Algoritmo SVM

Support Vector Machine (SVM) é um algoritmo de aprendizado supervisionado robusto usado principalmente para tarefas de **classificação** e, em menor grau, para **regressão**. A ideia central por trás do SVM é *encontrar um hiperplano que melhor separa as classes em um espaço de características.*

Origem e Criadores

O conceito de SVM foi introduzido por Vladimir Vapnik e seus colegas, incluindo Alexey Chervonenkis, no contexto do desenvolvimento da Teoria da Aprendizagem Estatística nos anos 1960-70. No entanto, o SVM como o conhecemos hoje foi popularizado por Vapnik e Corinna Cortes em seu trabalho seminal "Support-Vector Networks" publicado em 1995.

Referências Teóricas

- Artigo Principal: Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine Learning, 20(3), 273-297.
- Teoria da Aprendizagem Estatística: Vapnik, V. N. (1998). Statistical Learning Theory. Wiley.

Hiperparâmetros do SVM

1. Kernel:
Define a função de kernel a ser usada para transformar os dados. As opções incluem:
- Linear
- Polinomial
- RBF (Radial Basis Function)
- Sigmoid

2. C (Penalty Parameter):
O parâmetro de regularização que controla a escolha do hiperplano. Um valor alto de `C` tenta minimizar o erro de classificação, enquanto um valor mais baixo prioriza uma maior margem de separação, permitindo alguns erros na classificação.

3. Gamma:
Específico para os kernels RBF, polinomial, e sigmoid. Define o quanto a influência de um único treinamento é, que afeta o “decay” exponencial no alcance de influência dos vetores de suporte.

4. Degree:
Grau do polinômio usado no kernel polinomial.

Otimização de Hiperparâmetros

A seleção dos hiperparâmetros adequados é crítica para o desempenho do SVM. As técnicas comuns incluem:

Grid Search: Pesquisa exaustiva de uma variedade de combinações de parâmetros.

Random Search: Pesquisa aleatória em um espaço definido de parâmetros.
Bayesian Optimization: Técnicas baseadas em otimização bayesiana para selecionar os melhores hiperparâmetros com menos iterações.

Campo de Utilização

SVM é fundamentalmente um algoritmo de classificação, mas também pode ser adaptado para regressão (chamado de **SVR** - **Support Vector Regression**). É menos adequado diretamente para problemas de agrupamento, embora possa ser aplicado indiretamente em algumas técnicas de agrupamento de conjuntos de dados previamente rotulados.

Aplicações Comuns

Reconhecimento de Imagem: Classificação de objetos em imagens devido à sua alta capacidade em lidar com grandes dimensões de dados.
Bioinformática: Classificação de sequências de DNA e previsão de doenças.
Detecção de Fraudes: Separação de atividades normais de fraudulentas em dados de transações.

O **SVM** possui uma forte base matemática e tem sido amplamente usado em diversos domínios devido à alta precisão que pode alcançar quando bem otimizado.

Algoritmo MLP

O **Multi-Layer Perceptron (MLP)** é um tipo de rede neural **feedforward** composta por **pelo menos três camadas de nós:** uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. Cada nó (neurônio) em uma camada usa uma função de ativação não linear, exceto pelos nós de entrada. MLPs são capazes de aprender representações não lineares complexas, tornando-os adequados para tarefas de classificação e regressão.

Origem e Criadores

O conceito de perceptron foi inicialmente introduzido por **Frank Rosenblatt em 1957** como um classificador linear binário. Os MLPs como redes neurais de múltiplas camadas foram desenvolvidos posteriormente, particularmente após a reintrodução da técnica de retropropagação (backpropagation) para treinar redes neurais, o que foi popularizado nos anos 1980 por pesquisadores como David E. Rumelhart, Geoffrey E. Hinton e Ronald J. Williams.

Referências Teóricas

Artigo Principal de Retropropagação: Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533-536.
Livro Importante: Rumelhart, D. E., & McClelland, J. L. (1986). Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1. MIT Press.

Hiperparâmetros do MLP

1. Número de Camadas Ocultas:
Define a complexidade da rede. Mais camadas podem capturar padrões mais complexos, mas também podem aumentar o risco de overfitting.

2. Número de Neurônios por Camada:
Afeta a capacidade da rede de aprender representações detalhadas dos dados.

3. Função de Ativação:
Comuns são ReLU, sigmoid, e tanh. As escolhas de ativação podem afetar a rapidez do treinamento e a eficiência da modelagem de não-linearidades.

4. Taxa de Aprendizagem:
Controla o tamanho do passo nas atualizações dos pesos durante a otimização.

5. Batch Size:
O número de amostras processadas antes de atualizar o modelo.

6. Épocas:
Número de passadas completas pelo conjunto de dados de treinamento.

Otimização de Hiperparâmetros

O ajuste de hiperparâmetros em MLPs é crítico para o desempenho, e algumas técnicas incluem:

Grid Search e Random Search: Avaliam diferentes combinações de hiperparâmetros.
Otimização Bayesiana: Explora hiperparâmetros de forma eficiente.
Regularizações como Dropout e L2 penalties ajudam a mitigar o overfitting.

Campo de Utilização

Os **MLP**s são versáteis e são usados tanto para **classificação** quanto para **regressão**. Eles não são diretamente aplicáveis a problemas de agrupamento, mas ajustes como Autoencoders podem auxiliar nessas tarefas.

Aplicações Comuns

Reconhecimento de Padrões:
Como em OCR (Optical Character Recognition) e reconhecimento de fala.

Previsão Financeira:
Modelagem de séries temporais e previsão de preços de ativos.

Ciências da Saúde:
Diagnóstico médico a partir de dados de imagem ou dados clínicos tabulares.

Os MLPs são um ponto de partida clássico em aprendizado profundo, oferecendo uma estrutura simples e poderosa que pode ser expandida para as redes neurais mais complexas como CNNs e RNNs.

Redes Neurais KBANN

KBANN (Knowledge-Based Artificial Neural Network) é uma abordagem que combina redes neurais com conhecimento pré-existente de domínio, visando melhorar o desempenho em tarefas de aprendizado de máquina.

KBANNs integram regras e heurísticas ao treinamento da rede neural, permitindo que o modelo utilize tanto dados empíricos quanto conhecimento específico da área para melhorar suas previsões.

Processamento Heurístico refere-se a uma abordagem analítica onde os requisitos são descobertos com *base nos resultados de cada iteração* de **processamento**, permitindo refinamento e redesenvolvimento contínuos até que resultados satisfatórios sejam alcançados.

Origem

- **Desenvolvimento**: As redes KBANN foram introduzidas e desenvolvidas por **Paul J. Werbos** em 1991, que é conhecido por seu trabalho na teoria e aplicação de redes neurais, especialmente na introdução da retropropagação.

- **Publicação**: O conceito foi detalhado no artigo:

    - **Werbos, P. J.** (1991). "A Menu of Techno-Scientific Knowledge for the KBANN Architecture." *Artificial Intelligence*, 49(1), 162-172.
    - Esse trabalho foi uma das primeiras tentativas de integrar conhecimento especializado em redes neurais e representou um passo importante no avanço de arquiteturas de aprendizado de máquina.

Características

1. **Integração de Conhecimento**:

    - KBANN combina o poder das redes neurais com conhecimento humano codificado. Esse conhecimento pode ser uma série de regras, padrões ou heurísticas que são incorporadas na estrutura da rede.

2. **Treinamento Direcionado**:

    - O treinamento em uma KBANN pode ser orientado por um conhecimento inicial, permitindo um aprendizado mais eficiente e potencialmente mais rápido, já que a rede começa com uma base sólida de conhecimento.

3. **Adaptação e Aprimoramento**:

    - Embora a KBANN utilize conhecimento pré-existente, ela ainda é capaz de adaptar-se e aperfeiçoar-se com novas informações e dados ao longo do tempo, aumentando sua robustez e eficácia em tarefas específicas.

4. **Versatilidade**:
    - Essa abordagem é aplicada em diversas áreas, incluindo reconhecimento de padrões, diagnósticos médicos, sistemas de recomendação e outras aplicações onde o conhecimento prévio é útil.

Fontes

- **Werbos, P. J.** (1991). *A Menu of Techno-Scientific Knowledge for the KBANN Architecture*. Artificial Intelligence.
- **Werbos, P. J. Proceedings of the IEEE 1 October 1990.** Backpropagation Through Time: What It Does and How to Do It
- **Brachman, R. J., & Levesque, H. J.** (2004). *Knowledge Representation and Reasoning*. The MIT Press.
- **Russell, S. J., & Norvig, P.** (2020). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Redes KBANN (Knowledge-Based Artificial Neural Networks) podem ser vistas como um exemplo de redes neurais que utilizam o algoritmo de **backpropagation** (retropropagação) no seu treinamento.

Backpropagation

1. **O que é?**: O backpropagation é um algoritmo usado para treinar redes neurais, que calcula o gradiente da função de perda da rede em relação aos pesos da rede. A ideia é distribuir o erro da saída do modelo para as camadas anteriores, ajustando os pesos na direção que minimiza o erro.

2. **Cálculo de Gradientes**: Durante o processo de treinamento, o erro (perda) é calculado nas saídas da rede. Ao retropropagar o erro, cada camada da rede pode calcular como suas saídas contribuíram para o erro total. Isso é feito usando a regra da cadeia da análise de cálculo.

3. **Ajuste de Pesos**: Os pesos da rede são atualizados com base nos gradientes calculados e a taxa de aprendizado, ajustando a rede para aprender melhor ao longo do tempo.

KBANN e Backpropagation

- **Integração de Conhecimento**: As KBANNs utilizam conhecimento pré-existente (como regras ou heurísticas) que podem ser incorporadas no formato de como os pesos e as conexões da rede estão configurados inicialmente. Isso pode ajudar a rede a começar com um "conhecimento" adequado sobre a tarefa.

- **Treinamento**: Após a inicialização com o conhecimento, a KBANN pode ser treinada usando o algoritmo de backpropagation, ajustando os pesos com base nos dados de

treinamento, assim como ocorre em redes neurais tradicionais. A retropropagação permite que a rede refine suas conexões com base nos dados empíricos que alimentam o modelo.

A rede neural KBANN utiliza o mecanismo de backpropagation durante o seu processo de aprendizado, permitindo que o modelo se ajuste a partir de dados com base no conhecimento prévio, efetivamente combinando aprendizado supervisionado.

As redes **KBANN** representam um avanço importante ao combinar habilidade de aprendizado das redes neurais com inteligência humana. Essa abordagem traz um enfoque colaborativo onde o conhecimento recebido pode potencialmente acelerar e melhorar o desempenho dos algoritmos de aprendizado automático, particularmente em domínios onde informações especializadas são cruciais, ou seja, onde há conhecimento prévio produzido.

Na prática, o backpropagation é um componente fundamental do treinamento de redes neurais e é implementado automaticamente por bibliotecas de aprendizado de máquina como TensorFlow e PyTorch quando você define a estrutura da rede e a função de perda.

Como Funciona o Backpropagation na Prática

1. **Construindo a Rede Neural**: Quando você define a arquitetura do modelo, como as camadas, funções de ativação, e dropout, você está configurando a rede para aprender.

2. **Definindo a Função de Perda**: A função de perda (como `cross_entropy`, `MSE`, etc.) estima o erro entre as previsões do modelo e os valores reais. Isso fornece um feedback que é necessário para o ajuste dos pesos.

3. **Otimização**: Ao treinar o modelo usando `optimizer.step()` após `loss.backward()`, você está efetivamente chamando o processo de backpropagation, que calcula os gradientes do erro em relação aos pesos e atualiza os pesos da rede.

Implementação em Python

```
# Treinamento do modelo
for epoch in range(epochs):
    for i in range(0, len(X_train), batch_size):
        batch_X = X_train[i:i + batch_size]
        batch_y = y_train[i:i + batch_size]

        optimizer.zero_grad()  # Zera os gradientes acumulados
        outputs = model(batch_X)  # Forward Pass

        loss = F.cross_entropy(outputs, batch_y.long())  # Calcula a
perda
```

```
loss.backward()  # Backward Pass
optimizer.step()  # Atualização dos pesos
```

Passo a Passo

1. **`optimizer.zero_grad()`**: Zera os gradientes acumulados do otimizador. Isso é importante porque, no PyTorch, os gradientes são acumulados por padrão. Por isso, no início de cada loop de treinamento, você precisa limpar os gradientes.

2. **`outputs = model(batch_X)`**: Aqui ocorre o **forward pass**. A entrada `batch_X` é passada pela rede neural, e o modelo gera as previsões.

3. **`loss = F.cross_entropy(outputs, batch_y.long())`**: Calcula o valor da perda, que é a medida de quão longe as previsões do modelo estão dos rótulos verdadeiros.

4. **`loss.backward()`**: Aqui começa o **backpropagation**. O PyTorch cálcula os gradientes da função de perda em relação a todos os pesos da rede com base na regra da cadeia, propagando o erro retroativamente através da rede.

5. **`optimizer.step()`**: Este comando aplica os gradientes calculados para atualizar os pesos da rede, tentando minimizar a perda.

Implicações do Processo

O backpropagation é, portanto, um processo implícito no treinamento da rede neural. Quando você chama `loss.backward()` e `optimizer.step()`, você ativa todo um mecanismo de cálculo e atualização de pesos que é fundamental para o aprendizado.

Assim com base no código anterior, abaixo segue um resumo do que já está sendo tratado:

1. **Forward Pass**:
    - O modelo executa o forward pass através das camadas definidas ao chamar `model(inputs)`, onde `inputs` é seu tensor de entrada. Isso ocorre sempre que você chama `model(batch_X)`.
2. **Cálculo da Perda**:
    - A função de perda (neste caso, definida como `F.cross_entropy`) é chamada após o forward pass para calcular a diferença entre as previsões do modelo e os rótulos verdadeiros.
3. **Backpropagation**:
    - Chamando `loss.backward()`, o PyTorch automaticamente calcula os gradientes necessários com base na função de perda e como cada peso contribuiu para esse erro.
4. **Atualização dos Pesos**:

- `optimizer.step()` aplica os gradientes calculados para atualizar os pesos da rede neural.