Large-Scale Lasso and Elastic-Net Regularized Generalized Linear Models

DB Tsai

Steven Hillion
Outline

- Introduction
- Linear / Nonlinear Classification
- Feature Engineering - Polynomial Expansion
- Big-data Elastic-Net Regularized Linear Models
Introduction

– Classification is an important and common problem
  • Churn analysis, fraud detection, etc.…even product recommendations
– Many observations and variables, non-linear relationships
– Non-linear and non-parametric models are popular solutions, but they are slow and difficult to interpret
– Our solution
  • Automated feature generation with polynomial mappings
  • Regularized regressions with various performance optimizations
Logistic Regression on Spark

Model: Accuracy: 91.63%

Results - Status:
- Submitting the Spark Sequoia Forest job.
- Spark Alpine Forest: Output Path: /tmp/alpine_runtime/shillion/Logistic_Regression_on_Spark_1282/8a0ec37-964a-4557-9df7-529893731a69-14339396135218
- Spark: Logistic Regression: application_14339396135218_0316
- Spark: Logistic Regression: treeAggregate at RDDFunctions.scala:71 - 9
- Spark: Logistic Regression: reduce at DistinctValueCounter.scala:49 - 0
- Spark: Logistic Regression: treeAggregate at StandardScaler.scala:52 - 1
- Spark: Logistic Regression: take at LogisticRegression.scala:77 - 2
- Spark: Logistic Regression: count at OWLQN.scala:54 - 3
- Spark: Logistic Regression: take at OWLQN.scala:58 - 4
- Spark: Logistic Regression: treeAggregate at OWLQN.scala:128 - 6
- Spark: Logistic Regression: treeAggregate at OWLQN.scala:128 - 7
Linear / Nonlinear Classification

- Linear: In the data’s **original** input space, labels can be classified by a linear decision boundary.

- Nonlinear: The classifiers have nonlinear, and possibly discontinuous decision boundaries.
Linear Classifier Examples

- Logistic Regression
- Support Vector Machine
- Naive Bayes Classifier
- Linear Discriminant Analysis
Nonlinear Classifier Examples

- Kernel Support Vector Machine
- Multi-Layer Neural Networks
- Decision Tree / Random Forest
- Gradient Boosted Decision Trees
- K-nearest Neighbors Algorithm
Feature Engineering

\[(x'_1, x'_2) = (x_1, x_1x_2)\]

- Decision Boundary in Transformed Space
  \[x'_2 = 0\]

- Decision Boundary in Original Space
  \[x'_1x'_2 = 0\]
Feature Engineering

Ref: https://youtu.be/3liCbRZPrZA
Low-Degree Polynomial Mappings

- 2nd Order Example:
  \[
  \phi(x) = [1, x_1, \ldots, x_n, x_1^2, \ldots, x_n^2, x_1x_2, \ldots, x_{n-1}x_n]^T.
  \]

- The dimension of d-degree polynomial mappings
  \[
  C(n + d, d) = \frac{(n + d)(n + d - 1) \cdots (n + 1)}{d!}
  \]

- C.J. Lin, et al., Training and Testing Low-degree Polynomial Data Mappings via Linear SVM, JMLR, 2010
2-Degree Polynomial Mapping

- 2-Degree Polynomial Mapping:
  \# of features = \( O(n^2) \) for one training sample
- 2-Degree Polynomial Kernel Method:
  \# of features = \( O(nl) \) for one training sample
- \( n \) is the dimension of original training sample,
  \( l \) is the number of training samples.
- In typical setting, \( l >> n \).
- For sparse data, \( \bar{n} \) is the average \# non-zeros,
  \( O(\bar{n}^2) << O(n^2) ; \ O(\bar{n}^2) << O(nl) \)
Kernel Methods vs Polynomial Mapping

Logistic Regression (linear classification model)

\[ p(y|x) = \frac{1}{1 + e^{-yw^T x}} \]

Kernel Logistic Regression (nonlinear model)

\[ p(y|x) = \frac{1}{1 + e^{-yf(x)}} \]

\[ f(x) = \alpha_0 + \sum_{i=1}^{N} \alpha_i K(x_i, x) \]
Cover’s Theorem

A complex pattern-classification problem, cast in a high-dimensional space nonlinearly, is more likely to be linearly separable than in a low-dimensional space, provided that the space is not densely populated.

— Cover, T.M. , Geometrical and Statistical properties of systems of linear inequalities with applications in pattern recognition., 1965
Logistic Regression & Overfitting

- Given a decision boundary, or a hyperplane $ax_1 + bx_2 + c = 0$

- Distance from a sample to hyperplane $d = \frac{ax_1 + bx_2 + cx_0}{\sqrt{a^2 + b^2}}$ where $x_0 = 1$

$$w_0 = \frac{c}{\sqrt{a^2 + b^2}} z$$

$$w_1 = \frac{a}{\sqrt{a^2 + b^2}} z$$

$$w_2 = \frac{b}{\sqrt{a^2 + b^2}} z$$

Same classification result but more sensitive probability

$$P(y=1 | \vec{x}, \vec{w}) = \frac{\exp(zd)}{1 + \exp(zd)} = \frac{\exp(\vec{x} \cdot \vec{w})}{1 + \exp(\vec{x} \cdot \vec{w})}$$
Finding Hyperplane

- Maximum Likelihood Estimation: From a training dataset
  \[ X = (\vec{x}_1, \vec{x}_2, \vec{x}_3, \ldots) \quad Y = (y_1, y_2, y_3, \ldots) \]

- We want to find \( \vec{w} \) that maximizes the likelihood of data

  \[ L(\vec{w}, \vec{x}_1, \ldots, \vec{x}_N) = P(y_1|\vec{x}_1, \vec{w}) P(y_2|\vec{x}_2, \vec{w}) \cdots P(y_N|\vec{x}_N, \vec{w}) \]

- With linear separable dataset, likelihood can always be increased with the same hyperplane by multiplying a constant into weights which resulting steeper curve in logistic function.

- This can be addressed by regularization to reduce model complexity which increases the accuracy of prediction on unseen data.
Training Logistic Regression

– Converting the product to summation by taking the natural logarithm of likelihood will be more convenient to work with.

– The negative log-likelihood will be our loss function

\[
l(\tilde{w}, \tilde{x}) = - \sum_{k=1}^{N} \log P(y_k|\tilde{x}_k, \tilde{w})
\]

\[
= - \sum_{k=1}^{N} y_k \log P(y_k=1|\tilde{x}_k, \tilde{w}) + (1-y_k) \log P(y_k=0|\tilde{x}_k, \tilde{w})
\]

\[
= - \sum_{k=1}^{N} y_k \log \frac{\exp(\tilde{x}_k \tilde{w})}{1+\exp(\tilde{x}_k \tilde{w})} + (1-y_k) \log \frac{1}{1+\exp(\tilde{x}_k \tilde{w})}
\]

\[
= - \sum_{k=1}^{N} y_k \tilde{x}_k \tilde{w} - \log(1+\exp(\tilde{x}_k \tilde{w}))
\]
Regularization

– The loss function becomes

\[ l_{total}(\vec{w}, \vec{x}) = l_{model}(\vec{w}, \vec{x}) + l_{reg}(\vec{w}) \]

– The loss function of regularization doesn’t depend on data.

– Common regularizations are
  - L2 Regularization:
    \[ l_{reg}(\vec{w}) = \lambda \sum_{i=1}^{N} w_i^2 \]
  - L1 Regularization:
    \[ l_{reg}(\vec{w}) = \lambda \sum_{i=1}^{N} |w_i| \]
  - Elastic-Net Regularization:
    \[ l_{reg}(\vec{w}) = \lambda \sum_{i=1}^{N} \left( \frac{\alpha}{2} w_i^2 + (1-\alpha)|w_i| \right) \]
Geometric Interpretation

- The ellipses indicate the posterior distribution for no regularization.

- The solid areas show the constraints due to regularization.

- The corners of the L1 regularization create more opportunities for the solution to have zeros for some of the weights.
Intuitive Interpretation

– L2 penalizes the square of weights resulting very strong “force” pushing down big weights into tiny ones. For small weights, the “force” will be very small.

– L1 penalizes their absolute value resulting smaller “force” compared with L2 when weights are large. For smaller weights, the “force” will be stronger than L2 which drives small weights to zero.

– Combining L1 and L2 penalties are called Elastic-Net method which tends to give a result in between.
Optimization

- We want to minimize loss function
  \[ l_{\text{total}}(\tilde{w}, \tilde{x}) = l_{\text{model}}(\tilde{w}, \tilde{x}) + l_{\text{reg}}(\tilde{w}) \]

- First Order Minimizer - require loss, gradient vector of loss
  - Gradient Descent \( \tilde{w}_{n+1} = \tilde{w}_n - \gamma \tilde{G} \), \( \gamma \) is learning rate
  - L-BFGS (Limited-memory BFGS)
  - OWLQN (Orthant-Wise Limited-memory Quasi-Newton) for L1
  - Coordinate Descent

- Second Order Minimizer - require loss, gradient, hessian matrix of loss
  - Newton-Raphson, quadratic convergence which is fast!
    \[ \tilde{w}_{n+1} = \tilde{w}_n - H^{-1} \tilde{G} \]

Issue of Second Order Minimizer

- Scale horizontally (the numbers of training data) by leveraging on Spark to parallelize this iterative optimization process.

- Don't scale vertically (the numbers of training features).

- Dimension of Hessian Matrix: $dim(H) = n^2$

- Recent applications from document classification and computational linguistics are of this type.
Apache Spark Logistic Regression

- The total loss and total gradient have two parts; model part depends on data while regularization part doesn’t depend on data.

\[ l_{total}(\mathbf{w}, \mathbf{x}) = l_{model}(\mathbf{w}, \mathbf{x}) + l_{reg}(\mathbf{w}) \]

\[ \tilde{G}(\mathbf{w}, \mathbf{x})_{total} = \tilde{G}(\mathbf{w}, \mathbf{x})_{model} + \tilde{G}(\mathbf{w})_{reg} \]

- The loss and gradient of each sample is independent.

\[ (\tilde{G}(\mathbf{w}, \mathbf{x})_{model})_i = G_i(\mathbf{w}, \mathbf{x}) = \frac{\partial l(\mathbf{w}, \mathbf{x})}{\partial w_i} = \sum_{k=1}^{N} y_k x_{ki} - \frac{\exp(x_k \mathbf{w})}{1 + \exp(x_k \mathbf{w})} x_{ki} \]

\[ l(\mathbf{w}, \mathbf{x})_{model} = -\sum_{k=1}^{N} y_k x_k \mathbf{w} - \log(1 + \exp(\mathbf{x}_k \mathbf{w})) \]
Apache Spark Logistic Regression

- Compute the loss and gradient in parallel in executors/workers; reduce them to get the lossSum and gradientSum in driver/controller.

- Since regularization doesn’t depend on data, the loss and gradient sum are added after distributed computation in driver.

- Optimization is done in single machine in driver; L1 regularization is handled by OWLQN optimizer.
Apache Spark Logistic Regression

Loop until converge

Initialize Weights

Broadcast Weights to Executors

Driver/Controller

Compute loss and gradient for each sample, and sum them up locally

Driver/Controller

Reduce from executors to get lossSum and gradientSum

Handle regularization and use LBFGS/OWLQN to find next step

Final Model Weights

Drivers/Workers

Compute loss and gradient for each sample, and sum them up locally

Drivers/Workers

Compute loss and gradient for each sample, and sum them up locally

Drivers/Controllers
Apache Spark Linear Models

- [SPARK-5253] Linear Regression with Elastic Net (L1/L2)
- [SPARK-7262] Binary Logistic Regression with Elastic Net
  - Author: DB Tsai, merged in Spark 1.4
  - Internally handle feature scaling to improve convergence and avoid penalizing too much on those features with low variances
  - Solutions exactly match R’s glmnet but with scalability
  - For LiR, the intercept is computed using close form like R
  - For LoR, clever initial weights are used for faster convergence

- [SPARK-5894] Feature Polynomial Mapping
  - Author: Xusen Yin, merged in Spark 1.4
Convergence: a9a dataset

Logistic Regression with a9a Dataset (11M rows, 123 features, 11% non-zero elements)
16 executors in INTEL Xeon E3-1230v3 32GB Memory * 5 nodes Hadoop 2.0.5 alpha cluster
Convergence: news20 dataset

Logistic Regression with news20 Dataset (0.14M rows, 1,355,191 features, 0.034% non-zero elements)
16 executors in INTEL Xeon E3-1230v3 32GB Memory * 5 nodes Hadoop 2.0.5 alpha cluster
Convergence: rcv1 dataset

Logistic Regression with rcv1 Dataset (6.8M rows, 677,399 features, 0.15% non-zero elements)
15 executors in INTEL Xeon E3-1230v3 32GB Memory * 5 nodes Hadoop 2.0.5 alpha cluster

Spark Summit 2015
Polynomial Mapping Experiment

– New Spark ML Pipeline APIs allows us to construct the experiment very easily.
– *StringIndexer* for converting a string of labels into label indices used in algorithms.
– *PolynomialExpansion* for mapping the features into high dimensional space.
– *LogisticRegression* for training large scale Logistic Regression with L1/L2 Elastic-Net regularization.
alphaParam.foreach { alpha =>
  regParam.foreach { reg =>

    val stages = new mutable ArrayBuffer[PipelineStage]()

    val labelIndexer = new StringIndexer()
      .setInputCol("labelString")
      .setOutputCol("indexedLabel")
    stages += labelIndexer

    val polynomialExpansion = new PolynomialExpansion()
      .setInputCol("features")
      .setOutputCol("polyFeatures")
      .setDegree(2)
    stages += polynomialExpansion

    val lor = new LogisticRegression()
      .setFeaturesCol("polyFeatures")
      .setLabelCol("indexedLabel")
      .setRegParam(reg)
      .setElasticNetParam(alpha)
      .setMaxIter(params.maxIter)
      .setTol(params.tol)
    stages += lor

    val pipeline = new Pipeline().setStages(stages.toArray)
    val pipelineModel = pipeline.fit(training)
    val trainAcc = evaluateClassificationModel(pipelineModel, training, "indexedLabel")
    val testAcc = evaluateClassificationModel(pipelineModel, test, "indexedLabel")

    println(s"$trainAcc\t$testAcc\t$reg\t$alpha")
  }
}
Datasets

- a9a, ijcnn1, and webspam datasets are used in the experiment.

<table>
<thead>
<tr>
<th>Data set</th>
<th>$n$</th>
<th>$\bar{n}$</th>
<th>$l$</th>
<th># testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>a9a</td>
<td>123</td>
<td>13.9</td>
<td>32,561</td>
<td>16,281</td>
</tr>
<tr>
<td>real-sim</td>
<td>20,958</td>
<td>51.5</td>
<td>57,848</td>
<td>14,461</td>
</tr>
<tr>
<td>news20</td>
<td>1,355,181</td>
<td>455.5</td>
<td>15,997</td>
<td>3,999</td>
</tr>
<tr>
<td>ijcnn1</td>
<td>22</td>
<td>13.0</td>
<td>49,990</td>
<td>91,701</td>
</tr>
<tr>
<td>MNIST38</td>
<td>752</td>
<td>168.2</td>
<td>11,982</td>
<td>1,984</td>
</tr>
<tr>
<td>covtype</td>
<td>54</td>
<td>11.9</td>
<td>464,810</td>
<td>116,202</td>
</tr>
<tr>
<td>webspam</td>
<td>254</td>
<td>85.1</td>
<td>280,000</td>
<td>70,000</td>
</tr>
</tbody>
</table>
Logistic Regression with Polynomial Mapping: webspam

- **Training, alpha = 0.0**
- **Testing, alpha = 0.0**
- **Training, alpha = 0.5**
- **Testing, alpha = 0.5**
- **Training, alpha = 1.0**
- **Testing, alpha = 1.0**

Accuracy vs. regParm (lambda)
## Comparison

<table>
<thead>
<tr>
<th>Test Accuracy</th>
<th>Linear SVM</th>
<th>Linear SVM Degree-2 Polynomial</th>
<th>SVM RBF Kernel</th>
<th>Logistic Regression</th>
<th>Logistic Regression Degree-2 Polynomial</th>
</tr>
</thead>
<tbody>
<tr>
<td>a9a</td>
<td>84.98</td>
<td>85.06</td>
<td>85.03</td>
<td>85.0</td>
<td>85.26</td>
</tr>
<tr>
<td>ijcnn1</td>
<td>92.21</td>
<td>97.84</td>
<td>98.69</td>
<td>92.0</td>
<td>97.74</td>
</tr>
<tr>
<td>webspam</td>
<td>93.15</td>
<td>98.44</td>
<td>99.20</td>
<td>92.76</td>
<td>98.57</td>
</tr>
</tbody>
</table>

The results of Linear and Kernel SVM experiment are from C.J. Lin, et al., *Training and Testing Low-degree Polynomial Data Mappings via Linear SVM*, JMLR, 2010
Conclusion

- For some problems, linear methods with feature engineering are as good as nonlinear kernel methods.
- However, the training and scoring are much faster for linear methods.
- For problems of document classification with sparsity, or high dimensional classification, linear methods usually perform well.
- With Elastic-Net, sparse models get be trained, and the models are easier to interpret.
Thank you!

Questions?