UFG
UNIVERSIDADE FEDERAL DE GOIÁS

Aula 4

Deep Learning aplicado na classificação de imagens de satélite

Aplicação de Modelos para a Classificação de Culturas

Priscila M. Kai

Roteiro

01 **Aula 1**
Introdução ao Sensoriamento Remoto

02 **Aula 2**
Aquisição de Imagens de Sensoriamento Remoto

03 **Aula 3**
Extração de características

04 **Aula 4**
Aplicação de Modelos para a Classificação de Culturas

05 **Aula 5**
Construção de uma Rede Neural Densa

04

Aplicação de Modelos para a Classificação de Culturas

O que veremos?

Roteiro

1. Algoritmos para classificação
    - kNN
    - SVM
    - Random Forest

2. Ajuste de Hiperparâmetros
    - Hiperparâmetro
    - Pesquisa de grade (Grid Search)
        - Exemplo em Python

Algoritmos para classificação

kNN

kNN é um classificador simples e de fácil implementação, podendo ser usado em variados problemas. Para datasets de grande dimensionalidade não apresenta-se tão eficiente. É recomendado para datasets balanceados e pequenos, sendo sensível a outliers.

No kNN, uma amostra pertencente ao conjunto de teste é classificada de acordo com os exemplos de treinamento mais próximos com base no rótulo dominante entre os exemplos.

Algoritmos para classificação

kNN - Implementação

```
from sklearn.neighbors import KNeighborsClassifier

num_k = 6 #Número de vizinhos a serem usados
knn = KNeighborsClassifier(n_neighbors=num_k)
knn.fit(X, y)
```

Algoritmos para classificação

SVM

O SVM (Support Vector Machines - Máquinas de Vetores de Suporte) são eficientes para datasets com alta dimensionalidade mas são sensíveis a dados ruidosos. Usa hiperplanos de separação como o limite de decisão entre as duas classes.

Em dados linearmente separáveis, é possível construir um hiperplano linear que separa os pontos de dados pertencentes às duas classes. Entretanto, muitas vezes encontramos problemas contendo dados não linearmente separáveis. Nesse caso é usado o SVM não linear.

Algoritmos para classificação

SVM - Implementação

```
from sklearn.svm import SVC

svm_classifier = svm.SVC()
svm_classifier.fit(X, y)
```

Algoritmos para classificação

Random Forest

O Random Forest (Florestas aleatórias) não são sensíveis a outliers, funcionando bem para grandes datasets. No entanto, possui treinamento mais lento em comparação a outros algoritmos, como o kNN. São definidas como um conjunto de árvores de decisão, com aleatoriedade no processo de construção do modelo de cada árvore de decisão.

Algoritmos para classificação

Random Forest - Implementação

```
from sklearn.ensemble import RandomForestClassifier

random_f = RandomForestClassifier()
random_f.fit(X, y)
```

Ajuste de Hiperparâmetro

Hiperparâmetros são parâmetros que podem ser ajustados no processo de treinamento do modelo usado para a classificação.

Assim, o ajuste de parâmetros pode fazer com o que o modelo obtenha um desenho superior em comparação aos parâmetros predefinidos do modelo.

Entre os exemplos de hiperparâmetros temos o número de vizinhos do algoritmo kNN, *kernel* para o SVM e *max_depth* para o Random Forest.

Algoritmos para classificação

Pesquisa em grade (**Grid Search**)

A **pesquisa de grade** faz de forma exaustiva por uso de grade por valores de parâmetro com menor desempenho no dataset de treinamento de maneira a ajustar esses parâmetros ao conjunto de dados, ou seja, mantendo a melhor combinação.

```
from sklearn.model_selection import GridSearchCV

grid = GridSearchCV(classificador, parametros)
grid.fit(trainX, trainY)

print(grid.best_estimator_)
```

Algoritmos para classificação

Grid Search - kNN

```
import pickle
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

#Lista de Hiperparâmetros que queremos testar
n_neighbors = list(range(1,30))

grid = GridSearchCV(KNeighborsClassifier(), param_grid={'n_neighbors':
n_neighbors})
grid.fit(trainX, trainY)

print(grid.best_estimator_)
```

Algoritmos para classificação

Grid Search - kNN

```
#Treinamendo modelo com o melhor estimador
modelo_knn = KNeighborsClassifier(n_neighbors=10)
model.fit(trainX, trainY)

#Salvando o modelo
filename = 'modelo_knn.sav'
pickle.dump(modelo_knn, open(filename, 'wb'))

#Predição
prediction = modelo_knn.predict(testX)

#Classificação
print(classification_report(testY, prediction))
```