indicados com os círculos azuis, enquanto os pontos de dados projetados são indicados com as cruzes vermelhas. Repare que a projeção efetivamente retém apenas a informação na direção dada por U_1.

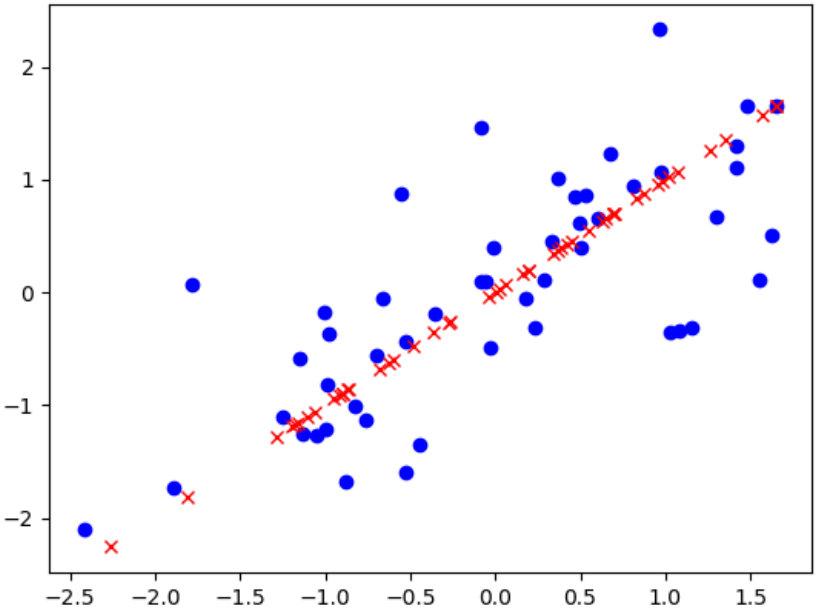

Figura 4: Conjuntos de dados: original e projetado.

3 Parte 3 - Detecção de Anomalias

Nesta parte, você implementará um algoritmo de detecção de anomalia para detectar comportamentos anômalos nos servidores de um centro de dados. As características medem a vazão (throughput) (mb/s) e a latência (ms) de resposta de cada servidor. Enquanto seus servidores estavam funcionando, foram coletados $m = 307$ exemplos de como eles estavam se comportando. Há a suspeita de que a grande maioria desses exemplos são exemplos *normais* (não anômalos) dos servidores que operam normalmente, mas também pode haver alguns exemplos de servidores que atuam de forma anômala nesse conjunto de dados. Esses dados são apresentados na Figura 5.

Você usará um modelo gaussiano para detectar exemplos anômalos em seu conjunto de dados. Você usará um conjunto de dados 2D que permitirá que você visualize o que o algoritmo está fazendo. Nesse conjunto de dados, você ajustará uma distribuição gaussiana e então encontrará valores que têm probabilidade muito baixa e, portanto, podem ser considerados anomalias.

3.1 Distribuição Gaussiana

Para realizar a detecção de anomalia, você precisará primeiro ajustar um modelo à distribuição dos dados.

Dado um conjunto de treinamento $\{x^{(1)}, \ldots, x^{(m)}\}$ (onde $x^{(i)} \in R^n$), você deve estimar a distribuição gaussiana para cada uma das características x_j.

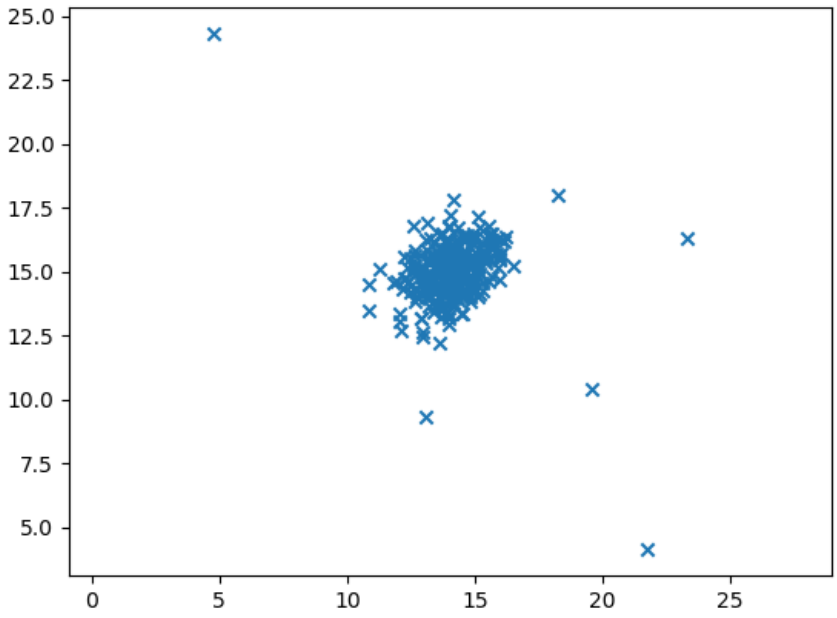

Figura 5: Conjunto de dados - servidores em um *data center*.

Para cada característica $j = 1, \ldots, n$, você precisa encontrar os parâmetros μ_j e σ_j^2 que se encaixam nos dados na j-ésima dimensão $x(1), ..., x(m)$ (a j-ésima dimensão de cada exemplo).

A distribuição gaussiana é dada por

$$p(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

3.2 Estimativa de parâmetros para uma gaussiana

Você pode estimar os parâmetros (μ_j, σ_j^2) da j-ésima característica usando as equações apresentadas em aula. Em particular, sua tarefa é implementar código da função `estimate_gaussian_params`. Esta função toma como entrada a matriz de dados X e deve produzir um vetor `mu` de dimensão n que contém a média de todas as n características e outro vetor `sigma2` de dimensão n que contém as variâncias de todas as características.

Uma vez que tenha completado `estimate_gaussian_params`, sua próxima tarefa será visualizar os contornos da distribuição gaussiana ajustada. O script `Parte3/main.py` produz essa vizualização. Se sua implementação foi feita de forma correta, você deve obter um gráfico semelhante à Figura 6. Dessa figura, você pode ver que a maioria dos exemplos está na região com maior probabilidade, enquanto que os exemplos anômalos estão nas regiões com probabilidades menores.

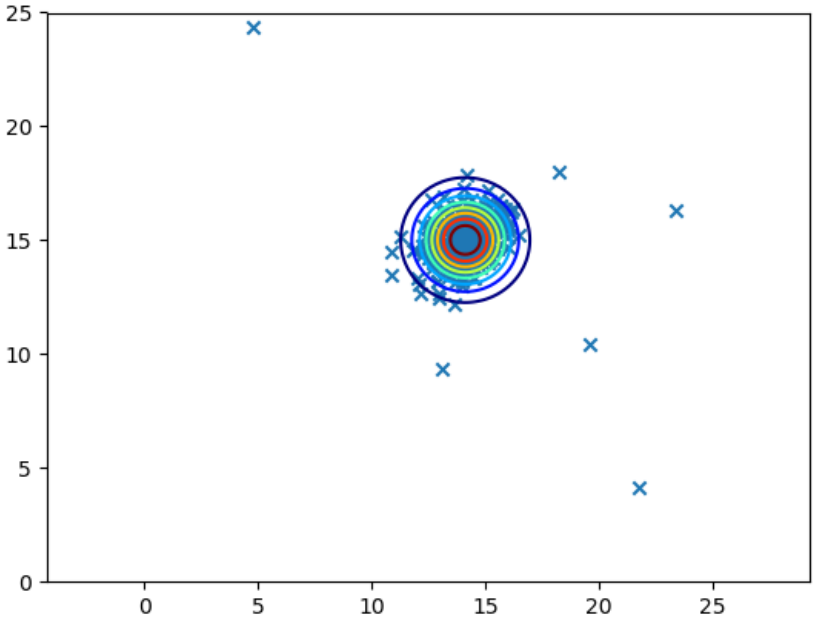

Figura 6: Conjunto de dados - servidores em um *data center*.

3.3 Selecionando ϵ

Agora que você avaliou os parâmetros gaussianos, você pode investigar quais exemplos têm uma probabilidade muito alta dada essa distribuição e quais exemplos têm uma probabilidade muito baixa. Os exemplos de baixa probabilidade são mais prováveis de ser as anomalias em nosso conjunto de dados.

Uma maneira de determinar quais exemplos são anomalias é selecionar um limite ϵ. Uma vez feita essa seleção, se um exemplo x tiver uma baixa probabilidade $\Pr(x) < \epsilon$, então x é considerado uma anomalia.

Uma forma adequada de selecionar o valor de ϵ é com base em um conjunto de validação cruzada. Nesta parte do exercício, você irá completar a implementação de um algoritmo para selecionar ϵ usando a medida F_1 em um conjunto de validação cruzada. O início da implementação deste algoritmo está na função `select_epsilon`. Você deve usar o conjunto de validação cruzada fornecido $\{(x_{cv}^{(1)}, y_{cv}^{(1)}), \ldots, x_{cv}^{(m)}, y_{cv}^{(m)})\}$, onde o rótulo $y = 1$ corresponde a um exemplo anômalo. Para cada exemplo no conjunto de validação, você deve computar $\Pr(x_{cv}^{(i)})$. O vetor de todas essas probabilidades $\Pr(x_{cv}^{(1)}), \ldots, \Pr(x_{cv}^{(m_{cv})})$, assim como os rótulos correspondentes, devem ser passados para a função `select_epsilon`. Na implementação da função `select_epsilon`, para muitos valores diferentes de ϵ, você irá calcular o resultado F_1 resultante ao calcular quantos exemplos o limite atual classifica corretamente e incorretamente.

A função `select_epsilon` deve retornar dois valores. O primeiro é o valor selecionado para ϵ. A função também deve retornar o escore F_1, que indica o

quão bem você está realizando a tarefa de encontrar as anomalias verdadeiras, dado um determinado valor de epsilon. O escore F_1 é calculado como uma função de duas outras medidas, precisão (*prec*) e revocação (*rec*):

$$F_1 = \frac{2 \times prec \times rec}{prec + rec}$$

Você calcula a precisão (*precision*) e a revocação (*recall*) por:

$$prec = \frac{tp}{tp + fp}$$

$$rec = \frac{tp}{tp + fn}$$

onde

- tp é a quantidade de verdadeiros positivos: o rótulo do exemplo diz que é uma anomalia e o algoritmo classificou-o corretamente como uma anomalia.
- fp é a quantidade de falsos positivos: o rótulo do exemplo diz que não é uma anomalia, mas o algoritmo classificou-o incorretamente como uma anomalia.
- fn é a quantidade de falsos negativos: o rótulo do exemplo diz que é uma anomalia, mas nosso algoritmo incorretamente classificou-o como não sendo anômalo.

Em `select_epsilon`, adicione um loop para testar muitos valores diferentes de ϵ e selecionar o melhor ϵ com base na medida F_1.

Você pode implementar o cálculo do escore F_1 usando um loop for para todos os exemplos contidos no conjunto de validação cruzada (para calcular os valores tp, fp, fn). Se sua implementação estiver correta, você deve ver um valor para epsilon de cerca de 8.99e-05. Além disso, o script `Parte3/main.py` deve produzir um gráfico semelhante ao da Figura 7.

4 O que deve ser entregue

Você deve preparar um único relatório para a apresentar sua análise e conclusões sobre as diversas partes desse trabalho. O formato desse relatório deve ser em PDF. Alternativamente à entrega do relatório em PDF, você pode entregar um notebook Jupyter[1].

Independente de escolher entregar um relatório em PDF ou na forma de um notebook Jupyter, entregue também todos os arquivos em Python que você criou para cada parte deste trabalho. Todos os arquivos em Python devem estar em uma única pasta.

Crie um arquivo compactado que contém o relatório (ou notebook Jupyter) e os arquivos (*scripts*) em Python. Esse arquivo compactado deve se chamar

[1] `http://jupyter.org/`

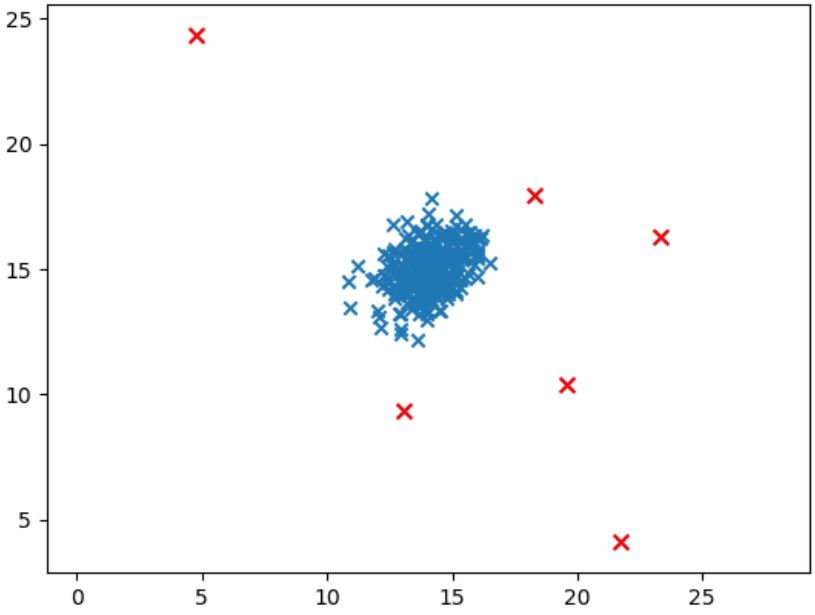

Figura 7: Conjunto de dados - servidores em um *data center* - anomalias destacadas.

`SEU_NOME_COMPLETO_T2.zip`. Esse arquivo compactado deve ser entregue pelo Moodle, até a data acordada.

5 Créditos

Esse trabalho é uma tradução/adaptação dos *programming assignments* encontrados no curso *Machine Learning*[2] encontrado no Coursera. O material original é de autoria do prof. Andrew Ng.

[2] `https://www.coursera.org/learn/machine-learning`