5 - Prática: Estatística p/ Aprendizado de Máquina (I)

Ronny Gabryel Colatino de Souza

Descrição da atividade

Statistics and Probability Refresher, and Python Practice

Cara, esse módulo me surpreendeu muito. Eu sempre achei estatística meio chata na faculdade, mas ver como Python facilita tudo mudou minha perspectiva completamente. Tipo, coisas que eu fazia na mão ou numa calculadora científica, agora é só uma linha de código. E o melhor: consigo visualizar os dados de um jeito que faz muito mais sentido.

Types of Data (Numerical, Categorical, Ordinal)

Tipos de dados	Operações Matemáticas	Exemplo Prático	Função Python Útil
Numérico	Soma, Média, Mediana	Idade: 25, 30, 28	np.mean(), np.sum()
Categórico	Só contagem e frequência	Cor: Azul, Verde, Vermelho	.value_counts()
Ordinal	Ordenação, mas média questionável	Avaliação: Ruim, Médio, Bom	.sort_values()

Então, logo no começo eu meio que subestimei essa parte. Pensava "ah, é só classificar dados, que difícil pode ser?". Mas na real, essa classificação muda tudo na hora de analisar.

Dados Numéricos são os mais tranquilos - idade, salário, altura. Dá pra fazer qualquer conta com eles. Em Python é só jogar num array e pronto.

Dados Categóricos me deram um nó no início. Tipo, como você analisa "cor dos olhos"? Aí descobri o `.value_counts()` e nossa vida ficou muito mais fácil. Só contar quantas vezes cada categoria aparece.

Dados Ordinais foram os que mais me confundiram. São categóricos, mas têm ordem - tipo "péssimo, ruim, médio, bom, ótimo". Você pode ordenar, mas não pode calcular média. Fiquei quebrando a cabeça com isso no começo.

O lance é que se você não identifica o tipo certo, faz besteira na análise. Já vi gente tentando calcular média de código postal... não faz sentido né.

Mean, Median, Mode

Aqui foi onde eu pensei "nossa, Python facilita demais a vida". Lembro de fazer isso tudo na mão no ensino médio.

Média é a mais óbvia - soma tudo e divide. `np.mean()` resolve em dois segundos.

Mediana eu sempre meio que ignorava, mas agora vejo como ela é útil. Pega o valor do meio quando você ordena tudo. O legal é que ela não sofre tanto com valores muito altos ou baixos. Tipo, se você tem salários [2000, 2500, 3000, 50000], a média vai pra quase 15 mil, mas a mediana fica em 2750, que representa melhor o que a maioria ganha.

Moda é meio chatinha em Python. Às vezes não tem valor que se repete, às vezes tem empate. A biblioteca `statistics.mode()` até quebra em alguns casos. Tive que tratar isso na mão algumas vezes.

Activity Using mean, median, and mode in Python

Nessa atividade eu peguei um dataset de idades de um grupo: [18, 19, 20, 21, 22, 22, 23, 65]. A média deu uns 26 anos, mas olhando os dados, a maioria é jovem, só tem um cara mais velho que puxou a média pra cima. A mediana foi 21.5, que faz muito mais sentido.

Foi aí que caiu a ficha média pode enganar quando tem valores atípicos. Salário médio no Brasil parece alto por causa dos super ricos, mas mediana conta a história real.

```
idades = [18, 19, 20, 21, 22, 22, 23, 65]
print(f"Média: {np.mean(idades)}")
print(f"Mediana: {np.median(idades)}")
```

```
Média: 26.25
Mediana: 21.5
```

Activity Variation and Standard Deviation

Turma	Notas	Média	Variância	Desvio Padrão	Interpretação
A - Consistente	[7, 8, 8, 9, 8]	8.0	0.4	0.63	Turma homogênea
B - Dispersa	[4, 6, 8, 10, 12]	8.0	8.0	2.83	Turma heterogênea

Essa parte me abriu a mente. Duas turmas podem ter a mesma média, mas serem completamente diferentes.

Imagine duas turmas com média 7:

- Turma A: todo mundo tirou entre 6 e 8
- Turma B: metade tirou 3, metade tirou 10

Média igual, mas uma é muito mais consistente. É aí que entra desvio padrão.

Variância mede o quão espalhados os dados estão. O problema é que ela fica numa unidade esquisita (se você mede altura em metros, variância fica em metros^2).

Desvio padrão é só a raiz da variância, então volta pra unidade original. Muito mais fácil de entender.

Na prática, desvio padrão baixo = dados consistentes. Alto = dados espalhados.

Probability Density Function; Probability Mass Function

Essa parte foi meio abstrata no começo, mas os gráficos ajudaram muito.

PMF é pra dados que você pode contar - número de filhos, faces de um dado. É tipo um gráfico de barras mostrando a probabilidade de cada valor.

PDF é pra coisas contínuas - altura, peso. Aqui não faz sentido perguntar "qual a probabilidade de alguém ter exatamente 1.7532847 metros?". Você trabalha com intervalos.

Plotar no matplotlib fez tudo fazer sentido. PMF = barrinhas, PDF = curva suave.

Common Data Distributions (Normal, Binomial, Poisson, etc)

Aqui conectou teoria com mundo real. Cada distribuição tem seu lugar.

Normal é a famosa curva de sino. Altura, peso, QI, notas de prova um monte de coisa segue isso. O legal é que se você sabe que é normal, já sabe que 68% dos dados ficam a 1 desvio padrão da média.

Binomial é pra coisas tipo jogar moeda 10 vezes. Quantas caras vão dar Útil pra testes.

Poisson é pra eventos rarosquantos emails chegam por hora, quantos acidentes por semana. Bem específica, mas quando se aplica, é perfeita.

Em Python, `scipy.stats` tem tudo pronto. É só importar e usar.

Activity Percentiles and Moments

Percentis são mais úteis do que eu pensava. Percentil 90 significa que você está melhor que 90% das pessoas. Vestibular usa isso.

Momentos foram mais complicados:

- 1º momento = média
- 2º momento = variância
- 3º momento = assimetria (se a curva pende pra um lado)
- 4º momento = curtose (se a curva é pontiaguda ou achatada)

Na prática, só usei os dois primeiros. Os outros são mais pra quem faz pesquisa séria.

Activity A Crash Course in matplotlib

Matplotlib é poderoso, mas confesso que a sintaxe é meio estranha no começo. Tem duas formas de usar: `plt.plot()` que é mais simples, e a orientada a objetos com `fig, ax = plt.subplots()`.

Pra gráficos básicos, `plt.plot()` resolve. Mas quando você quer mais controle, a versão orientada a objetos é melhor.

O que mais me impressionou foi a customização. Dá pra mudar tudo - cores, estilos, títulos, legendas. Os gráficos ficam bem profissionais.

Activity Advanced Visualization with Seaborn

Seaborn salvou minha vida. Pega matplotlib e deixa tudo mais bonito automaticamente.

Uma linha `sns.histplot()` faz um histograma muito mais bonito que matplotlib puro. As cores são melhores, os estilos são mais modernos.

Minha função favorita é `sns.pairplot()`. Ela cria uma matriz gigante mostrando como todas as variáveis se relacionam. Perfeito pra explorar dados novos sem ter que fazer gráfico por gráfico.

Activity Covariance and Correlation

Essa parte foi importante pra não cair em pegadinhas. Covariância mede se duas coisas se movem juntas, mas o número é difícil de interpretar.

Correlação é muito melhor, vai de -1 a 1:

- Perto de 1 = quando uma sobe, a outra sobe
- Perto de 0 = não tem relação linear
- Perto de -1 = quando uma sobe, a outra desce

correlação não é causalidade Venda de sorvete e número de afogamentos são correlacionados, mas não é o sorvete que causa afogamento. Os dois sobem no verão.

`sns.heatmap()` com matriz de correlação fica lindo e mostra tudo numa tacada.

Exercise Conditional Probability

Probabilidade condicional mexeu com minha cabeça. P(A|B) é diferente de P(A).

Exemplo: qual a chance de chover se está nublado? É bem diferente da chance geral de chover.

Em Python, usei filtering de dataframes:

É matemática, mas aplicada a dados reais fica mais interessante.

```
dados = {
    'tempo': ['ensolarado', 'nublado', 'nublado', 'chuvoso', 'nublado', 'ensolarado'],
    'chuva': [False, True, False, True, True, False]
}

df = pd.DataFrame(dados)
```

```
nublados = df[df['tempo'] == 'nublado']
chuva_nublado = nublados[nublados['chuva'] == True]
prob = len(chuva_nublado) / len(nublados)
```

Exercise Solution Conditional Probability of Purchase by Age

Nessa atividade analisei dados de e-commerce. Quem tem mais chance de comprar baseado na idade

Descobri que gente entre 25-35 compra mais. Faz sentido já têm dinheiro, mas ainda gastam em coisas não essenciais.

Isso tem aplicação direta em marketing. Se você vai fazer propaganda no Facebook, foca nessa faixa etária.

Bayes' Theorem

Teorema de Bayes quase fritou meu cérebro. Ele inverte probabilidades: se você sabe P(B|A), pode calcular P(A|B).

Exemplo clássico: teste de COVID com 95% de precisão. Se deu positivo, qual a chance real de ter COVID? Não é 95%! Depende de quantas pessoas têm COVID na população.

Se só 1% da população tem COVID, mesmo com teste positivo, você tem menos de 20% de chance de realmente ter. Contraintuitivo, mas matemática não mente.

Biblioteca	Função Principal	Comandos Essenciais	Quando Usar
NumPy	Cálculos matemáticos	np.mean(), np.std(), np.median()	Estatísticas básicas
Pandas	Manipulação de dados	.describe(), .value_counts(), .corr()	Análise exploratória
Matplotlib	Gráficos básicos	plt.plot(), plt.hist(), plt.scatter()	Visualizações simples
Seaborn	Gráficos bonitos	sns.histplot(), sns.heatmap(), sns.pairplot()	Visualizações profissionais
SciPy.stats	Estatística avançada	norm, binom, poisson	Distribuições e testes

Machine Learning with Python

Cara, esse parte foi incrivel até agora era isso que eu tava esperando machine learning em python foi uma virada de chave total na minha cabeça. Sempre ouvi falar de inteligência artificial e machine learning como algo super complexo, mas ver na prática como é "só" matemática aplicada com Python mudou tudo. O que mais me impressionou foi perceber que não é mágica é estatística turbinada que encontra padrões nos dados.

Supervised vs. Unsupervised Learning, and TrainTest

Logo de cara, essa divisão entre supervisionado e não supervisionado me ajudou a organizar as ideias. Antes eu achava que era tudo a mesma coisa.

Aprendizado Supervisionado é quando você tem as "respostas" - tipo, você mostra pro algoritmo mil fotos de gato com a etiqueta "gato" e mil fotos de cachorro com "cachorro". Aí ele aprende a diferença. É como ensinar uma criança mostrando exemplos.

Aprendizado Não Supervisionado é mais doido você joga os dados e fala "acha padrões aí". Tipo clustering, onde o algoritmo agrupa coisas similares sem você falar o que é similar.

O conceito de **Train/Test** foi fundamental. Você nunca testa o modelo com os mesmos dados que usou pra treinar, senão é cola né. É tipo estudar com o gabarito e depois fazer a prova com as mesmas questões.

Em Python, usar `train_test_split()` do sklearn virou rotina. Geralmente 80% treino, 20% teste.

Activity Using TrainTest to Prevent Overfitting a Polynomial Regression

Nessa atividade eu finalmente entendi o que é overfitting na prática. Criei um modelo de regressão polinomial que se ajustava perfeitamente aos dados de treino - parecia incrível!

Mas quando testei com dados novos, o desempenho despencou. O modelo tinha "decorado" os dados de treino, mas não aprendeu o padrão real. É como decorar as respostas de uma prova sem entender a matéria.

A sacada foi testar vários graus de polinômio:

- Grau 1: muito simples, underfitting
- Grau 3-4: equilibrado, boa generalização
- Grau 10+: overfitting total

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
```

Foi aí que caiu a ficha complexidade não é sempre melhor.

Bayesian Methods Concepts

Métodos Bayesianos foram os que mais me confundiram no início. É uma forma diferente de pensar sobre probabilidade.

Em vez de dizer "a probabilidade é X", Bayes fala "dado o que eu já sei, a probabilidade é Y". Conforme você vai coletando evidências, vai atualizando suas crenças.

No contexto de ML, é muito útil quando você tem poucos dados ou quer incorporar conhecimento prévio. Tipo, se você tá classificando emails como spam, você já sabe que emails com "GANHE DINHEIRO FÁCIL" provavelmente são spam.

O legal é que métodos Bayesianos dão não só a resposta, mas também o quão confiantes eles estão. Muito útil em aplicações críticas tipo medicina.

Activity Implementing a Spam Classifier with Naive Bayes

Essa foi uma das atividades mais satisfatórias! Implementar um classificador de spam do zero usando Naive Bayes.

O "naive" é porque assume que as palavras são independentes entre si, o que obviamente não é verdade. Mas funciona surpreendentemente bem!

O processo foi:

1. **Tokenizar** os emails (separar em palavras)
2. **Contar frequências** de cada palavra em spam vs não-spam
3. **Aplicar Bayes** para classificar novos emails

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
X_train_vec = vectorizer.fit_transform(emails_treino)
```

O resultado foi impressionante - mais de 90% de acurácia! E o modelo era super rápido, processava milhares de emails por segundo.

K-Means Clustering

K-Means foi meu primeiro contato com clustering e achei genial a simplicidade do algoritmo.

A ideia é: escolha K pontos aleatórios como "centros", agrupe cada ponto ao centro mais próximo, mova os centros para o meio de cada grupo, repita até estabilizar.

É tipo organizar uma festa você coloca mesas aleatórias, as pessoas se sentam na mesa mais próxima, você move as mesas pro centro de cada grupo, e assim vai até todo mundo ficar confortável.

O desafio é escolher o K certo. Usar o "método do cotovelo" ajuda você plota a distorção vs número de clusters e procura onde a curva "dobra".

Activity Clustering people based on income and age

Nessa atividade, clusterizei pessoas baseado em renda e idade. Foi interessante ver os grupos emergirem naturalmente:

- **Cluster 1:** Jovens com renda baixa (estudantes?)
- **Cluster 2:** Meia-idade, renda média (profissionais estabelecidos)

- **Cluster 3:** Mais velhos, renda alta (executivos sêniores)

O código foi bem direto:

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3)
clusters = kmeans.fit_predict(dados_renda_idade)
```

A visualização no scatter plot ficou muito clara - dava pra ver os grupos bem definidos. Isso tem aplicação direta em segmentação de clientes para marketing.

Measuring Entropy

Entropia foi um conceito que demorei pra digerir. É uma medida de "bagunça" ou incerteza nos dados.

Se você tem um dataset onde todos os exemplos são da mesma classe, entropia = 0 (zero incerteza). Se metade é de uma classe e metade de outra, entropia é máxima (máxima incerteza).

A fórmula matemática é chata, mas o conceito é útil: quanto menor a entropia, mais "puro" é o conjunto de dados.

Em machine learning, algoritmos como árvores de decisão usam entropia para decidir como dividir os dados sempre tentando diminuir a entropia a cada divisão.

Decision Trees Concepts

Árvores de Decisão viraram meu algoritmo favorito pela simplicidade de interpretar. É literalmente como você toma decisões na vida real.

"Se a idade > 30 E renda > 50k, então classe A. Senão, se idade < 25, então classe B..." e por aí vai.

O algoritmo constrói a árvore sempre escolhendo a pergunta que mais diminui a entropia (ou seja, que melhor separa as classes). É guloso - sempre faz a melhor escolha no momento, sem pensar no futuro.

O problema é que árvores podem ficar muito complexas e dar overfitting. Por isso usa poda (pruning) ou limita a profundidade.

Activity Decision Trees Predicting Hiring Decisions

Nessa atividade, treinei uma árvore para prever decisões de contratação baseado em anos de experiência, nível de educação e outras variáveis.

O legal foi visualizar a árvore resultante dava pra ver exatamente como o algoritmo "pensa":

Isso é ouro para RH - consegue explicar exatamente por que uma decisão foi tomada, diferente de outros algoritmos que são "caixas pretas".

Ensemble Learning

Ensemble Learning foi uma revelação: vários algoritmos fracos juntos podem ser mais fortes que um algoritmo forte sozinho.

É como pedir opinião para várias pessoas antes de tomar uma decisão importante. Cada pessoa pode errar individualmente, mas a maioria provavelmente vai acertar.

As principais técnicas:

- **Bagging:** Treina vários modelos com diferentes amostras dos dados
- **Boosting:** Treina modelos sequencialmente, cada um corrigindo erros do anterior
- **Voting:** Simplesmente vota pela resposta mais comum

Random Forest é um exemplo clássico - são centenas de árvores de decisão votando juntas.

Activity XGBoost

XGBoost foi o algoritmo mais pesado que usei, mas os resultados compensaram. É como o "ferrari" dos algoritmos de boosting.

A ideia é treinar modelos sequencialmente, onde cada novo modelo foca especificamente nos casos que os modelos anteriores erraram. É como ter um professor particular que só trabalha suas dificuldades.

```
import xgboost as xgb

model = xgb.XGBClassifier()
model.fit(X_train, y_train)
```

O desempenho foi impressionante, mas o treinamento demorou muito mais que outros algoritmos. É uma questão de trade-off entre precisão e velocidade.

Support Vector Machines (SVM) Overview

SVM foi o algoritmo mais matemático que estudei. A ideia central é encontrar a "melhor linha" (ou hiperplano) que separa as classes.

Mas o que é "melhor" É a linha que tem a maior "margem" ou seja, que fica mais longe possível dos pontos mais próximos de cada classe.

O truque genial do SVM é o "kernel trick" quando os dados não são linearmente separáveis, ele "mapeia" para uma dimensão maior onde se tornam separáveis. É como olhar uma sombra 2D de um objeto 3D complexo.

Kernels comuns:

- **Linear:** Para dados já separáveis
- **RBF (Radial):** Para padrões circulares/complexos
- **Polinomial:** Para relações polinomiais

Activity Using SVM to cluster people using scikit-learn & Activity MAC Installing Graphviz

Essas duas atividades foram mais técnicas. Na primeira, usei SVM para classificação (não clustering, o nome confundiu um pouco).

O interessante foi comparar diferentes kernels:

- Kernel linear funcionou bem com dados simples
- Kernel RBF capturou padrões mais complexos
- Mas cuidado com overfitting - RBF pode ser muito flexível

A instalação do Graphviz foi chatinha no começo, mas necessária para visualizar árvores e outros gráficos. No Windows foi mais complicado que no Linux.

```
from sklearn.svm import SVC

svm_linear = SVC(kernel='linear')
svm_rbf = SVC(kernel='rbf')
```

Conclusões

Cara, esse módulo foi surreal. Tipo, eu sempre tive aquela imagem de que estatística era coisa chata da faculdade e machine learning era algo de outro mundo, super complexo. Mas depois de ver tudo funcionando na prática com Python, mudou completamente minha perspectiva.

O que mais me chamou atenção foi descobrir que por trás de toda essa parada de IA tem matemática e estatística coisas que a gente aprende na escola, só que aplicadas de um jeito inteligente. Ver um algoritmo simples como Naive Bayes conseguir 90% de acerto em classificar spam foi tipo "nossa, é isso mesmo?". Não é mágica, é só a máquina encontrando padrões que nosso cérebro não consegue processar tão rápido.

A experiência de botar a mão na massa foi o que fez a diferença. Uma coisa é o professor explicar overfitting, outra é você ver seu modelo funcionando

perfeitamente nos dados de treino e depois indo pro buraco nos dados de teste. Aí você entende na prática por que não pode "colar" no machine learning.

Python salvou minha vida nesse módulo. Coisas que eu fazia na calculadora ou no papel, agora é uma linha de código. E matplotlib/seaborn cara, poder visualizar os dados desse jeito mudou tudo. Um gráfico fala mais que mil números numa planilha.

O que me deixa mais animado é que agora eu vejo aplicação em tudo. Tipo, quando vejo uma propaganda no Instagram, já penso "eles usaram clustering pra me segmentar". Quando o Gmail filtra spam, lembro do Naive Bayes. É como se tivesse ganhado óculos especiais pra enxergar o mundo digital.

Referencias

Material do curso:

- Todo o conteúdo do card "Estatística para Aprendizado de Máquina (I)"
- Os notebooks e exercícios que fizemos durante o módulo

Sites usados para reforçar

Pra revisar estatística básica:

- Khan Academy (Statistics and Probability) exemplos práticos
- StatQuest no YouTube

Python pra ciência de dados:

- Documentação oficial do Python
- NumPy docs
- Pandas docs
- Matplotlib
- Seaborn

Machine learning na prática:

- Scikit-learn - https://scikit-learn.org/stable/
- Machine Learning Mastery - https://machinelearningmastery.com/