Guía de Laboratorio – Aprendizaje automático con flujos de trabajo automáticos y mejora de rendimiento de modelos

Sistema de conocimiento:

- Trabajo con flujos de trabajo automáticos (pipelines)
- Métodos para la mejora del rendimiento de modelos de aprendizaje automático.

Objetivos:

- Utilizar flujos de trabajo automáticos (pipelines) para mejorar el proceso de construcción de modelos de aprendizaje automático.
- Aplicar técnicas que mejoren el rendimiento de modelos de aprendizaje automático.

Bibliografía:

- Johnston, B., & Mathur, I. (2019). *Applied supervised learning with Python: Use scikit-learn to build predictive models from real-world datasets and prepare yourself for the future of machine learning*. Packt Publishing Ltd.

Introducción

Para ejecutarse y producir resultados con éxito, un modelo de aprendizaje automático debe automatizar algunos flujos de trabajo estándar. El proceso de automatizar estos flujos de trabajo estándar se puede hacer con la ayuda de Scikit-learn Pipelines. Desde la perspectiva de un científico de datos pipeline es un concepto generalizado, pero muy importante. Básicamente, permite el flujo de datos desde su formato bruto a información útil. El funcionamiento de los pipelines se puede entender con la ayuda del siguiente diagrama:

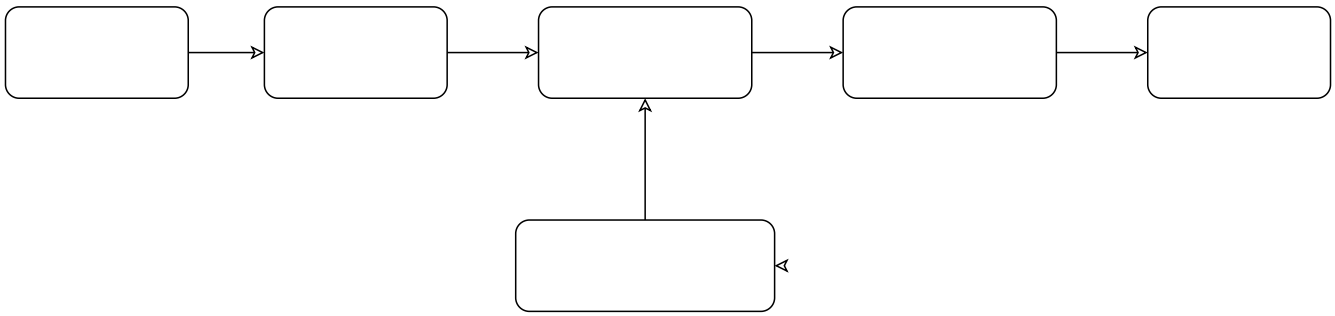

Figura 1: Flujo de trabajo automático.

Los bloques de las pipelines de ML son los siguientes:

- *Ingesta de datos*: Como su nombre indica, es el proceso de importación de los datos para su uso en el proyecto de ML. Los datos pueden extraerse en tiempo real o por lotes de uno o varios sistemas. Es uno de los pasos más complicados porque la calidad de los datos puede afectar a todo el modelo de ML.
- *Preparación de datos*: Después de importar los datos, tenemos que preparar los datos que se utilizarán para nuestro modelo ML. El preprocesamiento de datos es una de las técnicas más relevantes de la preparación de datos.
- *Entrenamiento del modelo ML*: El siguiente paso es entrenar nuestro modelo ML. Tenemos varios algoritmos de ML como supervisado, no supervisado, refuerzo para extraer las características de los datos, y hacer predicciones.
- *Evaluación del modelo*: A continuación, tenemos que evaluar el modelo ML. En el caso de AutoML pipeline, el modelo ML puede ser evaluado con la ayuda de varios métodos estadísticos y reglas de negocio.
- *Reentrenamiento del modelo ML*: En el caso de la canalización AutoML, no es necesario que el primer modelo sea el mejor. El primer modelo se considera un modelo de referencia y podemos entrenarlo repetidamente para aumentar la precisión del modelo.
- *Despliegue*: Por último, tenemos que desplegar el modelo. Este paso implica aplicar y migrar el modelo a las operaciones de negocio para su uso.

Modelización de la cadena ML y preparación de datos

La fuga de datos, que se produce del conjunto de datos de entrenamiento al conjunto de datos de prueba, es un problema importante que los científicos de datos deben abordar al preparar los datos para el modelo de ML. Generalmente, en el momento de la preparación de los datos, el científico de datos utiliza técnicas como la estandarización o la normalización en todo el conjunto de datos antes del aprendizaje. Pero estas técnicas no pueden ayudarnos a evitar la fuga de datos, ya que el conjunto de datos de entrenamiento se habría visto influido por la escala de los datos en el conjunto de datos de prueba.

Mediante el uso de pipelines (tuberías) ML, podemos evitar esta fuga de datos porque las tuberías aseguran que la preparación de datos como la normalización se limita a cada pliegue de nuestro procedimiento de validación cruzada.

Ejemplo

El siguiente es un ejemplo en Python que demuestra la preparación de datos y el flujo de trabajo de evaluación de modelos. Para ello, utilizamos el conjunto de datos *Pima Indian Diabetes* de Sklearn. En primer lugar, vamos a crear un pipeline que estandarice los datos. A continuación, se generará un modelo de análisis discriminatorio lineal y, por último, se evaluará la tubería empleando una validación cruzada de 10 veces.

En primer lugar, importar los paquetes necesarios de la siguiente manera:

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

Luego cargamos el conjunto de datos, como ya hemos hecho previamente:

```
path = "./datasets/pima-indians-diabetes.csv"
headernames = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(path, names=headernames)
X = data.iloc[:,0:-1]
y = data.iloc[:,-1]
```

A continuación, crearemos un pipeline con la ayuda del siguiente código:

```
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('lda', LinearDiscriminantAnalysis()))
model = Pipeline(estimators)
```

Por último, evaluamos esta pipeline y mostramos el accuracy como sigue:

```
kfold = KFold(n_splits=20, random_state=7, shuffle=True)
results = cross_val_score(model, X, y, cv=kfold)
print(results.mean())
```

La salida es la siguiente:

```
0.7759446693657219
```

La salida anterior es el resumen de la exactitud de la configuración en el conjunto de datos.

Modelización de la canalización ML y extracción de características

La fuga de datos también puede ocurrir en el paso de extracción de características del modelo ML. Por ello, los procedimientos de extracción de características también deben restringirse para detener la fuga de datos en nuestro conjunto de datos de entrenamiento. Al igual que en el caso de la preparación de datos, mediante el uso de tuberías de ML, podemos evitar esta fuga de datos

también. *FeatureUnion*, una herramienta proporcionada por ML pipelines puede ser usada para este propósito.

Ejemplo

El siguiente es un ejemplo en Python que demuestra la extracción de características y el flujo de trabajo de evaluación de modelos. Para ello, utilizamos el conjunto de datos Pima Indian Diabetes de Sklearn.

En primer lugar, se extraerán 3 características con PCA (Análisis de Componentes Principales). A continuación, se extraerán 6 características con Análisis Estadístico. Después de la extracción de características, el resultado de la selección de múltiples características y los procedimientos de extracción se combinarán empleando la herramienta *FeatureUnion*. Por último, se creará un modelo de Regresión Logística y se evaluará el pipeline mediante validación cruzada de 10 veces.

En primer lugar, importe los paquetes necesarios como se indica a continuación:

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest
```

A continuación, se creará la unión de rasgos del siguiente modo:

```
features = []
features.append(('pca', PCA(n_components=3)))
features.append(('select_best', SelectKBest(k=6)))
feature_union = FeatureUnion(features)
```

Luego se crea la pipeline con la ayuda del siguiente script:

```
estimators = []
estimators.append(('feature_union', feature_union))
estimators.append(('logistic', LogisticRegression(max_iter=1000)))
model = Pipeline(estimators)
```

Luego evaluamos esta tubería y la salida es la eficiencia:

```
kfold = KFold(n_splits=20, random_state=7, shuffle=True)
results = cross_val_score(model, X, y, cv=kfold)
print(results.mean())
```

```
[Output] 0.7745951417004048
```

Mejora del rendimiento de los modelos ML

Los ensembles pueden mejorar los resultados del aprendizaje automático combinando varios modelos. Básicamente, los modelos ensemble están formados por varios modelos de aprendizaje supervisado entrenados individualmente y sus resultados se combinan de varias maneras para lograr un mejor rendimiento predictivo en comparación con un único modelo. Los métodos ensemble pueden dividirse en los dos grupos siguientes:

- *Métodos de ensemble secuenciales:* Como su nombre indica, en este tipo de métodos de ensemble, los modelos base se generan secuencialmente. El objetivo de estos métodos es explotar la dependencia entre los modelos base.
- *Métodos de conjuntos paralelos:* Como su nombre indica, en este tipo de métodos de conjuntos, los aprendices base se generan en paralelo. El objetivo de estos métodos es explotar la independencia entre los modelos base.

Métodos de aprendizaje por conjuntos

A continuación se enumeran los métodos de aprendizaje por ensamblaje más populares, es decir, los métodos para combinar las predicciones de distintos modelos:

- *Bagging*: El término bagging también se conoce como agregación bootstrap. En los métodos bagging, el modelo ensemble intenta mejorar la precisión de la predicción y reducir la varianza del modelo combinando predicciones de modelos individuales entrenados sobre muestras de entrenamiento generadas aleatoriamente. La predicción final del modelo ensemble vendrá dada por el cálculo de la media de todas las predicciones de los estimadores individuales. Uno de los mejores ejemplos de métodos de ensamblaje son los bosques aleatorios (Random Forest).
- *Boosting*: En el método boosting, el principio fundamental de la construcción de un modelo de conjunto es construirlo de forma incremental mediante el entrenamiento secuencial de cada estimador del modelo base. Como su nombre indica, básicamente combina varios modelos base débiles, entrenados secuencialmente en múltiples iteraciones de datos de entrenamiento, para construir un conjunto potente. Durante el entrenamiento de los aprendices de base débiles, se asignan pesos más altos a aquellos aprendices que fueron clasificados erróneamente con anterioridad. Un ejemplo de método de refuerzo es AdaBoost.
- *Votación:* En este modelo de aprendizaje conjunto, se construyen múltiples modelos de diferentes tipos y se utilizan algunas estadísticas sencillas, como calcular la media o la mediana, etc., para combinar las predicciones. Esta predicción servirá como entrada adicional para el entrenamiento para hacer la predicción final.

Algoritmos de ensamblaje

A continuación se presentan tres algoritmos de ensamblaje bagging:

Bagged Decision Tree:

Los métodos de ensemble bagging funcionan bien con los algoritmos que tienen alta varianza y, en este caso, el mejor es el algoritmo de árbol de decisión. En el siguiente script de Python, vamos a construir un modelo de árbol de decisión ensamblado utilizando la función BaggingClassifier de sklearn con DecisionTreeClasifier (un algoritmo de árboles de clasificación y regresión) en el conjunto de datos “Pima Indians diabetes”.

Primero importamos los paquetes que necesitamos:

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
```

A continuación, dé la entrada para la validación cruzada de 10 veces de la siguiente manera:

```
seed = 7
kfold = KFold(n_splits=10, random_state=seed, shuffle=True)
cart = DecisionTreeClassifier()
```

Tenemos que indicar el número de árboles que vamos a construir. Aquí vamos a construir 150 árboles:

```
num_trees = 150
```

A continuación, construimos el modelo con la ayuda del siguiente script:

```
model = BaggingClassifier(base_estimator=cart, n_estimators=num_trees, random_state=seed)
```

Calcula e imprime los resultados como sigue:

```
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

Salida:

```
0.7630211893369789
```

Random Forest

Es una extensión de los árboles de decisión en bolsas (Bagged Decision Tree). Para los clasificadores individuales, las muestras del conjunto de datos de entrenamiento se toman con reemplazo, pero los árboles se construyen de tal forma que se reduce la correlación entre ellos. Además, se considera un subconjunto aleatorio de características para elegir cada punto de división en lugar de elegir con avidez el mejor punto de división en la construcción de cada árbol.

En el siguiente script de Python, vamos a construir un modelo ensemble de bosque aleatorio embolsado utilizando la clase RandomForestClassifier de sklearn sobre el conjunto de datos de diabetes de los indios Pima.

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

seed = 7
kfold = KFold(n_splits=10, random_state=seed, shuffle=True)

num_trees = 150
max_features = 5

model = RandomForestClassifier(n_estimators=num_trees, max_features=max_features)

results = cross_val_score(model, X, y, cv=kfold)
print(results.mean())
```

Extra Trees

Esta es otra extensión de los ensambles de bagged decision tree. En este método, los árboles aleatorios están construidos a partir de los ejemplos del conjunto de datos de entrenamiento. En el script de Python siguiente vamos a construir un modelo de ensemble de árboles extra, usando la clase de sklearn *ExtreTreesClassifier* en el dataset Pima Indians.

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import ExtraTreesClassifier

seed = 7
kfold = KFold(n_splits=10, random_state=seed, shuffle=True)

num_trees = 150
max_features = 5

model = ExtraTreesClassifier(n_estimators=num_trees, max_features=max_features)

results = cross_val_score(model, X, y, cv=kfold)
print(results.mean())
```

Boosting Ensemble Algorithms

A continuación vamos a mostrar los dos algoritmos más usados de boosting ensemble:

AdaBoost

Es uno de los algoritmos de boosting ensemble con más éxito. La clave principal de este algoritmo reside en la forma en que asigna pesos a las instancias del conjunto de datos. Debido a esto el algoritmo necesita prestar menos atención a las instancias mientras construye los modelos subsiguientes.

En la siguiente receta de Python, vamos a construir un modelo de conjunto Ada Boost para la clasificación utilizando la clase AdaBoostClassifier de sklearn en el conjunto de datos de diabetes de los indios Pima.

En primer lugar, importar los paquetes necesarios de la siguiente manera:

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import AdaBoostClassifier

seed = 5
kfold = KFold(n_splits=10, random_state=seed, shuffle=True)

num_trees = 50
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)

results = cross_val_score(model, X, y, cv=kfold)
print(results.mean())
```

Salida:

```
0.7616883116883117
```

Stochastic Gradient Boosting

Otro algoritmo es Gradient Boosting Machines. En el siguiente script de Python, vamos a construir un modelo Gradient Boosting ensemble estocástico para la clasificación mediante el uso de la clase GradientBoostingClassifier de sklearn en el conjunto de datos de la diabetes de los indios Pima.

En primer lugar, importar los paquetes necesarios de la siguiente manera:

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import GradientBoostingClassifier
```

```
seed = 5
kfold = KFold(n_splits=10, random_state=seed, shuffle=True)

num_trees = 50

model = GradientBoostingClassifier(n_estimators=num_trees, random_state=seed)

results = cross_val_score(model, X, y, cv=kfold)
print(results.mean())
```

Salida:

```
0.7746582365003418
```

La salida arriba muestra que hemos obtenido alrededor de un 77.5% de eficiencia con nuestro modelo ensamblado de clasificación Gradient Boosting.

Voting Ensemble Algorithms

Como se ha comentado, la votación crea primero dos o más modelos independientes a partir del conjunto de datos de entrenamiento y, a continuación, un clasificador de votación envolverá el modelo junto con la toma de la media de las predicciones del submodelo cada vez que se necesiten nuevos datos.

En el siguiente script de Python, vamos a construir un modelo *Voting ensemble* para la clasificación utilizando la clase *VotingClassifier* de sklearn sobre el conjunto de datos de diabetes de los indios Pima. Estamos combinando las predicciones de regresión logística, clasificador de árbol de decisión y SVM juntos para un problema de clasificación de la siguiente manera:

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier

kfold = KFold(n_splits=10, random_state=7, shuffle=True)

estimators = []
model1 = LogisticRegression(max_iter=1000)
estimators.append(('logistic', model1))

model2 = DecisionTreeClassifier()
estimators.append(('cart', model2))

model3 = SVC()
```

```
estimators.append(('svm', model3))

ensemble = VotingClassifier(estimators)

results = cross_val_score(ensemble, X, y, cv=kfold)

print(results.mean())
```

El resultado de este ensemble es:

```
0.7721804511278195
```

Mostrando que se logra un 77% de eficiencia.

Mejora del rendimiento mediante el ajuste de algoritmos

Como sabemos, los modelos ML están parametrizados de tal forma que su comportamiento puede ajustarse para un problema específico. El ajuste del algoritmo significa encontrar la mejor combinación de estos parámetros para mejorar el rendimiento del modelo ML. Este proceso se denomina optimización de hiperparámetros y los parámetros del propio algoritmo se denominan hiperparámetros y coeficientes.

Aquí, vamos a discutir acerca de algunos métodos para el ajuste de parámetros del algoritmo proporcionados por Python Scikit-learn.

Ajuste de los parámetros: Grid Search

Se trata de un método de ajuste de parámetros. El punto clave del funcionamiento de este método es que construye y evalúa el modelo metódicamente para cada posible combinación de parámetros del algoritmo especificados en una cuadrícula. Por lo tanto, podemos decir que este algoritmo tiene naturaleza de búsqueda.

Ejemplo

En el siguiente script de Python, vamos a realizar una búsqueda en cuadrícula utilizando la clase GridSearchCV de sklearn para evaluar varios valores alfa para el algoritmo de regresión Ridge en el conjunto de datos de diabetes de los indios Pima.

```
import numpy
from pandas import read_csv
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

alphas = numpy.array([1,0.1,0.01,0.001,0.0001,0])
param_grid = dict(alpha=alphas)
```

```
model = Ridge()
grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid.fit(X, y)

print(grid.best_score_)
print(grid.best_estimator_.alpha)
```

La salida de este script es el siguiente:

```
0.27610844129292433
1.0
```

La salida anterior nos dice que la evaluación optima para este algoritmo es 0.27 y el parametro que obtuvo este valor fue 1.0.

Ajuste de los parámetros: Random Search

Se trata de un método de ajuste de parámetros. El punto clave del funcionamiento de este método es que muestrea los parámetros del algoritmo a partir de una distribución aleatoria para un número fijo de iteraciones.

Ejemplo

En la siguiente receta de Python, vamos a realizar una búsqueda aleatoria utilizando la clase RandomizedSearchCV de sklearn para evaluar diferentes valores alfa entre 0 y 1 para el algoritmo de regresión Ridge en el conjunto de datos de diabetes de los indios Pima.

```
import numpy
from pandas import read_csv
from scipy.stats import uniform
from sklearn.linear_model import Ridge
from sklearn.model_selection import RandomizedSearchCV

param_grid = {'alpha': uniform()}
model = Ridge()
random_search = RandomizedSearchCV(estimator=model,
param_distributions=param_grid, n_iter=50,
random_state=7)
random_search.fit(X, y)

print(random_search.best_score_)
print(random_search.best_estimator_.alpha)
```

La salida es la siguiente:

```
0.2761075573402853
0.9779895119966027
```

Estos resultados nos muestran valores similares al GridSearch.

Ejercicios

Cree un Jupyter Notebook y en el mismo resuelva los siguientes ejercicios.

1. Cargue el modelo de datos ‘breast-cancer.csv’ y entrene modelos para clasificación usando un BaggingClassifier, RandomForestClassifier y un ExtraTreesClassifier.
2. Para el conjunto de datos 'iris-data.csv', entrene 3 modelos: arboles de decisión, KNN, y una red neuronal de simple capa. Luego haga los siguientes incisos.
   a) Cree un ensemble usando Boosting Ensemble de estos tres modelos y evalúe su rendimiento.
   b) Cree un ensemble usando un modelo de votación.
   c) Mejore los rendimientos de cada uno de los algoritmos usando ajuste de parámetros por GridSearch y RandomSearch. Diga en cada caso cuál fue el mejor ajuste para los parámetros.

Elaborado por: M.Sc. Angel Alberto Vazquez Sánchez
Profesor Auxiliar del Departamento de Inteligencia Computacional.