GC-02

Satellite Image Land Classification

Author: Raymond Martin

Abstract

Satellite image land classification has several applications. Examples include land-use planning, disaster monitoring, forestry, ecological protection, and agriculture.
For this project, the Deepsat SAT-6 Airborne Dataset was selected in order to explore Apache Spark's performance when applied to multiclass land classification tasks.
From those methods considered, Random Forest produces the highest accuracy (88.2% when only 1000 samples are trained).

Dataset

-The dataset is image data (pixels) from satellite photos of US territory.
-The six classifiction categories are: barren_land, trees, water, grassland, buildings, and road.
-Pixels have four bands: RGB and infrared.
-The images were originally much larger, and they have all been normalized to 28x28 pixels. Therefore, there are 28x28x4 = 3136 features.
-Dataset is quite large overall: 324,000 training samples and 80,000 test samples.
-The data is pre-randomized.

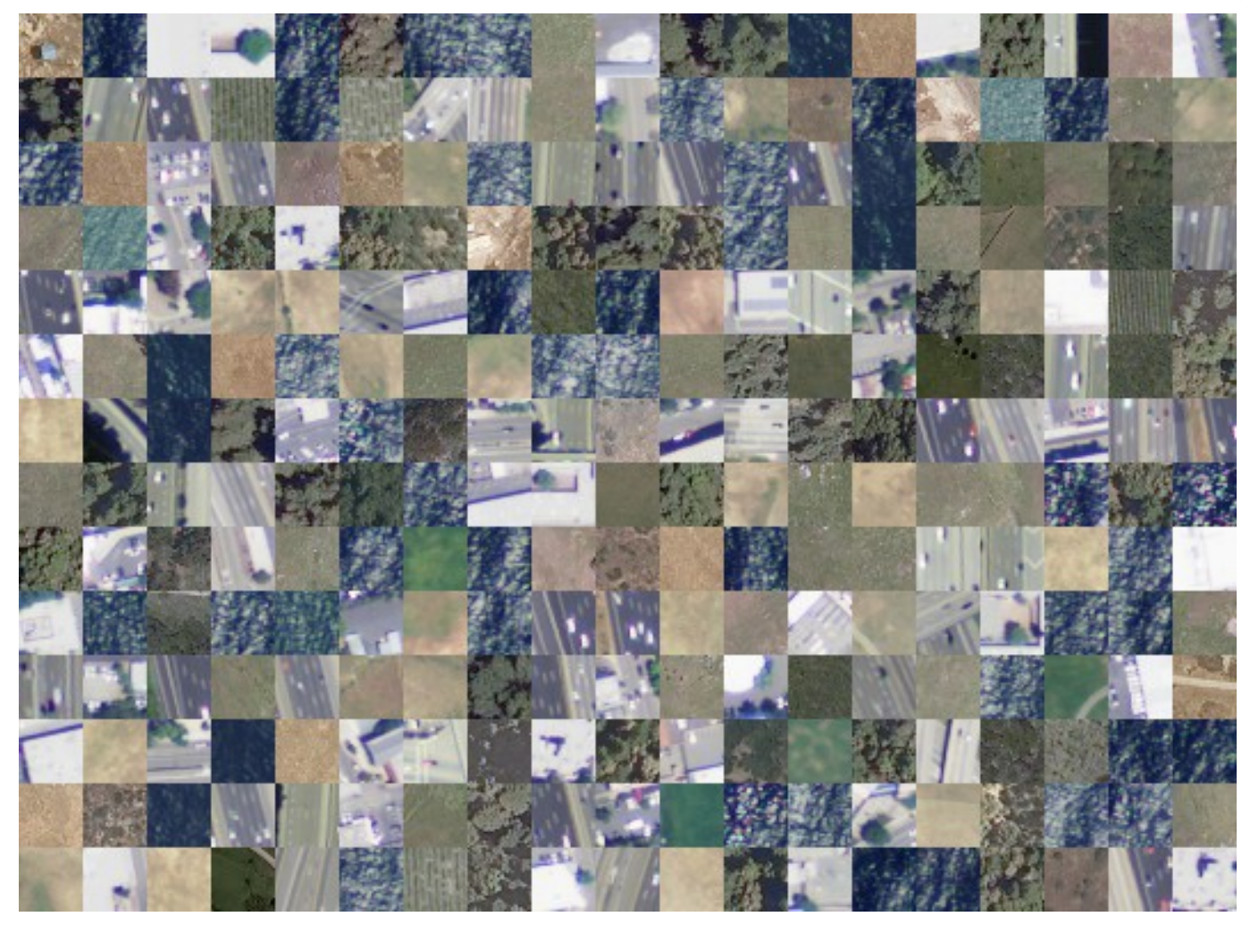
Task

-How to best classify land types based off satellite images using the only the Spark framework with a pyspark.ml dataframe-based approach
-To this end, I consider several multiclass classification methods: **Decision Trees**, **Random Forest**, **Naïve Bayes Classifier**, and **Multinomal Logistic Regression (Softmax)**

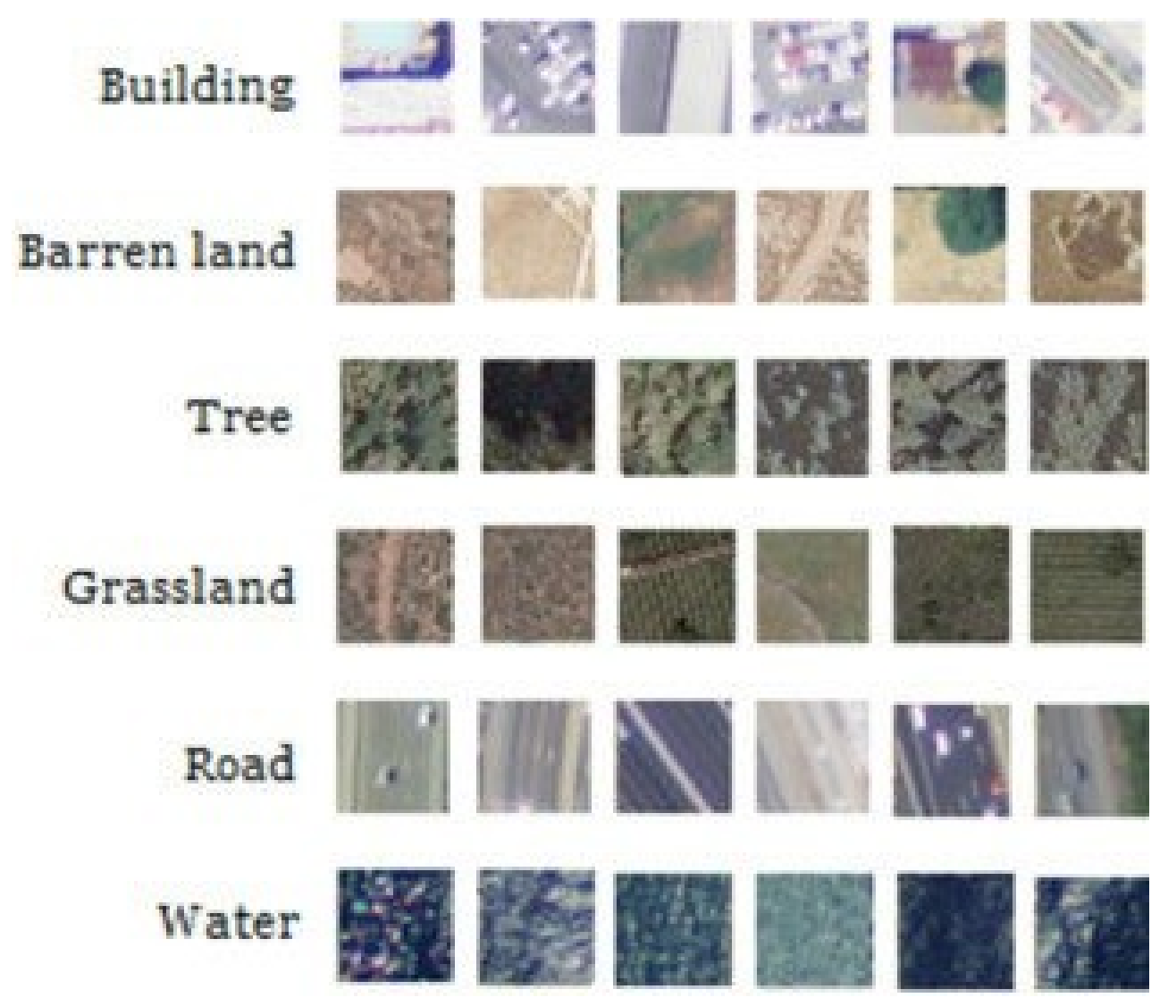
Data Preprocessing

-The dataset consists of 404,000 image samples, each with 3136 features. The labels are one-hot encoded into six classes. Due to the gigantic memory requirements of this large dataset, it was necessary to use only a subset of the data for training.
-Pyspark.ml methods require that all features be assembled into a single vector. I used Vector Assembler function to select all the 3136 feature columns and transform them.
-The one-hot encoded labels did not fit directly into pyspark.ml methods. I had to convert them to the correct format by collecting them into a vector and recreating the labels dataframes.
-I found it necessary to temporarily convert the data to a Pandas dataframe (from a spark dataframe) in order to properly concatenate two dataframes. A SQL-style join strategy on the two either eliminated rows, duplicated them, or created NULL values.

Sample Code and Results



```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.linalg import Vectors
import numpy as np
import pandas as pd
from pyspark.sql.functions import *
from pyspark.sql.types import IntegerType
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

#Take in raw data:
y_train_df = spark.read.csv("y_train_sat6.csv", inferSchema=True).limit(1000)
X_train_df = spark.read.csv("X_train_sat6.csv", inferSchema=True).limit(1000)

#Convert spark dataframe to correct format for machine learning methods:
y_train_array = np.array(y_train_df.select('_c0', '_c1', '_c2', '_c3', '_c4', '_c5').collect())
y_train_df = spark.createDataFrame(y_train_array, IntegerType())
y_train_df = y_train_df.select(col("value").alias("label"))

#Concatenate the labels and features into a single dataframe:
y_train_pandas = y_train_df.toPandas()
X_train_pandas = X_train_df.toPandas()
train_pandas = pd.concat([y_train_pandas, X_train_pandas], axis = 1)
train_df = spark.createDataFrame(train_pandas)

#Use VectorAssembler to map all features to a single vector “features”
assembler = VectorAssembler(inputCols=[x for x in X_train_df.columns], outputCol="features")
train_df = assembler.transform(train_df)

#The above preprocessing was also performed for test data (not shown here)

evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")

#Create Naïve Bayes model and evaluate:
from pyspark.ml.classification import NaiveBayes
nb = NaiveBayes(smoothing=1.0, modelType="multinomial")
model = nb.fit(train_df)
predictions = model.transform(test_df)
nbaccuracy = evaluator.evaluate(predictions)

#Create Decision Tree model and evaluate:
from pyspark.ml.classification import DecisionTreeClassifier
dt = DecisionTreeClassifier(maxDepth=3, labelCol=“label“, impurity=“gini”)
model = dt.fit(train_df)
predictions = model.transform(test_df)
dtaccuracy = evaluator.evaluate(predictions)

#Random Forest:
from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(labelCol="label", featuresCol="features", numTrees=50)
model = rf.fit(train_df)
predictions = model.transform(test_df)
rfaccuracy = evaluator.evaluate(predictions)

#Softmax:
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(maxIter=15, regParam=0.2, elasticNetParam=0.7)
model = lr.fit(train_df)
predictions = model.transform(test_df)
lraccuracy = evaluator.evaluate(predictions)
```



Results

#Accuracy, 500 training samples:

- Naïve Bayes: 76.4%
- Decision Tree: 81.2%
- Random Forest: 86.4%
- Softmax: 56%

#Accuracy, 1000 training samples:

- Naïve Bayes: 79.8%
- Decision Tree: 81.4%
- Random Forest: 88.2%
- Softmax: 58.8%

#Accuracy, 2000 training samples:

- Naïve Bayes: 80.4%
- Decision Tree: Out of memory!

Comments

-Because of the high dimensionality of the data, memory requirements are very high. This proved especially true when I attempted to implement a feedforward neural network. The neural networks I built (pyspark.ml's Multilayer Perceptron Classifier) were either too simplistic to be effective (predicted Class 5 for all test samples) or too complex (crash/timeout).

-Undoubtedly a CNN would likely produce the highest accuracy, especially for pattern (image) data with a high number of samples such as this dataset.

-This was my first time using Apache Spark, and the majority of my work was getting the data correctly processed. Once processed correctly, I found implementation of pyspark.ml classification methods to be simple.

-The final report for this project will perform additional testing and cross validation in order to increase confidence of results.

Conclusion

-Even before cross validation is applied to validate accuracy results, it seems clear enough that Random Forest is the winner amongst simple (non-neural net) classifier methods for this dataset.
-Multinomial logistic regression alone seems to provide poor results.
-Pyspark.ml is still new. It could improve by adding concatenate functions (similar to pandas.concat) to avoid difficulties arising from SQL-style joins.

Acknowledgments

Professor: Dr. Mingon Kang

Contact Information

Email: raymondpmartin@gmail.com

References

Dataset available at:
https://www.kaggle.com/crawford/deepsat-sat6

Kennesaw State University
College of Computing and Software Engineering