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

DB Tsai **NETFLIX**

Steven Hillion Alpine

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



← → C f salesdemo.alpinenow.com/#work_flows/1282									
■ Alpine Q Search					Advi	sor No v	~ 0		Steven 👻
Logistic Regression on Spark *	Ru	in Stop	Clear	Sav	ve Re	vert	Close	A	ctions 🔻
OPERATORS DATA <					[00:22:55]	Analytic	c Flow fin	ished	/iew status
All Operators Random Sampli	ng Test								
conf									
RECENT	Logistic Regression								
ALL OPERATORS Confusion Matrix Training	Confusion Matr	rix							
Summary statistics	Alpine Forest								
LOGISTIC REGRESSION ?	Edit Operator Cop	y Paste	Re	name	Delet	е	Step Rur	n E:	xplore 🔻
Results - Status									. .
[00:18:35] Submitting the Spark Sequora Forest job. [00:18:35] Spark Alpine Forest Output Path : /tmp/alpine_runtime/shillion/Logistic_Regression_on_Spark_1282/8 [00:18:35] Alpine Forest:check jars trying to upload jars to Hadoop Cluster if needed	10eca37-9d4a-4557-9df7-529903731a49-1433996315218/ Mo	del:	0 1	2	3 pred	dicted	6 7	8	9
[00:18:35] Logistic Regression started running [00:18:35] Logistic Regression:check jars trying to upload jars to Hadoop Cluster if needed [00:18:37] Alpine Forest : application_143354350695_0316	ACC	0 0	0.97 0.0	0 0.00	0.00 0.00	0.00	0.01 0.0	0 0.00	0.00
Spark: [00:18:37] Logistic Regression : application_1433354350695_0317		1	0.00 0.9	7 0.00	0.00 0.00	0.00	0.00 0.0	0 0.01	0.00
Spark: [00:19:16] Logistic Regression : treeAggregate at RDDFunctions.scala:71 - 0		2	0.01 0.0	1 0.90	0.01 0.01	0.00	0.01 0.03	2 0.02	0.00
Spark: [00:19:16] Alpine Forest : reduce at DistinctValueCounter.scala:49 - 0 Snark:		3	0.01 0.0	1 0.02	0.88 0.00	0.04	0.00 0.0	1 0.02	0.01
[00:19:25] Logistic Regression : treeAggregate at StandardScaler.scala:52 - 1 Snark:		D: 4	0.00 0.0	1 0.00	0.00 0.93	0.00	0.01 0.0	0 0.01	0.03
[00:19:36] Logistic Regression : take at LogisticRegression.scala:77 - 2 Spark:		bserve	0.00 0.0		0.00 0.00	0.00	0.01 0.0		0.03
[00:19:44] Logistic Regression : count at OWLQN.scala:54 - 3 Spark:		0 5	0.01 0.0	1 0.01	0.03 0.01	0.88	0.02 0.0	1 0.02	0.01
[00:19:51] Logistic Regression : take at OWLQN.scala:58 - 4 Spark:		6	0.01 0.0	0 0.01	0.00 0.01	0.01	0.96 0.0	0.00	0.00
[00:19:52] Logistic Regression : treeAggregate at OWLQN.scala:128 - 5 Spark:		7	0.00 0.0	1 0.01	0.00 0.01	0.00	0.00 0.9	3 0.00	0.03
[00:19:55] Logistic Regression : treeAggregate at OWLQN.scala:128 - 6 Spark:		8	0.01 0.03	3 0.01	0.02 0.01	0.03	0.01 0.0	0.85	0.02
[00:19:57] Logistic Regression : treeAggregate at OWLQN.scala:128 - 7			0.01 0.0	1 0.00	0.02 0.04	0.01	0.00 0.0	3 0.01	0.88

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

 $(x_{1}, x_{2}) = (x_{1}, x_{1}x_{2})$

Decision Boundary in Transformed Space

 $x_{2}^{'}=0$

Decision Boundary in Original Space

$$x_{1}^{'}x_{2}^{'}=0$$





Feature Engineering







Ref: https://youtu.be/3liCbRZPrZA

Low-Degree Polynomial Mappings

– 2nd Order Example:

$$\phi(\boldsymbol{x}) = [1, x_1, \dots, x_n, x_1^2, \dots, x_n^2, x_1 x_2, \dots, x_{n-1} x_n]^T.$$

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, I >> n.
- For sparse data, <u>n</u> is the average # non-zeros, $O(\underline{n}^2) << O(\underline{n}^2)$; $O(\underline{n}^2) << O(\underline{nl})$

Kernel Methods vs Polynomial Mapping

Logistic Regression (linear classification model)

$$p(y|\mathbf{x}) = \frac{1}{1 + e^{-y\mathbf{w}^T\mathbf{x}}}$$

Kernel Logistic Regression (non linear model)

$$p(y|\mathbf{x}) = \frac{1}{1 + e^{-yf(\mathbf{x})}}$$
$$f(\mathbf{x}) = \alpha_0 + \sum_{i=1}^N \alpha_i K(\mathbf{x}_i, \mathbf{x}_i)$$



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 $ax_1 + bx_2 + c = 0$



Finding Hyperplane

Maximum Likelihood Estimation: From a training dataset

$$X = (\vec{x}_1, \vec{x}_2, \vec{x}_3, ...) \quad Y = (y_{1}, y_{2}, y_{3}, ...)$$

– We want to find \vec{w} that maximizes the likelihood of data

$$L(\vec{w}, \vec{x_{1}}, ..., \vec{x_{N}}) = P(y_{1}|\vec{x_{1}}, \vec{w})P(y_{2}|\vec{x_{2}}, \vec{w})...P(y_{N}|\vec{x_{N}}, \vec{w})$$

- 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

$$\begin{split} l(\vec{w}, \vec{x}) &= -\sum_{k=1}^{N} \log P(y_k | \vec{x}_k, \vec{w}) \\ &= -\sum_{k=1}^{N} y_k \log P(y_k = 1 | \vec{x}_k, \vec{w}) + (1 - y_k) \log P(y_k = 0 | \vec{x}_k, \vec{w}) \\ &= -\sum_{k=1}^{N} y_k \log \frac{\exp(\vec{x}_k \vec{w})}{1 + \exp(\vec{x}_k \vec{w})} + (1 - y_k) \log \frac{1}{1 + \exp(\vec{x}_k \vec{w})} \\ &= -\sum_{k=1}^{N} y_k \vec{x}_k \vec{w} - \log(1 + \exp(\vec{x}_k \vec{w})) \end{split}$$



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 $l_{reg}(\vec{w}) = \lambda \sum_{i}^{N} w_{i}^{2}$
 - L2 Regularization:

- 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 $l_{total}(\vec{w}, \vec{x}) = l_{model}(\vec{w}, \vec{x}) + l_{reg}(\vec{w})$
- First Order Minimizer require loss, gradient vector of loss
 - Gradient Descent $\vec{w}_{n+1} = \vec{w}_n \gamma \vec{G}$, γ 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!

$$\vec{w}_{n+1} = \vec{w}_n - H^{-1}\vec{G}$$



Ref: Journal of Machine Learning Research 11 (2010) 3183-3234, Chih-Jen Lin et al.

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 part; model part depends on data while regularization part doesn't depend on data.

$$l_{total}(\vec{w}, \vec{x}) = l_{model}(\vec{w}, \vec{x}) + l_{reg}(\vec{w})$$
$$\vec{G}(\vec{w}, \vec{x})_{total} = \vec{G}(\vec{w}, \vec{x})_{model} + \vec{G}(\vec{w})_{reg}$$

- The loss and gradient of each sample is independent.

$$(\vec{G}(\vec{w},\vec{x})_{model})_{i} = G_{i}(\vec{w},\vec{x}) = \frac{\partial l(\vec{w},\vec{x})}{\partial w_{i}} = \sum_{k=1}^{N} y_{k} x_{ki} - \frac{\exp(\vec{x}_{k}\vec{w})}{1 + \exp(\vec{x}_{k}\vec{w})} x_{ki}$$
$$l(\vec{w},\vec{x})_{model} = -\sum_{k=1}^{N} y_{k} \vec{x}_{k} \vec{w} - \log(1 + \exp(\vec{x}_{k}\vec{w}))$$



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



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



Seconds

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) 16 executors in INTEL Xeon E3-1230v3 32GB Memory * 5 nodes Hadoop 2.0.5 alpha cluster





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.



```
alphaParam.<u>foreach</u> { 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")
```

```
Spork
summit 2015
```

Datasets

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

Data set	n	$ar{n}$	l	# testing
a9a	123	13.9	$32,\!561$	$16,\!281$
real-sim	$20,\!958$	51.5	$57,\!848$	$14,\!461$
news20	$1,\!355,\!181$	455.5	$15,\!997$	$3,\!999$
ijcnn1	22	13.0	$49,\!990$	91,701
MNIST38	752	168.2	$11,\!982$	$1,\!984$
covtype	54	11.9	$464,\!810$	$116,\!202$
webspam	254	85.1	280,000	70,000





Accuracy

regParm (lambda)



Logistic Regression with Polynomial Mapping: a9a



Accuracy



regParm (lambda)

Logistic Regression: ijcnn1 — Training, alpha = 0.93 0.0 ---- Testing, alpha = 0.0 0.92 — Training, alpha = 0.5 ---- Testing, alpha = 0.92 0.5 Training, alpha = 1.0 0.91 ---- Testing, alpha = 1.0 0.91 0.9 0.1 0 0.01 1

regParm (lambda)

Accuracy



Logistic Regression with Polynomial Mapping: ijcnn1



regParm (lambda)

Accuracy



Logistic Regression: webspam

Accuracy

Soc

summit 2015



regParm (lambda)

Logistic Regression with Polynomial Mapping: webspam



regParm (lambda)

summit 2015

Comparison

Test Accuracy	Linear SVM	Linear SVM Degree-2 Polynomial	SVM RBF Kernel	Logistic Regression	Logistic Regression Degree-2 Polynomial
a9a	84.98	85.06	85.03	85.0	85.26
ijcnn1	92.21	97.84	98.69	92.0	97.74
webspam	93.15	98.44	99.20	92.76	98.57



 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.



Thank you!

Questions?

