Enzyme Classifier - SparkML

July 26, 2019

Team Members

- Shubhendu Vimal - 11915067
- Dharani Kiran Kavuri - 11915033
- Anmol More - 11915043

ReadMe :

- Data Preparation through python script
- Convert raw data csv and then to libsvm in R
- Run spark ML algos in jupyter notebook

```
In [1]: import pandas as pd
        import glob
        from pyspark.ml.classification import LogisticRegression
        from pyspark.ml import Pipeline
        from pyspark.ml.classification import DecisionTreeClassifier
        from pyspark.ml.classification import RandomForestClassifier
        from pyspark.ml.classification import MultilayerPerceptronClassifier, OneVsRest
        from pyspark.ml.feature import IndexToString, StringIndexer, VectorIndexer
        from pyspark.ml.evaluation import MulticlassClassificationEvaluator
        from pyspark.sql import SparkSession
```

Read data
Read cleaned and processed data for enzymes from data directory

```
In [2]: data = pd.DataFrame()
        files = glob.glob("data/*.csv")
        for file in files :
            print("Processing File :", file)
            df = pd.read_csv(file)
            print(df.shape)
            data = data.append(df, ignore_index=True)


        def type_to_numeric(x):
            if x=='uniprot-Isomerase':
```

```
            return 0
        if x=='uniprot-Lyase':
            return 1
        if x=='uniprot-ligase':
            return 2
        if x=='uniprot-Transferases' :
            return 3
        if x=='uiprot-Hydrolases' :
            return 4
        if x=='uniprot-Oxidoreductases' :
            return 5

data['type_numeric'] = data['Type'].apply(type_to_numeric)

#Final csv to work upon and run spark ML algorithms
#Taking 100000 records of each enzyme
data.to_csv("final_data/protien-sequences.csv", index=False)
data.sample(5)
```

```
Processing File : data/uniprot-Oxidoreductases.csv
(89604, 24)
Processing File : data/uniprot-Isomerase.csv
(100001, 24)
Processing File : data/uniprot-Hydrolases.csv
(100001, 24)
Processing File : data/uniprot-ligase.csv
(100001, 24)
Processing File : data/uniprot-Lyase.csv
(100001, 24)
Processing File : data/uniprot-Transferases.csv
(100001, 24)
```

```
Out[2]:           A    C     D     E     F     G    H     I     K     L  ...     S  \
        334374   2.0  0.0   5.0  12.0   2.0   7.0  1.0  10.0   7.0  10.0  ...   7.0
        447198  22.0  1.0   7.0   7.0   6.0  14.0  8.0  10.0   0.0  14.0  ...   6.0
        329103  18.0  3.0  12.0  15.0  11.0  21.0  5.0  19.0  16.0  33.0  ...  24.0
        215190  31.0  7.0  11.0  19.0  10.0  14.0  8.0  11.0   5.0  35.0  ...  14.0
        92571   22.0  4.0  19.0  20.0  10.0  20.0  8.0  15.0  22.0  25.0  ...  17.0

                                                         Sequence     T  \
        334374  MGRTDDMLIIRGVNVFPSQIESVLLENGDTTPHYQLIVNRKGNLDD...   5.0
        447198  MPHRILVLHGPNLNLLGTREPEVYGRTTLADIDAALTAQAQTAGAE...   9.0
        329103  MVHELLQQAKWRIIDQSHFGPMFDAKQSFAIDDTLCTSVGKGLSDP...  13.0
        215190  MFDIGVNLTSTQFAKDRDKVVKRAREAGISGMLITGTNALESQQAL...  14.0
        92571   MKLSFNTWVYNSFPSMLPFYPLEEVISRIAAFGYDGIEIGCASPHA...   9.0

                                 Type     V    W     Y     gravy      weight  type_numeric
```

```
334374     uniprot-ligase  10.0  0.0   1.0 -0.410577  11844.5304              2
447198      uniprot-Lyase  11.0  1.0   5.0  0.094595  15959.8291              1
329103     uniprot-ligase  21.0  4.0  13.0 -0.012950  31174.5785              2
215190  uiprot-Hydrolases  13.0  5.0   2.0 -0.122394  28765.5914              4
92571   uniprot-Isomerase  15.0  8.0  14.0 -0.422857  31818.8477              0

[5 rows x 25 columns]
```

R Code
Convert csv data to libsvm for sparkML

```
In [3]: %load_ext rpy2.ipython
```

```
In [4]: %%R
        # Ref : https://vatsalbits.wordpress.com/2016/01/13/csv-to-libsvm/

        # download e1071 library if not available
        if (!require(e1071)) {install.packages("e1071")}

        # download sparseM library if not available
        if (!require(SparseM)) {install.packages("SparseM")}

        # load the libraries
        library(e1071)
        library(SparseM)

        # load the csv dataset into memory
        data <- read.csv('final_data/protien-sequences.csv')

        # take the numeric columns are format as matrix
        x <- as.matrix(data[,c('A', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L', 'M',
                               'N', 'P', 'Q', 'R', 'S', 'T', 'V', 'W', 'Y',
                               'gravy', 'weight')])
        # assign labels to vector y
        y <- data[,'type_numeric']

        # convert input columns to sparse matrix
        x_matrix <- as.matrix.csr(x)

        # write output to libsvm format
        write.matrix.csr(x_matrix, y=y, file="final_data/protien-sequences-libsvm.txt")
```

```
R[write to console]: Loading required package: e1071

R[write to console]: Loading required package: SparseM

R[write to console]:
Attaching package: SparseM
```

```
R[write to console]: The following object is masked from package:base:

    backsolve
```

```
In [5]: #create a new spark session
        MAX_MEMORY = "5g"
        spark = SparkSession.builder.appName('protien-classifier').
        config("spark.executor.memory", MAX_MEMORY).
        config("spark.driver.memory", MAX_MEMORY).getOrCreate()
```

```
In [6]: # Load training data
        data = spark.read.format("libsvm").load("final_data/protien-sequences-libsvm.txt")
```

Run Spark-MLlib Algorithms
Logistic Regression Classifier
Code chunks are directly taken from official spark-ml documentation.
Ref - https://spark.apache.org/docs/latest/ml-classification-regression.html

```
In [7]: %%time
        #split dataset in test and training sets
        (train, test) = data.randomSplit([0.7, 0.3])

        #use max iteration of 10, and few default parameters
        logistic_reg = LogisticRegression(maxIter=20, regParam=0.005,
                                          elasticNetParam=0.005)

        # Fit the model
        logic_reg_model = logistic_reg.fit(train)

        # coefficients and intercept for multi class logistic regression
        print("Logistic Reg Coefficients: " + str(logic_reg_model.coefficientMatrix))
        print("Logistic Reg Intercept: " + str(logic_reg_model.interceptVector))

        # Print model summary
        model_summary = logic_reg_model.summary

        accuracy = str(model_summary.accuracy)
        false_positive_rate = str(model_summary.weightedFalsePositiveRate)
        true_positive_rate = str(model_summary.weightedTruePositiveRate)
        f_measure = str(model_summary.weightedFMeasure())
        precision = str(model_summary.weightedPrecision)
        recall = str(model_summary.weightedRecall)

        print("Model accuracy : " + accuracy +
              "\nFalse positive rate : " + false_positive_rate +
              "\nTrue positive rate : " + true_positive_rate +
              "\nF-measure : " + f_measure +
              "\nPrecision : " + precision +
              "\nRecall : " + recall)

Logistic Reg Coefficients: DenseMatrix([[ 5.45202991e-03, -8.41721306e-02,  1.47792590e-02,
               3.31893586e-03, -3.73312494e-04, -9.73336894e-04,
              -1.93253095e-02, -5.61266461e-03,  2.89474262e-02,
               3.58842586e-03,  6.87395544e-04, -1.34753590e-03,
              -2.70238535e-03,  1.24997026e-02,  3.13580444e-04,
              -6.59144958e-03,  1.79010417e-03, -1.75218553e-03,
              -1.02059722e-01, -1.99632503e-02, -8.86660848e-01,
               8.33571514e-07],
             [ 8.85022716e-03, -1.18602161e-02,  2.83949155e-03,
              -4.75651902e-03, -1.67761010e-02,  3.44992374e-03,
              -2.00237860e-02, -3.42373036e-03, -1.04233788e-02,
               1.00570350e-03,  9.45213109e-03,  2.87557424e-02,
              -0.00000000e+00,  4.38811521e-03, -3.17804872e-03,
               7.87092740e-03,  1.46965982e-02, -3.95256967e-03,
              -3.74618000e-02, -7.53803470e-03,  1.78657822e-01,
               1.01338335e-06],
             [-5.44382414e-04, -3.65995056e-02,  1.58249042e-02,
               2.90505556e-02,  1.97166963e-02, -2.58209170e-03,
              -2.72139077e-02, -3.92640394e-03,  1.39468558e-02,
               6.41660339e-03, -4.90823743e-03, -1.22397775e-02,
              -2.10386699e-03, -2.55006639e-03,  3.58658471e-04,
              -9.47427993e-03, -3.44740078e-03, -1.08598272e-03,
               2.32355489e-02,  1.23598495e-02, -1.95403376e-01,
              -1.24076148e-06],
             [-2.03297917e-02,  1.21864343e-02, -1.58565824e-02,
              -2.09937573e-02, -1.88554774e-02,  6.02781399e-03,
               2.04237107e-02,  8.34704605e-02,  2.49272055e-03,
              -3.20490486e-02, -6.43147253e-02,  7.89723454e-03,
              -7.28943716e-03, -9.92204577e-03,  2.17382198e-03,
              -7.46535750e-04, -1.25944695e-02,  2.74750585e-02,
               1.58983343e-02, -0.00000000e+00,  1.12359426e+00,
              -3.18344785e-07],
             [-2.45574514e-03,  2.79102499e-02,  1.62842341e-03,
              -4.20275874e-03,  3.90743498e-03, -2.62320748e-03,
               6.86609432e-02, -6.35138103e-03, -4.75216675e-03,
               1.00420738e-03,  1.78526063e-02, -2.75581708e-03,
               7.65353346e-04, -5.89824624e-03, -3.73230530e-04,
               1.84585536e-02,  1.37432514e-03, -7.31473093e-03,
              -4.08326154e-03,  1.26975951e-02, -5.71198888e-02,
               4.24723852e-07],
             [-7.61661945e-04,  1.22353827e-01, -1.39996583e-02,
              -9.55126011e-03,  8.01425588e-03, -3.50855993e-03,
               8.79739093e-03, -1.08732097e-02, -9.43516283e-03,
               0.00000000e+00,  1.20510455e-03, -7.56656920e-03,
               7.07526911e-03,  6.41561717e-03, -7.95823268e-04,
               2.42710127e-04, -4.77668365e-03, -6.11424855e-03,
               3.65983003e-02,  6.71549501e-04, -3.34873437e-01,
              -6.77072976e-07]])
Logistic Reg Intercept: [0.17462809241234534,0.010481053067311924,-0.6559807909752295,0.5357280
Model accuracy : 0.36205831255588106
False positive rate : 0.12731148458569602
True positive rate : 0.36205831255588106
F-measure : 0.3611527231912624
Precision : 0.3628963004678636
Recall : 0.36205831255588106
CPU times: user 21.3 ms, sys: 6.57 ms, total: 27.8 ms
Wall time: 12.6 s
```

Decision Tree Classifier

```
In [8]: %%time
        #mark labels as index of libsvm data
        label_index = StringIndexer(inputCol="label", outputCol="index_label").fit(data)

        # Mark all input columns as features
        feature_index = VectorIndexer(inputCol="features",
                                      outputCol="index_features",
                                      maxCategories=2).fit(data)

        # Lets do 60:40 ratio for training and test data
        (train, test) = data.randomSplit([0.6, 0.4])

        # Make a decision tree model
        decision_tree_classifier = DecisionTreeClassifier(labelCol="index_label",
                                                          featuresCol="index_features")

        # Create a pipeline of decision tree model
        pipeline = Pipeline(stages=[label_index, feature_index, decision_tree_classifier])

        # train our decision tree using input columns
        decision_tree_model = pipeline.fit(train)

        model_summary = decision_tree_model.stages[2]
        # summary only
        print("Created decision tree classifier with : \n")
        print(model_summary)

        # Get predictions for test set
        predictions_test = decision_tree_model.transform(test)
        # Look at top 10 predictions, 9/10 predictions seems correct
        predictions_test.select("prediction", "index_label", "features").show(10)

        # calculate accurancy on test set
        multi_class_classification_evaluator = MulticlassClassificationEvaluator(
            labelCol="index_label", predictionCol="prediction", metricName="accuracy")
        accuracy_percentage = multi_class_classification_evaluator.evaluate(predictions_test)

        print("Test set error = %f " % (1.0 - accuracy_percentage))
```

```
Created decision tree classifier with :

DecisionTreeClassificationModel (uid=DecisionTreeClassifier_62c948558ae8) of depth 5 with 51 n
+----------+-----------+--------------------+
|prediction|index_label|            features|
+----------+-----------+--------------------+
|       5.0|        5.0|(22,[0,1,2,3,4,5,...|
|       5.0|        5.0|(22,[0,1,2,3,4,5,...|
|       0.0|        5.0|(22,[0,1,2,3,4,5,...|
|       3.0|        5.0|(22,[0,1,2,3,4,5,...|
|       5.0|        5.0|(22,[0,1,2,3,4,5,...|
|       5.0|        5.0|(22,[0,1,2,3,4,5,...|
|       5.0|        5.0|(22,[0,1,2,3,4,5,...|
|       5.0|        5.0|(22,[0,1,2,3,4,5,...|
|       3.0|        5.0|(22,[0,1,2,3,4,5,...|
|       3.0|        5.0|(22,[0,1,2,3,4,5,...|
+----------+-----------+--------------------+
only showing top 10 rows

Test set error = 0.651715
CPU times: user 41.3 ms, sys: 9.75 ms, total: 51.1 ms
Wall time: 13.2 s
```

Random Forest Classifier
As per our evaluation, this gives us best result with accuracy of 57% with 30 trees
We can further work on this to increase number of trees, max depth and other hyper parameter tunings

```
In [9]: %%time
        #mark labels as index of libsvm data
        label_index = StringIndexer(inputCol="label", outputCol="index_label").fit(data)

        # Mark all input columns as features
        feature_index = VectorIndexer(inputCol="features", outputCol="index_features",
                                      maxCategories=2).fit(data)

        # Lets do 70:30 ratio for training and test data this time
```

```
(train, test) = data.randomSplit([0.7, 0.3])

# Create a random forest classifier with 20 trees
random_forest_classifier = RandomForestClassifier(labelCol="index_label",
                                                  featuresCol="index_features",
                                                  numTrees=30, maxDepth=10)

# label converter to convert indexed labels
label_converter = IndexToString(inputCol="prediction", outputCol="predicted_label",
                               labels=label_index.labels)

# Create a pipeline of random forest model
pipeline = Pipeline(stages=[label_index, feature_index,
                            random_forest_classifier, label_converter])

# Train random forest classifier
random_forest_model = pipeline.fit(train)

model_summary = random_forest_model.stages[2]
# summary only
print("Created random forest classifier with : \n")
print(model_summary)

# Get predictions for test set
predictions_test = random_forest_model.transform(test)

# see few of our predictions
predictions_test.select("predicted_label", "label", "features").show(10)

# Calculate test set error
multi_class_classification_evaluator = MulticlassClassificationEvaluator(
    labelCol="index_label", predictionCol="prediction", metricName="accuracy")
accuracy_percentage = multi_class_classification_evaluator.evaluate(predictions_test)

print("Test set error = %f " % (1.0 - accuracy_percentage))
```

```
Created random forest classifier with :

RandomForestClassificationModel (uid=RandomForestClassifier_5f69f240db35) with 30 trees
+---------------+-----+--------------------+
|predicted_label|label|            features|
+---------------+-----+--------------------+
|            5.0|  5.0|(22,[0,1,2,3,4,5,...|
|            5.0|  5.0|(22,[0,1,2,3,4,5,...|
|            5.0|  5.0|(22,[0,1,2,3,4,5,...|
|            5.0|  5.0|(22,[0,1,2,3,4,5,...|
|            5.0|  5.0|(22,[0,1,2,3,4,5,...|
|            5.0|  5.0|(22,[0,1,2,3,4,5,...|
|            5.0|  5.0|(22,[0,1,2,3,4,5,...|
|            5.0|  5.0|(22,[0,1,2,3,4,5,...|
|            5.0|  5.0|(22,[0,1,2,3,4,5,...|
|            5.0|  5.0|(22,[0,1,2,3,4,5,...|
+---------------+-----+--------------------+
only showing top 10 rows

Test set error = 0.436635
CPU times: user 46 ms, sys: 11 ms, total: 57 ms
Wall time: 35.3 s
```

One-vs-Rest classifier (a.k.a. One-vs-All)
Create classifier with multiple binary classifiers and combine
Increasing number of iterations in this case doesn't affects accuracy beyond 39%

```
In [10]: %%time
         #split train and test set in 80:20 ratio
         (train, test) = data.randomSplit([0.8, 0.2])

         # instantiate logistic regression classifier
         logistic_regression_classifier = LogisticRegression(maxIter=30,
                                                             tol=1E-7, fitIntercept=True)

         # initializer our one vs rest classifier
         one_vs_rest = OneVsRest(classifier=logistic_regression_classifier)

         # train model on training data
         one_vs_rest_model = one_vs_rest.fit(train)

         # get predictions on test data
         test_predictions = one_vs_rest_model.transform(test)

         # Test accuracy using multi class evaluator
         multi_class_evaluator = MulticlassClassificationEvaluator(metricName="accuracy")
         accuracy_percentage = multi_class_evaluator.evaluate(test_predictions)
         print("Test set error = %f " % (1.0 - accuracy_percentage))
```

```
Test set error = 0.614291
CPU times: user 256 ms, sys: 58 ms, total: 314 ms
Wall time: 43.2 s
```

Feedforward Artificial Neural Network Classifier

- References :
- https://dzone.com/articles/deep-learning-via-multilayer-perceptron-classifier
- https://spark.apache.org/docs/2.2.1/api/java/index.html?org/apache/spark/ml/classification/Random

Neural network test test accuracy calculation has some issues with library

```
In [12]: %%time
         #split data in 60:40 ratio
         random_split = data.randomSplit([0.6, 0.4])
         train = random_split[0]
         test = random_split[1]

         # specify input, middle, output layers
         # No of input layers - 20
         # intermediate layers - 15, 10, 8
         # output layers - 3
         layers = [20, 15, 10, 8, 6]

         # create multilayer neural network classifier
         nn_model = MultilayerPerceptronClassifier(maxIter=100,
                                                   layers=layers, blockSize=128)

         # train our neural network
         model = nn_model.fit(train)

         # compute accuracy on the test set
         test_predictions = model.transform(test)

         predictionAndLabels = test_predictions.select("prediction", "label")
```

```
CPU times: user 31.3 ms, sys: 6.53 ms, total: 37.9 ms
Wall time: 28.7 s
```

```python
# Team Members
# - Shubhendu Vimal - 11915067
# - Dharani Kiran Kavuri - 11915033
# - Anmol More - 11915043

#Data Preparation in plain Python

#Read raw fasta protien sequence files and convert it to parsable CSV files using biopython

#Embedded block of code for converting to pdf only, run separately (time taking)

import sys
import pandas as pd
from Bio.SeqUtils.ProtParam import ProteinAnalysis

#arg 1 - fasta file name
#arg 2 - no of records to process
file = str(sys.argv[1])
limit = int(sys.argv[2])
print("Processing file : ", file)

df = pd.DataFrame()
enzyme_type = file[:-6]

# read fasta files, line by line and process enzyme sequence
print("Group all as : ", enzyme_type)
with open(file) as fileobject:
    start = True
    row = ""
    for line in fileobject:
        if("sp|" in line or "tr|" in line) :
            start = True
            if(row != "") :
                row_dict = {}
                row_dict["Sequence"] = row
                row_dict["Type"] = enzyme_type

                try :
                    #use biopython library to process enzyme sequence
                    analysed_seq = ProteinAnalysis(row)
                    #print(analysed_seq)
                    amino_acid_counts = analysed_seq.count_amino_acids()
                    row_dict.update(amino_acid_counts)
                    analysed_seq = ProteinAnalysis(row)
                    row_dict["weight"] = analysed_seq.molecular_weight()
                    row_dict["gravy"] = analysed_seq.gravy()
                    df = df.append(row_dict, ignore_index=True)
                    print(df.shape)
                    if(df.shape[0] > limit) :
                        break
                except :
                    print("Error")
                row = ""
        elif(start) :
            row += line.rstrip()
print(df.head())
df.to_csv("data/" + enzyme_type + ".csv", index=False)
```