xgb

WeizhouS

3/21/2019

Supervised learning

- Ranking: predicting an ordering on a set of choices
- Recommendation: recommending an item to a user based on consumption history and profile

XGBoost

- Speed and performance
- Core algorithm is parallelizable
- Consistently outperforms single-algorithm methods
- Satte-of-the-art performance in many ML tasks

Boosting

- Not a specific machine learning algorithm
- Concept that can be applied to a set of machine learning models: "Meta-algorithm"
- Ensemble meta-algorithm used to convert many weak learners into a strong leraner

Cross-validation

- Robust method for estimating the performance of a model on unseen data
- Generates many non-overlapping train/test splits on training data
- Reports the average test set performance across all data splits

When to NOT use XGBoost

- Image recognition
- Computer vision
- Natural language processing and understanding problems

Common regression metrics

- RMSE
- MAE

Common loss function in XGBoost

- `reg:linear` use for regression problems
- `reg:logistic` use for classification problems, output decision
- `binary:logistic` use when you want probability rather

Base learners

- When combined, the final prediction is non-linear
- Good at distinguishing or predicting *part* of the dataset
- Linear base learner rarely used
- Tree base learner almost exclusively used in XGBoost

Regularization

- Control on complexity
- `gamma`: minimum loss reduction required for a split to occur
- `alpha`: L1 regularization on leaf weights
- `lambda`: L2 regularization on leaf weights

Common tree tunable parametes

- learning rate: `eta`
- `gamma`
- `lambda`
- `alpha`
- `max_depth`
- `subsample`: % samples used per tree
- `colsample_bytree`: % features used per tree
- number of estimators

Grid search

- Exhaustively

Random search

- Parameter space to explore can be massive

Pipeline

- Takes a list of named 2-tuples (name, pipeline_step) as input
- Tuples can contain any arbitrary scikit-learn compatible estimator or transformer object
- Pipeline implements fit/predict methods
- Can be used as input estimator into grid/randomized serach and cross_val_score methods

Preprocessing I: LabelEncoder and OneHotEncoder Preprocessing II: DictVectorizer

- Used in text processing
- Converts lists of feature mappings into vectors
- Convert dataframe into a list of dictionary entries

More to come

- XGBoost for ranking/recommendation problems
- Using more sophisticated hyperparameter tuning strategies for tuning XGBoost models (Bayesian Optimization)

```python
# Import necessary modules
from sklearn_pandas import DataFrameMapper
from sklearn_pandas import CategoricalImputer

# Check number of nulls in each feature column
nulls_per_column = X.isnull().sum()
print(nulls_per_column)

# Create a boolean mask for categorical columns
categorical_feature_mask = X.dtypes == object

# Get list of categorical column names
categorical_columns = X.columns[categorical_feature_mask].tolist()

# Get list of non-categorical column names
non_categorical_columns = X.columns[~categorical_feature_mask].tolist()

# Apply numeric imputer
numeric_imputation_mapper = DataFrameMapper([([numeric_feature], Imputer(strategy="median")) for   nume

# Apply categorical imputer
categorical_imputation_mapper = DataFrameMapper([(category_feature, Imputer()) for category_feature in
# Import FeatureUnion
from sklearn.pipeline import FeatureUnion

# Combine the numeric and categorical transformations
numeric_categorical_union = FeatureUnion([("num_mapper", numeric_imputation_mapper),
                                          ("cat_mapper", categorical_imputation_mapper)])

# Create full pipeline
pipeline = Pipeline([("featureunion", numeric_categorical_union),
                     ("dictifier", Dictifier()),
                     ("vectorizer", DictVectorizer(sort=False)),
                     ("clf", xgb.XGBClassifier(max_depth=3))])

# Perform cross-validation
cross_val_scores = cross_val_score(pipeline, kidney_data, y, scoring="roc_auc", cv=3)

# Print avg. AUC
print("3-fold AUC: ", np.mean(cross_val_scores))

# Create the parameter grid
gbm_param_grid = {
    'clf__learning_rate': np.arange(0.05, 1.0, 0.05),
    'clf__max_depth': np.arange(3, 10, 1),
    'clf__n_estimators': np.arange(50, 200, 50)
}

# Perform RandomizedSearchCV
randomized_roc_auc = RandomizedSearchCV(estimator=pipeline, param_distributions=gbm_param_grid, n_iter=

# Fit the estimator
randomized_roc_auc.fit(X, y)

# Compute metrics
print(randomized_roc_auc.best_score_)
print(randomized_roc_auc.best_estimator_)
```