TITLE

Hongda Li

March 19, 2021

Abstract

...

1 Introduction and Overview

2 Theoretical Background

In this section we will go through several of the popular machine learning models, dimensionality reduction techniques and measurements of performance metric. Major part of the discussion will centered around the mathematics and idea behind these subjects. The presentation is made with the hope to be informative about the mathematical background of the project, and how the mathematical understandings of lead to a better understanding of the structure and topology of the data via training on validating Machine Learning Models. Our hypothesis is that, the performance Neuro Net works, or Convolutional Neuro Net work is as good as classical models if not better. In addition, we hypothesize the existence of Irreducible errors among the EMNIST data set.

2.1 Data Standardization

Data standardization is used for classical training method. This is necessary because models such as the SVM, LDA, PCA, and Decision Tree has biases depending on the size of the samples for each of the classes. Hence, all the data is Standardized by flattening the image of the data, and for all features of each sample is set to zero mean and unit variance. In addition, for training SVM and Decision tree, there are exact same number of samples presented for each of the labels.

2.2 Dimensionality Reduction Using Fisher's LDA

Principal Component Analysis and Linear Discriminant Analysis are both used together to create low dimensional embeddings of the feature space. The original feature space for each of the images lie in a 784 dimension because each pixel of the image is considered as a feature and the image has size 28×28.

LDA is both a classifier and a dimensionality reduction technique, here we introduce Fisher's LDA, the idea is to look for a subspace (Not necessary orthogonal) such that, the embeddings of all the samples in the training set has maximal spread and minimal deviance among all the classes labels. One a high level, the problem is phrased as:

$$\max_{\|x\|=1} \left\{ \frac{v^T S_{\text{Between}} v}{v^T S_{\text{Within}} v} \right\} \tag{1}$$

Where 2 of the matrices are defined as:

$$S_{\text{between}} = \sum_{i=1}^{k} \left((u_i - \mu)(u_i - \mu)^T \right) \tag{2}$$

$$S_{\text{within}} = \sum_{i=1}^{k} \left(\sum_{x_i \in C_i} (x_i - m_i)(x_i - m_i)^T \right)$$

Where quantity μ_i denotes the average of the class C_i projected onto the chosen subspace v, and the quantity m_i is the centroid of all the samples for the class C_i:

$$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} v^T x_i \qquad \mu = \frac{1}{k} \sum_{i=1}^{k} \mu_i \tag{3}$$

$$m_i = \sum_{x_i \in C_i} x_i \tag{4}$$

Notice that equation Expression 1 is the key here, the solution to the optimization problem is Eigenvector of the maximal Eigenvalue for the matrix $S^{\dagger}_{\text{within}} S_{\text{between}}$, however, the solution is not unique, all Eigenvector can be projected onto, and this is the subspace of embeddings for the LDA.

Notice that taking the inverse of the matrix to be full rank, and LDA can only project the feature space down to $\min(k, n) - 1$ where k is the number of classes and n is the total number of the samples.

However, the drawback for includes the fact that it requires the classes of the labels to have Gaussian Distributions of the same covariance matrix for each of the class, in addition, it cannot separate non-linear distributions.

PCA Dimensionality Reduction

PCA: Principal Component Analysis is an unsupervised learning technique and it's usually used as a dimensionality reduction technique. The Singular Value decomposition decompose any matrix A into the product of 3 matrices: $U\Sigma V^T$, where U, V^T are orthogonal matrices and Σ is a diagonal matrix with Singular Values on the diagonal.

In practice, matrix A is the data matrix and the lower dimensional embedding are the ΣV^T is the A matrix is the data matrix and $U\Sigma$ if the A matrix is a row matrix.

PCA is very different for one reason, it supports non-linearity and it's based on the idea of explaining variance in the data set. And nothing can demonstrate this idea better than the Low-rank Approximation theorem:

$$U_k \Sigma_k V_k^T = \underset{\text{rank}(X)=k}{\text{argmin}} \left\{ \|X - A\|_F^2 \right\} \tag{5}$$

Where the, the k truncated singular value decomposition of matrix A is the solution to the k rank approximation problem under the Frobenius Norm (And it measures the variance across differences between these 2 matrices).

This is incredibly useful in a sense that, if the variance of the data set can explained by some kind of low rank structure, then those structure will be review in the Principal Components, the U, or the V matrix.

Principal Components is very useful for describing the data in an orthogonal subspace that is much lower than the original features space where the data is located int. The idea is so powerful that, Randomized SVD and Sparse SVD algorithm is implemented in the Sklearn Library.

However, compare to the Fisher's LDA embeddings, PCA is lossy once the truncated SVD is used. In addition, SVD is sensitive to noise and unitary transformation of the original data (Preservation of Inner products). Rotated samples are very likely to create more singular values. Here we take the conservative approach of choosing 90% or more energy for singular value by defult.

LDA + PCA for Dimensionality Reduction and Data Mining

Our idea is combine both PCA and LDA for dimensionality reduction. We use PCA and LDA to first visualize the separations of data using the lower dimensional embeddings to get some ideas about the structure of the data. Classical Machine Learning models such as the SVM and LDA have complexity that scale with the number of features, samples, and labels involved. Hence creating an embeddings in a lower dimension is extremely beneficial to the running time of the algorithms of these classical models.

To get the best out of both algorithm PCA is performed on images partition by labels into equally sized classes and then LDA is performed on the PCA embeddings of the data, this is done using the training set. This is can be easily carried out using *sklearn*

When making prediction with a model, test set is projected onto the PCA components and then transformed into the LDA sub-space before feeding into the model for prediction.

SVM

The idea behind SVM: Support Vector Machine is simple and powerful. The problem is formulated as a Quadratic Optimization problem, summarized as the following:

$$\min_{\beta,\beta_0} \left\{ \frac{1}{2} \|\beta\|^2 + C \sum_{i=1}^{N} \xi_i \right\} \text{ s.t: } \begin{cases} y_i(x_i^T \beta + \beta_0) \geq 1 - \xi_i & \forall i \\ \xi_i \geq 0, \sum_i (\xi_i) \leq 1 - \xi_i & \forall i \end{cases} \tag{6}$$

The quantity β, β_0 are parameters that define the separating hyperplane, and the slack variables ξ_i are used as a penalty for placing the samples on the wrong side of the hyperplane. In practice, the problem is put into a dual primal format and then optimization with the KTT (Karush-Kuhn-Tucker conditions) are used.

And, buy consider the Fenchel Transform on the primal objective of the function (Which is itself), we obtain the Lagrangian of the system in the following form:

$$\mathcal{L}(\beta, \beta_0, \xi, \alpha) = \frac{1}{2}\|\beta\|^2 + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i [y_i(x_i^{\beta} + \beta_0) - (1 - \xi_i)] - \sum_{i=1}^{N} \mu_i \xi_i \tag{7}$$

And the constraints of the primal problem is relaxed using positivity assumptions on the variables: $\alpha_i, \mu_i, \xi_i \geq 0 \; \forall i$.

Much more theoretical Perspective of this part is highlighted in the book *Elements of Statistical Learning*, where taking the derivative on the Lagrangian wrt the primal variable (β_i, ξ_i) will yield the results that:

$$\begin{aligned} \beta &= \sum_{i=1}^{N} \alpha_i y_i x_i \\ 0 &= \sum_{i=1}^{N} \alpha_i y_i \\ \alpha_i &= C - \mu_i \quad \forall i \end{aligned} \tag{8}$$

Often in practice, much noble algorithms are involved in solving the SVM optimal solutions and most of them are put into the dual form because it has a connection to the Kernel Function and inner products o all the features, and it give rise to the use of the Kernel function.

In practice, SVM is very effective under the task of binary classification under small sample but much worse for when then features space is in a very high dimension, and for multi-class classification where many labels are involved. This is the case because of the complexity of the algorithm both dimension and the number of samples used for training are very large, and the complexity is given by $\mathcal{O}(mn^2)$ where n denotes the number of samples and m denotes the number of features.

To overcome, we use the dimensionality technique introduced previous to improve the speed of the SVM. In addition, SVM serves as a great tools for looking into the structure of the classification and highlighting potential difficulties for other machine learning model, it serves as a "Lower Bound" for how well other machine learning model can perform. Finally, via experiments, it's a model that hardly over-fit and under-fit is indicative of potential irreducible errors on the data.

Decision Tree and Random Forest

The idea behind decision tree is yet, another simple and powerful idea. There are 2 type of decision tree, regression tree where it seeks to approximate a function and the classification tree for classifying the samples with labels.

Under the most basic premise, each node of the of the tree, it split the samples into 2 sets trying to maximizes the information gain when splitting. The entropy of the tree can be measured by the expression:

$$E(p) = \sum_{i=1}^{k} -p_i \log(p_i) \tag{9}$$

Where the quantity p_i denotes the portion of samples with label i and k denotes the total number of samples in the current node of the tree. Observe that the function reaches maximum when one of the labels saturated the node of the tree.

To make a branch on the tree, a separation criterion is made on one of the feature, and it seeks to maximizes the information gain, which can be summarized as one expressions:

$$E(\text{Parent}) - \frac{N_{<k}}{N} E(F_1 < k) - \frac{N_{\geq k}}{N} E(F_1 \geq k) \tag{10}$$

Where, the number $N_{<k}$ is the total number of samples in the left node of the parent node and $N_{\geq k}$ is the total number of samples in the right node. And the expression $E(F_1 < k)$ denotes the condition energy of the left node with all the sample where their feature F_1 satisfies the criterion: $F_1 < k$, it's the same story on the right side of the node.

The simplicity of the decision tree gives it very large variance on the model, which means that, depending on which features are chosen to split and the parameters, many different types of tree can be configured. In general, the construction of the optimal tree is an NP-Hard problem, hence usually heuristics are used for constructing the tree given a training sample.

However, the power of tree is manifested by a forest. The idea is to use bootstrap to train multiple tree and use the idea of bagging, where we either takes the average of the prediction from the group of trees trained from the data (Committee Method), or we weight their votes using their by penalizing trees that are more complex (Stacked Generalization). When the tree is a classifier, we simply take the votes to produce the predicted label.

Nonetheless, one of the major practical advantage of the Random Forest Classifier over the SVM is the speed of training, which comes from our observation. The algorithm allows the tree to be train concurrently, reducing the amount of time needed to train the Random Forest.

Logistic Regression, Softmax Regression

The logistic regression is a generalized linear model. It's derived from using the idea of a Maximal Likelihood Estimator and Linear Regression. However, under its most general form, it's presented as a softmax regression for multi-class classification tasks.

Under the multi-class classification, we consider the generalization problem with $\{x_i, y_i\}$ where $x_i \in \mathbb{R}^n$ and $y_i \in \{1, 2, \cdots, K\}$.

And the hypothesis of given an observation and a lits of parameters will be given in the form:

$$h_\Theta(x) = \frac{1}{\sum_{j=1}^{K} \exp(\theta_j^T x)} \begin{bmatrix} \exp(\theta_1^T x) \\ \exp(\theta_2^T x) \\ \vdots \\ \exp(\theta_K^T x) \end{bmatrix} \tag{11}$$

And the parameters are in the form of a matrix Θ where θ_i are the columns of the matrix and the matrix will be $x \times K$.

In practice, we take the logarithm and linearize the MLE, and gradient descend are usually used to compute the optimal parameters. Due to the fact that the model itself is a generalize linear model, it inherits regularization options from linear regression, such as L1, L2 or elastic nets.

Neuro Network

Performance Metric

To measure the performance of the models, confusion matrix is used. The confusion: $M_{i,j}$ the number of sample with label i are being misclassified as label j.

The matrix M exposes all the false positive and false negative value for each of the label. The false positive quantity (portion of relevant elements among all elements has been classified into this category) is given by:

$$\text{FalsePositive} = \frac{M - \text{diag(M)}}{M\mathbb{J}} \tag{12}$$

Where, the division is an element-wise operation. For each row of the confusion matrix, we take the sum on the rows and divides the sum of the non-diagonal elements by the sum of that entire row of the matrix. The notation $\mathbb{J}$ is the a vector consisting of all ones, having the same length as the height of the matrix M.

Similarly, the False Negative (The portion of elements that has been classified into this category and they are relevant) for each of the label is computed by:

$$\text{FalseNegative} = \frac{M^T - \text{diag}(M^T)}{M^T \mathbb{J}} \tag{13}$$

In addition, the overall accuracy of the model is computed via the sum of the diagonal elements on the confusion matrix over the sum of all the elements that are in the matrix.

The false positive and false negative rate are computed for each of the label and it serves as a sign of how much confusion is involved for each of the label for a given model. However, this is only used for models with a large amount of labels and under the case where there are limited number of labels, we made the choice to examine the confusion matrix manually.

3 Algorithm Implementation and Development

4 Scratch Paper Works for Overleaf

Algorithm 1: Algorithm 1: Splitting by Classes and Data Standardization

Input: ClassSize, TestSet, Classes, Shuffle(Optional)
Images, Labels = EMNIST Test set if TestSet else EMNIST Training set
Initialize: SelectedIndices
for L in Classes **do**
    Indices = np.where(Labels==L)
    Reshape Indices into a 1-D numpy array
    if Shuffle **then**
        Shuffle Indices
    end if
    Indices = Indices[min(Indices.size, ClassSize)]
    SelectedIndices.append(Indices)
end for
RowDataMatrix := Images[Indices, ::, ::]
RowDataMatrix := RowDataMatrix as type np.floats
Labels := Labels[Indices]
RowDataMatrix := RowDataMatrix/255
RowDataMatrix := RowDataMatrix - mean(RowDataMatrix, axis=0)
Return: RowDataMatrix, Labels

5 Computational Results

Figure 1: Figure 4: Projection of the PCA embeddings

Figure 2:

Algorithm 2: Algorithm 2: Subclass Classification

Input: AuxFunc, TestSet, ClassSize
Mapping, NewAxisLabels, NewLabels
Images, Labels := Images and labels from the EMNIST test set if TestSet else from EMNIST Training Set
TransFormedLabels = np.vectorize(Mapping)(Labels)
Initialize: IndicesChosen
for II in NewLabels **do**
    Idx = np.where(TransformedLabels = II)
    Reshaoe Idx so it's 1-d Numpy array
    Shuffle Idx
    Idx := Idx[:min(idx.size, classsize)]
    Merge Idx into: IndicesChosen
end for
Images := Images[IndicesChosen, ::, ::]
LabelsChosen := Labels[IndicesChosen]
DataMatrix := Reshape Images and standardize Images
Return: DataMatrix, Mapping(Labels), LabelsChosen, NewLabels, NewAxisLabels

Algorithm 3: Algorithm 3: Extracting Statistics from the Confusion Matrix

Input: PredictedLabels, ActualLabels
M := Get Confusion Matrix using sklearn using PredictedLabels, ActualLabels
OverAllAccuracy := sum(diag(M))/sum(M)
FalsePositive := sum(M - diag(M), axis=0)/sum(M, axis=0)
FalseNegative := sum(M-diag(M), axis=1)/sum(M, axis=1)
Sort the FalsePositive and FalseNegative in ascending order, with the labels
Plot the Confusion matrix M, and plot FalsePositive and FalseNegative in bar plot.

Algorithm 4: Algorithm 4: Training Models

TrainX, TrainY = Using Algorithm 1 orAlgorithm 2 to get the training, labels set.
TestX, TestY = Using Algorithm 1 orAlgorithm 2 to get the testing, labels set.
DimReduce := Get a model by fitting TrainX, TrainY data using the LDA or PCA in sklearn
EmbeddingsTrain := DimReduce.getEmbeddings(TrainX)
EmbeddingsTest := DimReduce.getEmbeddings(TestX)
Model := Initialize a Model using sklearn
Model := Train on EmbeddingsTrain
PredictedLabelsTrain = Model.predict(TrainEmbeddings)
PredictedLabelsTest = Model.predict(TestEmbeddings)