Nome: Felipe Bortolletto
Professor: Prof. Anderson Ara
Disciplina: TEORIA DO APRENDIZADO ESTATÍSTICO
Data: 26 de abril de 2024

Enunciado da Atividade

Para o conjunto de dados Rice (Cammeo and Osmancik).,faça a predição da variavel "Class"utilizando classificadores SVM e de Redes Neurais. Utilize métodos de validação e medidas de desempenho cabíveis. Crie um relatório de até 5 páginas sobre a comparação dos classificadores.

Introdução

Este trabalho tem como objetivo apresentar uma aplicação prática das abordagens de classificação utilizando o classificador SVM e o modelo de Redes Neurais. Especificamente, neste estudo, utilizaremos o modelo perceptron multicamadas (MLP) e um segundo modelo com uma adaptação em sua arquitetura.
O dataset de comparação Rice (Cammeo and Osmancik) contém sete variáveis de entrada para a classificação, todas relacionadas ao formato e às dimensões dos grãos de arroz. Dessa forma, buscamos implementar os modelos para classificar os grãos de acordo com as espécies Cammeo e Osmancik.
As rotinas implementadas foram feitas em python, e podem ser encontradas em Github

SVM e Redes neurais

Segundo [1], as Máquinas de Vetores de Suporte (SVM, do inglês Support Vector Machines) são um processo de aprendizado de máquina supervisionado, baseado nos princípios da Minimização do Risco Empírico e da dimensão de Vapnik-Chevonenkis (VC). As SVMs são utilizadas para reconhecimento de padrões e estimativas de regressão. Sua metodologia envolve encontrar um hiperplano que melhor separa diferentes classes de pontos de dados em um espaço dimensional. Segundo sua teoria, ao maximizar a margem entre as classes, as SVMs buscam garantir um bom desempenho de generalização. Apesar de trabalhar com hiperplanos lineares, a utilização de kernels se mostra como um grande aliado das SVMs, ao permitir a transformação dos dados para dimensões superiores, onde a separação linear é possível. Isso possibilita a aplicação de SVMs em problemas não lineares, permitindo encontrar fronteiras de decisão mais complexas e aumentando a capacidade do modelo de se ajustar ao conjunto de dados. Dessa forma,

fornecendo flexibilidade, a pluralidade de funções kernel a serem utilizadas torna as SVMs altamente adaptáveis e eficazes em uma ampla gama de problemas de aprendizado de máquina. Essa metodologia se destaca por garantir um problema de otimização convexo, ou seja, que possui um mínimo global bem definido, em contraste com arquiteturas de redes neurais, que frequentemente ficam presas em mínimos locais devido à sua alta complexidade.

Vamos então partir para a segunda metodologia aplicada neste estudo. Segundo [2], uma rede neural artificial é um modelo computacional que recebe seu nome pela "similaridade"ao funcionamento dos neurônios do cérebro humano. Cada neurônio de uma rede neural é uma unidade de processamento que recebe uma entrada, processa esses sinais e gera uma saída, funcionando de forma similar a uma função que recebe x e retorna y. O funcionamento da rede neural atribui pesos e vieses a fim de encontrar a melhor combinação de parâmetros que represente os dados de entrada, determinando a importância de cada entrada para a saída do neurônio e tenha capacidade de generalização suficiente para realizar previsões em novos conjuntos de dados.

O perceptron, em específico, é uma arquitetura de rede neural que possui uma única camada. Recebe o vetor X de entradas, com cada uma das variáveis x_i sendo multiplicada por um peso sináptico, realizando a soma ponderada dessas entradas e aplicando uma função de ativação para produzir a saída. O perceptron aprende a classificar padrões linearmente separáveis, ajustando seus pesos com base nos erros de classificação.

Já a arquitetura perceptron multicamadas (MLP) é uma rede neural feedforward que possui a mesma fundamentação do perceptron, porém com mais camadas ocultas entre a camada de entrada e a camada de saída. Cada neurônio em uma camada oculta processa as saídas da camada anterior, permitindo que a rede aprenda a representar relações mais complexas nos dados de entrada. O MLP é capaz de aprender a mapear relações não lineares e é amplamente utilizado em tarefas de classificação e reconhecimento de padrões.

Tendo em mente o número crescente de parâmetros a ser adotado para a solução do nosso sistema, é evidente que a complexidade do universo de soluções para arquiteturas de redes neurais é alta. Isso acarreta um domínio de soluções com alta rugosidade e, consequentemente, com a presença de mínimos locais. Com isso, os hiperparâmetros do sistema se tornam extremamente relevantes para encontrar uma solução.

Por outro lado, todo o sistema de redes neurais é relativamente simples: as funções de ativação utilizadas precisam ser deriváveis. Implementando a metodologia do backpropagation, é possível, através do cálculo de derivadas simples e de uma função de perda específica, ajustar os pesos da rede para minimizar o erro do sistema.

Implementção SVM

A implementação do SVM pode ser encontrada em SVM . Para ambas as abordagens iremos normalizar os dados antes da aplicação das metodologias. A função utilizada para normalização é vinda do scikit-learn, de nome StandardScaler, importamos também a função ShuffleSplit para geração e utilizar a metodologia de validação K-fold cruzada, bem como a função SVC, que representa nosso modelo SVM.

Com isso, queremos observar três diferentes metodologias de kernel para o SVM: linear, gaussiano e sigmoide. Além disso, analisaremos a variação dos parâmetros C e γ para os três casos.

As figuras 2, 4 e 3 mostram os resultados obtidos para diferentes combinações de C e γ. O parâmetro C é um regularizador do modelo; quanto maior o valor de C, menor é o erro de margem associado, levando a maior complexidade e suscetibilidade ao overfitting. Já o parâmetro γ controla a janela de influência de cada amostra singular para os kernels sigmoide e RBF.

Para o kernel linear, apenas o parâmetro C tem efeito, tendo em vista que o parâmetro γ não é utilizado na equação 0.1. Para os kernels sigmoide (0.2) e RBF (0.3), temos, além do γ, o parâmetro de viés b, que movimenta a solução para cima e para baixo.

$$K(x_i, x_j) = \langle x_i, x_j \rangle \tag{0.1}$$

$$K(x_i, x_j) = \tanh(\gamma \langle x_i, x_j \rangle + b) \tag{0.2}$$

$$K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right) \tag{0.3}$$

Os resultados para todas as metodologias se mostraram satisfatórios para determinadas combinações de C e γ. O valor de γ 0.1428 utilizado para analise, foi obtido através da equação 0.4, sendo n o número de elementos na amostra, e var a variância dos dados. Essa abordagem é a padrão adotada pelo *scikit-learn* e é seria a implementação utilizada caso deixássemos a opção default.

$$gamma = \frac{1}{n \times \text{var}} \tag{0.4}$$

Para o kernel linear (Figura 2), podemos observar que a variação de C para o nosso conjunto de dados não resultou em grandes variações na acurácia. No caso do kernel Sigmoid (Figura 3), maiores valores de γ levaram a valores menores de acurácia, e a solução ótima foi encontrada para C entre 0.75 e 1, com γ igual a 0.01. Já para o kernel RBF (Figura 4), as soluções foram mais uniformes, e diferentes combinações indicaram uma solução ótima.

Implementção MLP

Inicialmente implementamos o modelo MLP - Perceptron pela biblioteca scikit-learn. A abordagem é bastante simples e pode ser observada no final da rotina SVM. Alguns parâmetros podem ser ajustados para a aplicação do modelo, destacando-se o número de camadas ocultas, o número de épocas para treinamento, a função de ativação escolhida e a taxa de aprendizado. Uma implementação inicial com 300 épocas, 100 camadas ocultas, função de ativação Relu e taxa de aprendizado de 0.001 resultou em uma acurácia de 0.9278. Além disso, exploramos diferentes possibilidades de implementação. A Tabela **??** mostra as diferentes configurações adotadas para a solução.

Ao todo, temos um conjunto de 55 soluções, cada uma com uma combinação única dos hiperparâmetros mostrados na tabela. O desempenho médio das aplicações em termos de acurácia foi de 0.9269, com o menor valor sendo 0.9258 e o maior valor sendo 0.929131, ambos para aplicações com função de ativação Relu. As médias de acurácia para as funções de ativação foram 0.9268 para a identidade, 0.9265 para a logística, 0.9274 para Relu e 0.9266 para tanh. Perceba que os resultados mostraram diferenças mínimas entre si, apenas para a terceira casa decimal, o que indica que o modelo desempenha bem na classificação dos dados, independentemente dos hiperparâmetros escolhidos.

Variável	**Valores Possíveis**
Épocas	200, 300, 400
Número de camadas ocultas	16, 50, 100, 150
Função de ativação	logística, Relu, identidade, tanh
Taxa de aprendizado	0.001, 0.01, 0.1

Tabela 1: Variações adotadas

A biblioteca scikit-learn oferece uma opção de implementação sólida, simples e eficaz. No entanto, ela impossibilita a criação de novas arquiteturas e também a visualização do aprendizado. Por essas razões, optamos por criar uma nova arquitetura de rede neural para a solução do mesmo problema usando a biblioteca dedicada PyTorch. A implementação é mostrada na rotina MLP Pytorch.

Aqui, utilizamos uma arquitetura composta por três camadas e três funções de ativação. A primeira camada recebe os parâmetros de entrada e trabalha com 32 camadas ocultas, passando pela função de ativação Relu(). Em seguida, a próxima camada recebe 32 parâmetros de entrada, além de mais 32 camadas ocultas, passando novamente pela função Relu() para nossa terceira camada, que recebe 32 parâmetros de entrada e retorna um único parâmetro para nossa última função de ativação sigmoid. Dessa forma, nossa saída é dada por uma probabilidade, onde valores entre 0 e 0.49 são atribuídos à classe Cammeo e valores entre 0.50 e 1 à classe Osmancik. Obtemos um resultado de acurácia, precisão, recall e F1-Score de 0.9265. A Figura

1 mostra a perda do treinamento e a perda do teste da rede. Podemos verificar que o modelo se ajustou adequadamente ao conjunto de teste e treinamento, não sofrendo de overfitting ou underfitting. Foram utilizadas 500 épocas e uma taxa de aprendizado de 0.1.

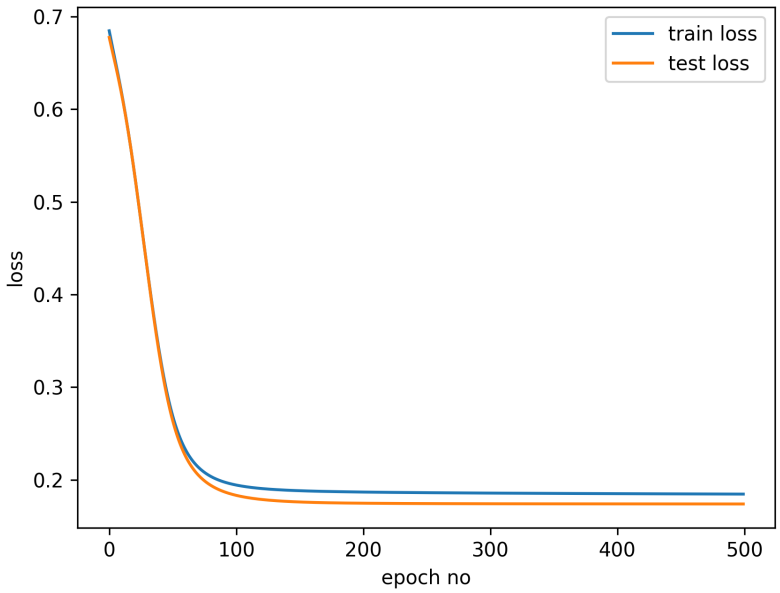

Figura 1: Test loss X Train loss Aplicação rede neural pytorch.

Conclusões

Podemos observar que todas as metodologias aplicadas neste trabalho tiveram bons resultados. Além disso, para os modelos de rede neural suas variações, com alterações nos hiperparâmetros associados, resultaram em grande parte em resultados praticamente idênticos. Ja a análise dos heatmaps de C e γ para o modelo SVM nos indica uma maior sensibilidade do modelo em relação aos parâmetros, onde obtivemos combinações com soluções mais distintas.
Não existe "sopa de graça": o benefício de uma estratégia é o veneno de outra. É importante entender que ambas as metodologias são ferramentas que podem ser aplicadas para a solução de problemas, e ambas possuem suas qualidades e defeitos. Cabe a nós, como operadores do sistema, entender qual situação demanda cada abordagem.

Figura 2: Heatmap C versus Gamma kernel linear.

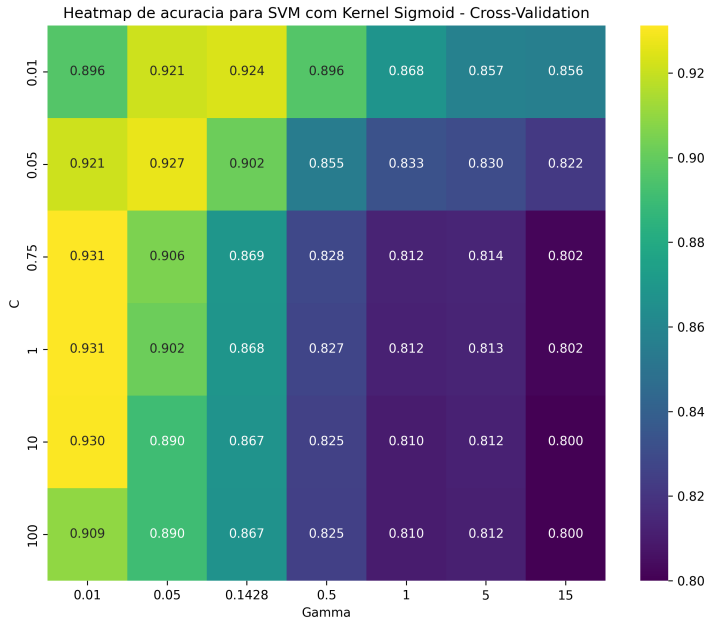

Figura 3: Heatmap C versus Gamma kernel Sigmoid.

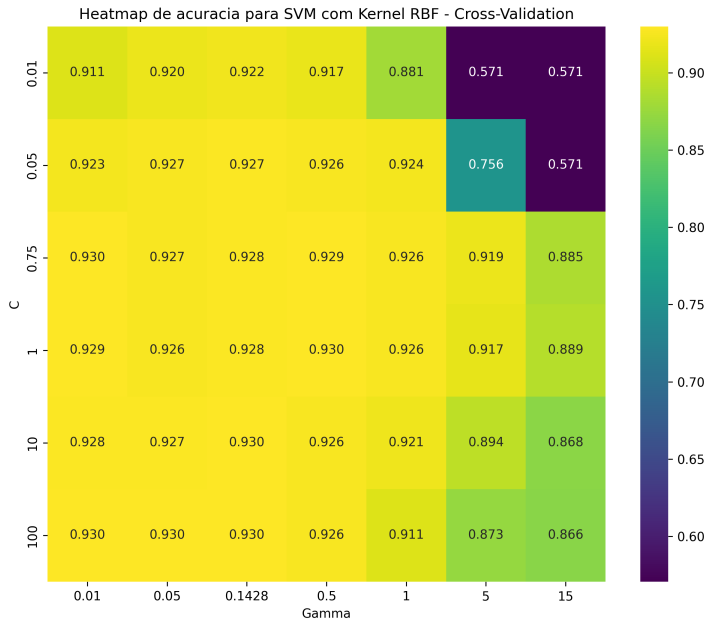

Figura 4: Heatmap C versus Gamma kernel Gausiano - RBF.

Bibliografia

[1] Christopher JC Burges. «A tutorial on support vector machines for pattern recognition». Em: *Data mining and knowledge discovery* 2.2 (1998), pp. 121–167.

[2] Thiago Lombardi Gaspar. «Reconhecimento de faces humanas usando redes neurais MLP». Tese de doutoramento. Universidade de São Paulo, 2006.